SDK V2.4 ------开发指南

文档编写信息：

编写版本	发布说明	编写人	编写日期	审核人	审核日期	批准人	批准日期
V1.0	基础版本， 包含常用c接口封装	SDK研发人员	2021年6月	陈志开	2021年6月	刘云鹤	2021年7月
V2.0	基础开发、系统能力、 应用支撑、 系统安全四大级划分， 丰富大量API内容	SDK研发人员	2022年6月	陈志开	2022年6月10日	刘云鹤	2022年7月
V2.2	完善SDK开发指南资料， 丰富系统能力和应用支撑 控件	SDK研发人员	2023年6月	陈志开	2023年6月10日	刘云鹤	2023年7月
V2.3	完善SDK开发配套组件， 例如doc、man手册、 symobls文件等， 增加少量控件	SDK研发人员	2023年1月	陈志开	2023年3月	刘云鹤	2023年3月
V2.4	增加系统配置管理模块， 扩展应用支撑控件， 精细化系统协议管控	SDK研发人员	2024年6月6日	-	-	-	-

1 概述

KylinSdk 自研开发者套件（以下简称 kysdk）是在 银河麒麟操作系统上，为生态建设与软件开发提供安全、可靠、快捷、稳定的开发者接口。相比于社区中其他的开发者套件或框架，kysdk 更加聚焦于解决麒麟桌面操作系统的兼容、适配、移植、优化等方面的问题。kysdk当前聚焦五大模块，包括桌面环境SDK、应用支撑 SDK、系统能力 SDK、基础开发 SDK，系统安全SDK，通用中间层，同时充分考虑 kysdk 的兼容性。

本文档旨在为开发者在 银河麒麟系统上进行应用开发时，提供一种高效查阅 kysdk 接口的声明和使用方法的方式。减少开发者在使用 kysdk 时的学习成本。

SDK 的整体模块介绍如下：

模块	描述
桌面环境 SDK	主要为应用开发提供与桌面环境有关的功能接口，统一封装常用功能提升开发效率， 并在基础功能更新时提升应用的兼容性和扩展性；
应用支撑 SDK	聚焦于应用显示层，为开发者提供麒麟扩展控件，向图形化应用提供图形化开发功能， 可使用统一的 UI 框架，进行应用窗口的管理以及与系统进行互动等， 降低应用开发与应用迁移学习成本；
系统能力 SDK	聚焦于为开发者提供更多系统能力，开发者可快速获取基础的系统、硬件信息、 当前的运行时信息等，提升开发效率，助力开发更聚焦实际业务内容；
基础开发 SDK	聚焦于应用开发过程中，为开发者提供日志管理、封装字符串处理等能力，提升开发效率；

模块	描述
系统安全 SDK	聚焦于系统安全能力，统筹规划与平衡系统安全能力和内生安全能力；
通用中间层	聚焦应用迁移、适配中， 为开发者提供本地资源权限的解决方案；

2 部署方式

- 银河麒麟系统使用自研开发者套件下载使用时，可直接通过 `apt` 进行安装:

```
$ sudo apt-get install libkysdk-base-dev libkysdk-system-dev libkysdk-desktop-dev libkysdk-security-dev
```

- 非银河麒麟系统使用自研开发者套件下载使用时，需提前添加 kysdk 源地址，安装步骤如下：

（1）在/etc/apt/sources.list.d/创建kysdk.list文件添加软件源：

```
deb http://archive.kylinos.cn/kylin/KYLIN-ALL developer-kits main restricted universe
```

（2）配置后更新源：

```
$ sudo apt update
```

3 系统能力 SDK

系统能力 SDK 中的每个软件包均表述了一个或一类系统能力，例如操作系统信息、硬件管理、系统内生安全策略、网络管理等；

安装命令：

```
# C接口软件包
$ sudo apt-get install libkysdk-system libkysdk-system-dev

# dbus接口软件包
$ sudo apt-get install libkysdk-system-dbus

# python接口软件包
$ sudo apt-get install libkysdk-system-python

# java接口软件包
$ sudo apt-get install libkysdk-system-java

# websocket接口软件包
$ sudo apt-get install libkysdk-system-javascript-websocket

# http接口软件包
$ sudo apt-get install libkysdk-system-javascript-http
```

3.1 系统信息

该层设计主要为应用提供与操作系统相关的功能接口，以自研、组合、封装三种方式，将与 OS 相关功能（如文件系统、硬件信息、通信等）以功能为角度重新实现；屏蔽系统差异、平台差异带来的开发复杂性与调试难度。

3.1.1 系统时钟

当系统时间在整分或系统时间被修改时，com.kylin.kysdk.TimeServer Dbus 服务会发出报时信号。

- **安装命令：**

```
$ sudo apt-get install libdbus-1-dev libdbus-glib-1-dev libkysdk-systime libkysdk-systime-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-systime
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKSYSTIME kysdk-systime)
target_include_directories(demo PRIVATE ${KYSDKSYSTIME_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKSYSTIME_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKSYSTIME_LIBRARIES})
```

3.1.1.1 设置/获取系统时间

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkydate.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkydate.so
```

- **dbus信息**
  - dbus 服务名称：com.kylin.kysdk.TimeServer
  - 路径名称：/com/kylin/kysdk/Timer
  - Interfaces：com.kylin.kysdk.TimeInterface
  - 信号：
    - 系统时间修改信号：TimeChangeSignal
    - 定时报时信号：TimeSignal
  - dbus服务名称：com.kylin.kysdk.DateServer
  - 路径名称：/com/kylin/kysdk/Date
  - Interfaces：com.kylin.kysdk.DateInterface
  - 信号：
    - 系统时间格式修改信号：DateSignal
    - 日期长格式修改信号：LongDateSignal
    - 日期短格式修改信号：ShortDateSignal
    - 时间格式修改信号：TimeSignal
- **子模块信息：**

> 系统时间修改报时(自1.2.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	void TimeChangeSignal(const char* t)	
描述	当系统时间被修改时，com.kylin.kysdk.TimeServerDbus服务会发出报时信号	
参数	t	当前时间 如： "2021/09/26 21:13:28"
返回值	无	无
备注	无	

> 系统时间整分报时(自1.2.0版本启用)

子模块	系统时钟
接口类型	C
原型	void TimeSignal(const char* t)

描述	当系统时间在整分时，com.kylin.kysdk.TimeServerDbus服务会发出报时信号	
参数	t	当前时间 如： "2021/09/26 21:07:00"
返回值	无	无
备注	无	

获取系统支持的日期格式(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char** kdk_system_get_dateformat();	
描述	获取系统支持的日期格式	
参数	无	无
返回值	char**	按照支持的日期格式返回当日日期;由 NULL 字符串表示结 尾;由 alloc 生成,需要被 kdk_date_freeall 回收。
备注	无	

回收字符串列表(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern inline void kdk_date_freeall(char **list);	
描述	回收字符串列表	
参数	char **	字符串列表
返回值	无	无
备注	无	

设置日期格式(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern int kdk_system_set_dateformat(char *date)	
描述	设置日期格式	
参数	date	通过kdk_system_get_dateformat接口获取到的格式中的一种
返回值	0	设置成功
	其它值	设置失败
备注	无	

设置 24 小时制格式(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern int kdk_system_set_24_timeformat()	
描述	设置 24 小时制格式	
参数	无	无

返回值	0	设置成功
	其它值	设置失败
备注	无	

设置 12 小时制格式(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern int kdk_system_set_12_timeformat()	
描述	设置 12 小时制格式	
参数	无	无
返回值	0	设置成功
	其它值	设置失败
备注	无	

获取当前的日期格式(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_get_now_dateformat()	
描述	获取当前的日期格式	
参数	无	无
返回值	char*	成功返回用户的当前日期格式
	NULL	获取失败
备注	无	

获取当前的时间格式(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_get_now_timeformat()	
描述	获取当前的时间格式	
参数	无	无
返回值	char*	成功返回用户的当前时间格式
	NULL	获取失败
备注	无	

获取当前星期接口(短格式)(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_shortweek()	
描述	获取当前星期接口(短格式)	
参数	无	无
返回值	char*	成功返回用户的当前星期
	NULL	获取失败
备注	无	

获取当前星期接口(长格式)(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_longweek()	
描述	获取当前星期接口(长格式)	
参数	无	无
返回值	char*	成功返回用户的当前星期
	NULL	获取失败
备注	无	

获取当前秒钟接口(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_second();	
描述	获取当前秒钟接口	
参数	无	无
返回值	char*	成功返回用户的当前的带秒钟数的时间，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

时间格式转换(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern kdk_dateinfo *kdk_system_tran_dateformat(stru ct tm *ptr)	
描述	时间格式转换	
参数	ptr	需要转换的时间
返回值	kdk_dateinfo	日期信息结构体 成员:date(char*);描述: 用户配置文件 中的格式的日期; 成员:time(char*);描述: 用户配置文件中的格式的时间; 成员:timesec(char*);描述: 用户配置文件中的格式的带秒钟时间。
	NULL	获取失败
备注	接口返回成功需要调用kdk_free_dateinfo接口释放	

释放由 kdk_system_tran_dateformat 返回的日期信息结构体(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern void kdk_free_dateinfo(kdk_dateinfo *date)	
描述	释放由 kdk_system_tran_dateformat 返回的日期信息结构体	

参数	date	由 kdk_system_tran_dateformat 返回的结构体 指针
返回值	无	无
备注	无	

获取当前时间(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_nowtime()	
描述	获取当前时间	
参数	无	无
返回值	char*	成功返回与用户设置时间格式相同的当前时间。例如:15:02
	NULL	获取失败
备注	无	

获取当前日期(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_nowdate()	
描述	获取当前日期	
参数	无	无
返回值	char*	成功返回与用户设置时间格式相同的当前时间。例如:2022/08/04
	NULL	获取失败
备注	无	

获取未登录时的时间,星期,日期(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern kdk_logn_dateinfo *kdk_system_logn_dateinfo(char *user)	
描述	获取未登录时的时间,星期,日期	
参数	user	用户名
返回值	kdk_logn_dateinfo	未登录日期信息结构体 成员:date(char*);描述: 登录配置文件中 的格式的日期; 成员:time(char*);描述: 登录配置文件中的格式的时间; 成 员:week(char*);描述: 当前星期
	NULL	获取失败
备注	接口返回成功需要调用kdk_system_logn_dateinfo接口释放	

释放由 kdk_system_logn_dateinfo 返回的日期信息结构体(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern void kdk_free_logn_dateinfo(kdk_logn_dateinfo *date)	
描述	释放由 kdk_system_logn_dateinfo 返回的日期信息结构体	
参数	date	由 kdk_system_logn_dateinfo 返回的结 构体指针
返回值	无	无
	无	无
备注	无	

转换工具箱时间(自2.0.0.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_gjx_time(char *date)	
描述	转换工具箱时间	
参数	date	工具箱日期
返回值	char*	配置文件中的格式的日期
	NULL	转换失败
备注	无	

设置日期长格式(自2.2.3.5版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern int kdk_system_set_long_dateformat(char *format);	
描述	设置日期长格式	
参数	format	长格式的一种
返回值	0	设置成功
	其它值	设置失败
备注	无	

设置日期短格式(自2.2.3.5版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern int kdk_system_set_short_dateformat(char *format);	
描述	设置日期短格式	
参数	format	短格式的一种
返回值	0	设置成功
	其它值	设置失败
备注	无	

获取当前的长格式日期(自2.2.3.5版本启用)

子模块	系统时钟

接口类型	C	
原型	extern char* kdk_system_get_longformat_date();	
描述	获取当前的长格式日期	
参数	无	无
返回值	char*	获取成功返回用户的当前长格式的日期
	NULL	获取失败
备注	无	

获取当前的短格式日期(自2.2.3.5版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_get_shortformat_date();	
描述	获取当前的短格式日期	
参数	无	无
返回值	char*	获取成功返回用户的当前短格式的日期
	NULL	获取失败
备注	无	

获取长格式(自2.2.3.5版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_get_longformat();	
描述	获取长格式	
参数	无	无
返回值	char*	获取成功返回用户设置的长格式,长格式有{yyyy MM dd,yy M d},默认yyyy MM dd 格式
	NULL	获取失败
备注	无	

获取短格式(自2.2.3.5版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_get_shortformat();	
描述	获取短格式	
参数	无	无
返回值	char*	获取成功返回用户设置的短格式,短格式有{yyyy/MM/dd,yy/M/d,yyyy-MM-dd,yy-M-d,yyyy.MM.dd,yy.M.d},默认 yyyy/MM/dd 格式
	NULL	获取失败
备注	无	

长格式转化(自2.2.3.5版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_longformat_transform(struct tm *ptr);	
描述	长格式转化	
参数	ptr	需要转化的时间
返回值	char*	获取成功返回用户设置的长格式形式输出转化的日期
	NULL	转化失败
备注	无	

短格式转化(自2.2.3.5版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_shortformat_transform(struct tm *ptr);	
描述	短格式转化	
参数	ptr	需要转化的时间
返回值	char*	获取成功返回用户设置的短格式形式输出转化的日期。如 2023/06/09
	NULL	转化失败
备注	无	

获取登录锁屏的时间,星期,长格式日期(自2.2.3.5版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern kdk_logn_dateinfo *kdk_system_login_lock_dateinfo(char *user);	
描述	获取登录锁屏的时间,星期,长格式日期	
参数	user	用户名
返回值	kdk_logn_dateinfo	date(char*);描述: 登录配置文件中的长格式的日期; time(char*);描述: 登录配置文件中的时间格式的时间; week(char*);描述: 长格式星期，例如：星期一
	NULL	获取失败
备注	接口返回成功需要调用kdk_system_logn_dateinfo接口释放	

时间格式转化(自2.2.3.5版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern kdk_timeinfo *kdk_system_timeformat_transform(struct tm *ptr);	
描述	时间格式转化	
参数	ptr	需要转化的时间
返回值	char*	成功返回用户设置的时间形式输出转化的时间
	NULL	转化失败

备注	无

释放由kdk_system_timeformat_transform返回的时间信息结构体(自2.2.3.5版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern void kdk_free_timeinfo(kdk_timeinfo *time);	
描述	释放由kdk_system_timeformat_transform返回的时间信息结构体	
参数	time	由kdk_system_timeformat_transform返回的结构体指针
返回值	无	无
	无	无
备注	无	

相对日期转换(自2.4.1.0版本启用)

子模块	系统时钟	
接口类型	C	
原型	extern char* kdk_system_tran_absolute_date(struct tm *ptr);	
描述	相对日期转换	
参数	ptr	日期
返回值	char*	成功返回字符串{今天，昨天}； 时间间隔大于两天返回绝对日期； 返回的字符串需要free释放
	NULL	转换失败
备注	无	

3.1.2 获取系统中的硬件信息

- **安装命令：**

```
sudo apt-get install libkysdk-hardware libkysdk-hardware-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-hardware
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKHARDWARE kysdk-hardware)
target_include_directories(demo PRIVATE ${KYSDKHARDWARE_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKHARDWARE_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKHARDWARE_LIBRARIES})
```

3.1.2.1 获取 CPU 信息

封装 C 接口获取 CPU 信息

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkycpu.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkyhw.so
```

- **子模块信息：**

获取 CPU 架构(自1.2.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern const char* kdk_cpu_get_arch()	
描述	获取 CPU 架构	
参数	无	无
返回值	const char*	架构信息，如"x86_64"； 返回的是const char*， 不要free
	NULL	获取失败
备注	无	

获取 CPU 制造厂商(自1.2.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern const char* kdk_cpu_get_vendor()	
描述	获取CPU制造商	
参数	无	无
返回值	const char*	制造厂商字符串，如 “GenuineIntel”； 返回的是const char*， 不要free
	NULL	获取失败
备注	无	

获取 CPU 型号(自1.2.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern const char* kdk_cpu_get_model()	
描述	获取CPU型号	
参数	无	无
返回值	const char*	CPU型号名称，如“Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz”；返回的是const char*，不要free
	NULL	获取失败
备注	无	

获取 CPU 额定主频(自1.2.0版本启用)

子模块	获取CPU信息

接口类型	C	
原型	extern const char* kdk_cpu_get_freq_MHz()	
描述	获取CPU额定主频	
参数	无	无
返回值	const char*	额定主频赫兹数，如“1794.742”，单位是MHz；返回的是const char*，不要free
	NULL	获取失败
备注	无	

获取 CPU 核心数量(自1.2.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern unsigned int kdk_cpu_get_corenums()	
描述	获取CPU核心数量	
参数	无	无
返回值	unsigned int	所有可用的CPU核心数量
	0	获取失败
备注	无	

获取 CPU 对虚拟化的支持(自1.2.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	const char* kdk_cpu_get_virt()	
描述	获取CPU对虚拟化的支持	
参数	无	无
返回值	const char*	若CPU支持虚拟化，则返回虚拟化技术，如“vmx”；返回的是const char*，不要free
	NULL	CPU不支持虚拟化
备注	无	

获取CPU线程数(自1.2.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern unsigned int kdk_cpu_get_process();	
描述	获取CPU线程数	
参数	无	无
返回值	unsigned int	cpu支持的线程数
	0	获取失败
备注	无	

获取CPU最大频率(自2.4.1.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern float kdk_cpu_get_max_freq_MHz();	
描述	获取CPU最大频率	
参数	无	无
返回值	float	cpu频率最大频率，如“2600.0000”，单位为MHz
	0	获取失败
备注	无	

获取CPU最小频率(自2.4.1.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern float kdk_cpu_get_min_freq_MHz();	
描述	获取CPU最小频率	
参数	无	无
返回值	float	cpu频率最小频率，如“1900.0000”，单位为MHz
	0	获取失败
备注	无	

获取CPU运行时间(自2.4.1.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern char* kdk_cpu_get_running_time();	
描述	获取CPU运行时间	
参数	无	无
返回值	char*	cpu运行时间，如“xx天xx小时xx分钟”； 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取CPU插槽(自2.4.1.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern unsigned int kdk_cpu_get_sockets();	
描述	获取CPU插槽	
参数	无	无
返回值	unsigned int	cpu插槽数量
	-1	获取失败
备注	无	

获取CPU L1缓存（数据）(自2.4.1.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern unsigned int kdk_cpu_get_L1d_cache();	
描述	获取CPU L1缓存（数据）	
参数	无	无
返回值	unsigned int	cpu L1缓存（数据）
	-1	获取失败
备注	无	

获取CPU L1缓存（指令）(自2.4.1.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern unsigned int kdk_cpu_get_L1i_cache();	
描述	获取CPU L1缓存（指令）	
参数	无	无
返回值	unsigned int	cpu L1缓存（指令）
	-1	获取失败
备注	无	

获取CPU L2缓存(自2.4.1.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern unsigned int kdk_cpu_get_L2_cache();	
描述	获取CPU L2缓存	
参数	无	无
返回值	unsigned int	cpu L2缓存
	-1	获取失败
备注	无	

获取CPU L3缓存(自2.4.1.0版本启用)

子模块	获取CPU信息	
接口类型	C	
原型	extern unsigned int kdk_cpu_get_L3_cache();	
描述	获取CPU L3缓存	
参数	无	无
返回值	unsigned int	cpu L3缓存
	-1	获取失败
备注	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

服务名称： com.kylin.kysdk.service

路径名称： /com/kylin/kysdk/cpuinfo

接口名称： com.kylin.kysdk.cpuinfo

- **python导入方法**

```
from kysdk import Cpuinfo
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.java.CpuMethod;
```

功能描述	接口类型	接口	入参	返回值
获取 CPU 架构	dbus	QString getCpuArch()	无	QString 返回值为CPU架构
	python	getCpuArch()->str	无	返回值为CPU架构
	websocket	cpuinfo.getCpuArch()	无	返回值为CPU架构
	http	http://127.0.0.1:8888/cpuinfo/getCpuArch	无	Json 返回值为CPU架构
	java	String getCpuArch()	无	String 返回值为CPU架构

功能描述	接口类型	接口	入参	返回值
获取CPU制造商	dbus	QString getCpuVendor()	无	QString 返回值为CPU制造厂商
	python	getCpuVendor()->str	无	返回值为CPU制造厂商
	websocket	cpuinfo.getCpuVendor()	无	返回值为CPU制造厂商
	http	http://127.0.0.1:8888/cpuinfo/getCpuVendor	无	Json 返回值为CPU制造厂商
	java	String getCpuVendor()	无	String 返回值为CPU制造厂商

功能描述	接口类型	接口	入参	返回值
获取CPU型号	dbus	QString getCpuModel()	无	QString 返回值为CPU型号
	python	getCpuModel()->str	无	返回值为CPU型号
	websocket	cpuinfo.getCpuModel()	无	返回值为CPU型号
	http	http://127.0.0.1:8888/cpuinfo/getCpuModel	无	Json 返回值为CPU型号
	java	String getCpuModel()	无	String 返回值为CPU型号

功能描述	接口类型	接口	入参	返回值
获取CPU额定主频	dbus	QString getCpuFreqMHz()	无	QString 返回值为CPU额定主频
	python	getCpuFreqMHz()->str	无	返回值为CPU额定主频
	websocket	cpuinfo.getCpuFreqMHz()	无	返回值为CPU额定主频
	http	http://127.0.0.1:8888/disk/getDiskTotalSizeMiB?diskname=parameter	无	Json 返回值为CPU额定主频
	java	String getCpuFreqMHz()	无	String 返回值为CPU额定主频

功能描述	接口类型	接口	入参	返回值
获取CPU核心数量	dbus	unsigned int getCpuCorenums()	无	unsigned int 返回值为CPU核心数量

	python	getCpuCorenums()->str	无	返回值为CPU核心数量
	websocket	cpuinfo.getCpuCorenums()	无	返回值为CPU核心数量
	http	http://127.0.0.1:8888/cpuinfo/getCpuCorenums	无	Json 返回值为CPU核心数量
	java	UInt32 getCpuCorenums()	无	UInt32 返回值为CPU核心数量

功能描述	接口类型	接口	入参	返回值
获取CPU对虚拟化的支持	dbus	QString getCpuVirt()	无	QString 返回值为CPU对虚拟化的支持
	python	getCpuVirt()->str	无	返回值为CPU对虚拟化的支持
	websocket	cpuinfo.getCpuVirt()	无	返回值为CPU对虚拟化的支持
	http	http://127.0.0.1:8888/cpuinfo/getCpuVirt	无	Json 返回值为CPU对虚拟化的支持
	java	String getCpuVirt()	无	String 返回值为CPU对虚拟化的支持

功能描述	接口类型	接口	入参	返回值
获取CPU线程数	dbus	unsigned int getCpuProcess()	无	unsigned int 返回值为CPU线程数
	python	getCpuProcess()->int	无	返回值为CPU线程数
	websocket	cpuinfo.getCpuProcess()	无	返回值为CPU线程数
	http	http://127.0.0.1:8888/cpuinfo/getCpuProcess	无	Json 返回值为CPU线程数
	java	UInt32 getCpuProcess()	无	UInt32 返回值为CPU线程数

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include "libkycpu.h"
#include <stdio.h>

int main()
{
    printf("架构: %s\n", kdk_cpu_get_arch());
    printf("生产厂商: %s\n", kdk_cpu_get_vendor());
    printf("CPU 型号: %s\n", kdk_cpu_get_model());
    printf("CPU 主频: %s MHz\n", kdk_cpu_get_freq_MHz());
    printf("CPU 单核核心数量: %u\n", kdk_cpu_get_corenums());
    printf("CPU 虚拟化支持: %s\n", kdk_cpu_get_virt());
    printf("CPU 线程数: %u\n", kdk_cpu_get_process());

    printf("CPU 最大频率: %0.2f MHz\n", kdk_cpu_get_max_freq_MHz());
    printf("CPU 最小频率: %0.2f MHz\n", kdk_cpu_get_min_freq_MHz());
    char *run_time = kdk_cpu_get_running_time();
    printf("CPU 运行时间: %s\n", run_time);
    free(run_time);
    printf("CPU 插槽: %d\n", kdk_cpu_get_sockets());
    printf("CPU L1缓存（数据）: %d\n", kdk_cpu_get_L1d_cache());
    printf("CPU L1缓存（指令）: %d\n", kdk_cpu_get_L1i_cache());
    printf("CPU L2缓存: %d\n", kdk_cpu_get_L2_cache());
    printf("CPU L3缓存: %d\n", kdk_cpu_get_L3_cache());
    return 0;
}
```

```python
#----------------------------python语言示例----------------------------
from kysdk import Cpuinfo
cpu = Cpuinfo()

# getCpuArch接口
cpu.getCpuArch()
# getCpuVendor接口
cpu.getCpuVendor()
# getCpuModel接口
cpu.getCpuModel()
# getCpuFreqMHz接口
cpu.getCpuFreqMHz()
# getCpuCorenums接口
cpu.getCpuCorenums()
# getCpuVirt接口
cpu.getCpuVirt()
# getCpuProcess接口
cpu.getCpuProcess()
```

```java
//----------------------------Java语言示例----------------------------
import kylin.kysdk.java.CpuMethod;
CpuMethod obj = new CpuMethod();
System.out.println("Arch: " + obj.getCpuArch());
System.out.println("Vendor: " + obj.getCpuVendor());
System.out.println("Model: " + obj.getCpuModel());
System.out.println("Freq: " + obj.getCpuFreqMHz());
System.out.println("Core: " + obj.getCpuCorenums());
System.out.println("Virt: " + obj.getCpuVirt());
System.out.println("Process: " + obj.getCpuProcess());
```

```javascript
//----------------------------websocket语言示例----------------------------
  var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        new QWebChannel(websocket,function(channel){
                var cpuinfo = channel.objects.cpuinfo;
                //返回信息接收
                cpuinfo.sendText.connect(function(message) {
                    ...
                });
                //获得CPU架构
                cpuinfo.getCpuArch();
                //获得CPU制造厂商
                cpuinfo.getCpuVendor();
                //获得CPU型号
                cpuinfo.getCpuModel();
                //获得CPU额定主频
                cpuinfo.getCpuFreqMHz();
                //获得CPU核心数量
                cpuinfo.getCpuCorenums();
                //对虚拟化的支持
                cpuinfo.getCpuVirt();
                //获得CPU线程数
                cpuinfo.getCpuProcess();
            }
        );
    }
```

```
//----------------------------http语言示例----------------------------
1.http://127.0.0.1:8888/cpuinfo/getCpuArch
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getCpuArch()信息
    }
2.http://127.0.0.1:8888/cpuinfo/getCpuVendor
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getCpuVendor()信息
    }
3.http://127.0.0.1:8888/cpuinfo/getCpuModel
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getCpuModel()信息
    }
4.http://127.0.0.1:8888/cpuinfo/getCpuFreqMHz
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getCpuFreqMHz()信息
    }
5.http://127.0.0.1:8888/cpuinfo/getCpuCorenums
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getCpuCorenums()信息
    }
6.http://127.0.0.1:8888/cpuinfo/getCpuVirt
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getCpuVirt()信息
    }
7.http://127.0.0.1:8888/cpuinfo/getCpuProcess
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getCpuProcess()信息
    }
```

3.1.2.2 获取网卡信息

封装 C 接口获取到网卡硬件信息。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkync.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkyhw.so
```

- **子模块信息：**

> 获取系统中所有的网卡(自1.2.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char** kdk_nc_get_cardlist()	
描述	获取系统中所有的网卡	
参数	无	无

返回值	char**	网卡名称列表， 由NULL字符串表示结尾； 由alloc生成， 需要被kdk_nc_freeall回收
	NULL	获取失败
备注	无	

检测指定网卡是否处于 UP 状态(自1.2.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_is_up(const char *nc)	
描述	检测指定网卡是否处于UP状态	
参数	nc	网卡名称，如eno1
返回值	1	Up状态
	0	Down状态
备注	无	

获取系统中当前处于 link up 状态的网卡列表(自1.2.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char** kdk_nc_get_upcards()	
描述	获取系统中当前处于 link up 状态的网卡列表	
参数	无	无
返回值	char**	网卡名称列表， 由NULL字符串表示结尾； 由alloc生成， 需要被kdk_nc_freeall回收
	NULL	获取失败
备注	无	

获取指定网卡的物理 MAC 地址(自1.2.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char* kdk_nc_get_phymac(const char *nc)	
描述	获取指定网卡的物理MAC地址	
参数	nc	网卡名称，如eno1
返回值	char*	物理MAC地址， 由alloc生成，应当被free
	NULL	网卡不存在或获取失败
备注	无	

获取指定网卡的第一个 IPv4 地址(自1.2.0版本启用)

子模块	获取网卡信息
接口类型	C
原型	extern char* kdk_nc_get_private_ipv4(const char *nc)
描述	获取指定网卡的第一个IPv4地址

参数	nc	网卡名称，如eno1
返回值	char*	IPv4地址，由alloc生成， 应当被free
	NULL	获取出错或无IP
备注	无	

获取指定网卡的所有 IPv4 地址(自2.0.0.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char** kdk_nc_get_ipv4(const char* nc)	
描述	获取指定网卡的所有IPv4地址	
参数	nc	网卡名称，如eno1
返回值	char**	IPv4地址列表， 以NULL表示结尾， 由alloc生成， 需要被kdk_nc_freeall回收
	NULL	获取出错
备注	无	

获取指定网卡的第一个 IPv6 地址(自1.2.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char* kdk_nc_get_private_ipv6(const char *nc)	
描述	获取指定网卡的第一个IPv6地址	
参数	nc	网卡名称，如eno1
返回值	char*	IPv6地址，由alloc生成， 应当被free
	NULL	获取出错或无IP
备注	无	

获取指定网卡的所有 IPv6 地址(自2.0.0.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char** kdk_nc_get_ipv6(const char *nc)	
描述	获取指定网卡的所有IPv6地址	
参数	nc	网卡名称，如eno1
返回值	char**	IPv6地址列表， 以NULL表示结尾， 由alloc生成， 需要被kdk_nc_freeall回收
	NULL	获取出错
备注	无	

获取指定网卡的有线/无线类型(自2.0.0.0版本启用)

子模块	获取网卡信息
接口类型	C

原型	extern int kdk_nc_is_wireless(const char *nc);	
描述	获取指定网卡的有线/无线类型	
参数	nc	网卡名称，如eno1
返回值	0	有线
	1	无线
备注	无	

获取指定网卡的厂商名称和设备型号(自2.0.0.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_vendor_and_product(const char *nc, char *vendor, char *product);	
描述	获取指定网卡的厂商名称和设备型号	
参数	nc	网卡名称，如eno1
	vendor	接受厂商名称的缓冲区
	product	接受设备型号的缓冲区
返回值	0	成功
	1	失败
备注	无	

获取指定网卡的驱动(自2.2.3.5版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char *kdk_nc_get_driver(const char *nc);	
描述	获取指定网卡的驱动	
参数	nc	网卡名称，如eno1
返回值	char*	成功返回网卡驱动， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

回收字符串列表(自1.2.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern inline void kdk_nc_freeall(char **ptr)	
描述	回收字符串列表	
参数	ptr	字符串列表
返回值	无	无
	无	无
备注	无	

获取指定网卡的广播地址(自2.4.1.0版本启用)

子模块	获取网卡信息	

接口类型	C	
原型	extern char *kdk_nc_get_broadAddr(const char *nc, const char *addr);	
描述	获取指定网卡的广播地址	
参数	nc	网卡名称，如eno1
	addr	ip地址
返回值	char*	成功返回网卡广播地址，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取指定网卡的子网掩码(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char *kdk_nc_get_netmask(const char *nc, const char *addr);	
描述	获取指定网卡的子网掩码	
参数	nc	网卡名称，如eno1
	addr	ip地址
返回值	char*	成功返回子网掩码，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取指定网卡的掩码长度(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_mask_length(int type, const char *nc, const char *addr);	
描述	获取指定网卡的子网掩码	
参数	type	1 ipv4，0 ipv6
	nc	网卡名称，如eno1
	addr	ip地址
返回值	int	网卡的掩码长度
	-1	获取失败
备注	无	

获取指定网卡的连接类型(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char *kdk_nc_get_conn_type(const char *nc);	
描述	获取指定网卡的连接类型	
参数	nc	网卡名称，如eno1

返回值	char*	成功返回网卡的连接类型，如"Ethernet",不需要free
	NULL	获取失败
备注	无	

获取wifi名称(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char *kdk_nc_get_wifi_name(const char *nc);	
描述	获取wifi名称	
参数	nc	网卡名称，如eno1
返回值	char*	成功返回wifi名称, 返回的字符串需要被 free 释放。
	NULL	获取失败
备注	无	

获取信号质量(连接 wifi 显示)(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_wifi_signal_qual(const char *nc);	
描述	获取信号质量(连接 wifi 显示)	
参数	nc	网卡名称，如eno1
返回值	int	成功返回信号质量
	-1	获取失败
备注	无	

获取信号强度(连接 wifi 显示)(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_wifi_signal_level(const char *nc);	
描述	获取信号强度(连接 wifi 显示)	
参数	nc	网卡名称，如eno1
返回值	int	成功返回信号强度
	-1	获取失败
备注	无	

获取底噪(连接 wifi 显示)(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_wifi_noise(const char *nc);	
描述	获取底噪(连接 wifi 显示)	
参数	nc	网卡名称，如eno1
返回值	int	成功返回底噪

	-1	获取失败
备注	无	

获取网卡速率(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char *kdk_nc_get_speed(const char *nc);	
描述	获取网卡速率	
参数	nc	网卡名称，如eno1
返回值	char*	成功返回网卡的速率，如"1000Mb/s", 返回的字符串需要被 free 释放。
	NULL	获取失败
备注	无	

获取指定网卡的接收包数量(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_rx_packets(const char *nc);	
描述	获取指定网卡的接收包数量	
参数	nc	网卡名称，如eno1
返回值	int	成功返回网卡的接收包数量
	-1	获取失败
备注	无	

获取指定网卡的总计接收(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_rx_bytes(const char *nc);	
描述	获取指定网卡的总计接收	
参数	nc	网卡名称，如eno1
返回值	int	成功返回网卡的总计接收, 单位为字节
	-1	获取失败
备注	无	

获取指定网卡的接收(错误包)(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_rx_errors(const char *nc);	
描述	获取指定网卡的接收(错误包)	
参数	nc	网卡名称，如eno1
返回值	int	成功返回网卡的接收 (错误包)数量

	-1	获取失败
备注	无	

获取指定网卡的接收(丢弃包) (自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_rx_dropped(const char *nc);	
描述	获取指定网卡的接收(丢弃包)	
参数	nc	网卡名称，如eno1
返回值	int	成功返回网卡的接收(丢弃包)数量
	-1	获取失败
备注	无	

获取指定网卡的接收FIFO(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_rx_fifo_errors(const char *nc);	
描述	获取指定网卡的接收FIFO	
参数	nc	网卡名称，如eno1
返回值	int	成功返回网卡的接收FIFO数量
	-1	获取失败
备注	无	

获取指定网卡的发送包数量(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_tx_packets(const char *nc);	
描述	获取指定网卡的发送包数量	
参数	nc	网卡名称，如eno1
返回值	int	成功返回网卡的发送包数量
	-1	获取失败
备注	无	

获取指定网卡的总计发送(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_tx_bytes(const char *nc);	
描述	获取指定网卡的总计发送	
参数	nc	网卡名称，如eno1
返回值	int	成功返回网卡的总计发送,单位为字节
	-1	获取失败
备注	无	

获取指定网卡的发送(错误包)(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_tx_errors(const char *nc);	
描述	获取指定网卡的发送(错误包)	
参数	nc	网卡名称，如eno1
返回值	int	成功返回网卡的发送(错误包)数量
	-1	获取失败
备注	无	

获取指定网卡的发送(丢弃包)(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_tx_dropped(const char *nc);	
描述	获取指定网卡的发送(丢弃包)	
参数	nc	网卡名称，如eno1
返回值	int	成功返回网卡的发送(丢弃包)数量
	-1	获取失败
备注	无	

获取指定网卡的发送FIFO(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_tx_fifo_errors(const char *nc);	
描述	获取指定网卡的发送FIFO	
参数	nc	网卡名称，如eno1
返回值	int	成功返回网卡的发送FIFO数量
	-1	获取失败
备注	无	

获取指定网卡的载波损耗(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_tx_carrier_errors(const char *nc);	
描述	获取指定网卡的载波损耗	
参数	nc	网卡名称，如eno1
返回值	int	成功返回网卡的载波损耗
	-1	获取失败
备注	无	

获取指定网卡的域(自2.4.1.0版本启用)

子模块	获取网卡信息

接口类型	C	
原型	extern int kdk_nc_get_scope(const char *nc, const char *addr);	
描述	获取指定网卡的域	
参数	nc	网卡名称，如eno1
	addr	ip地址
返回值	int	成功返回网卡的域
	-1	获取失败
备注	无	

获取网卡的uuid(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char* kdk_nc_get_uuid(char *nc);	
描述	获取网卡的uuid	
参数	nc	网卡名称，如eno1
返回值	char*	网卡的uuid， 返回的字符串需要被 free 释放。
	NULL	获取失败
备注	无	

获取网卡的链接状态(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern int kdk_nc_get_carrier(char *nc);	
描述	获取网卡的链接状态	
参数	nc	网卡名称，如eno1
返回值	int	网卡的链接状态
	-1	获取失败
备注	无	

获取网卡的工作模式(全双工/半双工)(自2.4.1.0版本启用)

子模块	获取网卡信息	
接口类型	C	
原型	extern char* kdk_nc_get_duplex(char *nc);	
描述	获取网卡的工作模式(全双工/半双工)	
参数	nc	网卡名称，如eno1
返回值	char*	网卡的工作模式， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

- **其他接口类型接口：**

- **引用方法**
- **dbus服务名**

> **服务名称：** com.kylin.kysdk.service
>
> **路径名称：** /com/kylin/kysdk/netcard
>
> **接口名称：** com.kylin.kysdk.netcard

- **python导入方法**

```
from kysdk import NetCard
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.java.NetCardMethod;
```

功能描述	接口类型	接口	入参	返回值
获取系统中所有的网卡	dbus	QStringList getNetCardName()	无	QStringList 返回值为所有网卡的名称
	python	getNetCardName()->list	无	返回值为所有网卡的名称
	websocket	netcard.getNetCardName()	无	返回值为所有网卡的名称
	http	http://127.0.0.1:8888/netcard/getNetCardName	无	Json 返回值为所有网卡的名称
	java	List getNetCardName()	无	List 返回值为所有网卡的名称

功能描述	接口类型	接口	入参	返回值
检测指定网卡是否处于UP状态	dbus	int getNetCardUp(const QString netCardName)	netCardName 网卡名称	int 返回值为网卡是否处于UP Down(0)/Up(1)
	python	getNetCardUp(netcard_name:str)->int	netcard_name 网卡名称	int 返回值为网卡是否处于UP Down(0)/Up(1)
	websocket	netcard.getNetCardUp(pid)	pid 网卡名称	返回值为网卡是否处于UP Down(0)/Up(1)
	http	http://127.0.0.1:8888/netcard/getNetCardUp?netcardname=parameter	parameter为网卡名称	Json 返回值为网卡是否处于UP Down(0)/Up(1)
	java	int getNetCardUp(String netCardName)	netCardName 网卡名称	int 返回值为网卡是否处于UP Down(0)/Up(1)

功能描述	接口类型	接口	入参	返回值
获取系统中当前处于link up 状态的网卡列表	dbus	QStringList getNetCardUpcards()	无	QString 返回值为系统中当前处于link up 状态的网卡列表
	python	getNetCardUpcards()->str	无	返回值为系统中当前处于link up 状态的网卡列表
	websocket	netcard.getNetCardUpcards()	无	返回值为系统中当前处于link up 状态的网卡列表
	http	http://127.0.0.1:8888/netcard/getNetCardUpcards	无	Json 返回值为系统中当前处于link up 状态的网卡列表
	java	List getNetCardUpcards()	无	String

				返回值为系统中当前处于 link up 状态的网卡列表

功能描述	接口类型	接口	入参	返回值
获取指定网卡的物理MAC地址	dbus	QString getNetCardPhymac(const QString netCardName)	netCardName 网卡名称	QString 返回值为网卡的物理M
	python	getNetCardPhymac(netcard_name)->str	netcard_name 网卡名称	返回值为网卡的物理M
	websocket	netcard.getNetCardPhymac(pid)	pid 网卡名称	返回值为网卡的物理M
	http	http://127.0.0.1:8888/netcard/getNetCardPhymac?netcardname=parameter	parameter为网卡名称	Json 返回值为网卡的物理M
	java	String getNetCardPhymac(String netCardName)	netCardName 网卡名称	String 返回值为网卡的物理M

功能描述	接口类型	接口	入参	返回
获取指定网卡的第一个IPv4地址	dbus	QString getNetCardPrivateIPv4(const QString netCardName)	netCardName 网卡名称	QStr 返回值为网卡的
	python	getNetCardPrivateIPv4(netcard_name)->str	netcard_name 网卡名称	返回值为网卡的
	websocket	netcard.getNetCardPrivateIPv4(pid)	pid 网卡名称	返回值为网卡的
	http	http://127.0.0.1:8888/netcard/getNetCardPrivateIPv4?netcardname=parameter	parameter为网卡名称	Jsc 返回值为网卡的
	java	String getNetCardPrivateIPv4(String netCardName)	netCardName 网卡名称	Stri 返回值为网卡的

功能描述	接口类型	接口	入参	返回值
获取指定网卡的所有IPv4地址	dbus	QStringList getNetCardIPv4(const QString netCardName)	netCardName 网卡名称	QStringList 返回值为网卡的所有IPv4地
	python	getNetCardIPv4(netcard_name)->list	netcard_name 网卡名称	返回值为网卡的所有IPv4地
	websocket	netcard.getNetCardIPv4(pid)	pid 网卡名称	返回值为网卡的所有IPv4地
	http	http://127.0.0.1:8888/netcard/getNetCardIPv4?netcardname=parameter	parameter为网卡名称	Json 返回值为网卡的所有IPv4地
	java	List getNetCardIPv4(String netCardName)	netCardName 网卡名称	List 返回值为网卡的所有IPv4地

功能描述	接口类型	接口	入参	返回
获取指定网卡的第一个IPv6地址	dbus	QString getNetCardPrivateIPv6(const QString netCardName)	netCardName 网卡名称	QStr 返回值为网卡的
	python	getNetCardPrivateIPv6(netcard_name)->str	netcard_name 网卡名称	返回值为网卡的
	websocket	netcard.getNetCardPrivateIPv6(pid)	pid 网卡名称	返回值为网卡的
	http	http://127.0.0.1:8888/netcard/getNetCardPrivateIPv6?netcardname=parameter	parameter为网卡名称	Jsc 返回值为网卡的
	java	String getNetCardPrivateIPv6(String netCardName)	netCardName 网卡名称	Stri 返回值为网卡的

功能描述	接口类型	接口	入参	返回值
获取指定网卡的有线/无线类型	dbus	int getNetCardType(const QString netCardName)	netCardName 网卡名称	int 返回值为网卡类型，有线(0)/无线(1)

	python	getNetCardType(netcard_name)->int	netcard_name 网卡名称	返回值为网卡类型， 有线(0)/无线(1)
	websocket	netcard.getNetCardType(pid)	pid 网卡名称	返回值为网卡类型， 有线(0)/无线(1)
	http	http://127.0.0.1:8888/netcard/getNetCardType? netcardname=parameter	parameter为网卡名称	Json 返回值为网卡类型， 有线(0)/无线(1)
	java	int getNetCardType(String netCardName)	netCardName 网卡名称	int 返回值为网卡类型， 有线(0)/无线(1)

功能描述	接口类型	接口	入参	返回值
获取指定网卡的厂商名称和设备型号	dbus	QStringList getNetCardProduct(const QString netCardName)	netCardName 网卡名称	QStringL 返回值为网卡厂
	python	getNetCardProduct(netcard_name)->list	netcard_name 网卡名称	返回值为网卡厂
	websocket	netcard.getNetCardProduct(pid)	pid 网卡名称	返回值为网卡厂
	http	http://127.0.0.1:8888/netcard/getNetCardProduct? netcardname=parameter	parameter为网卡名称	Json 返回值为网卡厂
	java	List getNetCardProduct(String netCardName)	netCardName 网卡名称	List 返回值为网卡厂

功能描述	接口类型	接口	入参	返回值
获取指定网卡的所有IPv6地址	dbus	QStringList getNetCardIPv6(const QString netCardName)	netCardName 网卡名称	QStringList 返回值为网卡的所有IPv6地
	python	getNetCardIPv6(netcard_name)->list	netcard_name 网卡名称	返回值为网卡的所有IPv6地
	websocket	netcard.getNetCardIPv6(pid)	pid 网卡名称	返回值为网卡的所有IPv6地
	http	http://127.0.0.1:8888/netcard/getNetCardIPv6? netcardname=parameter	parameter为网卡名称	Json 返回值为网卡的所有IPv6地
	java	List getNetCardIPv6(const QString netCardName)	netCardName 网卡名称	List 返回值为网卡的所有IPv6地

- **示例代码：**

```c
#----------------------------C语言示例----------------------------
#include "libkync.h"
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char **cards = kdk_nc_get_cardlist();
    size_t index = 0;
    printf("所有网卡: \n");
    while (cards[index])
    {
        char *mac = kdk_nc_get_phymac(cards[index]);
        char *ipv4 = kdk_nc_get_private_ipv4(cards[index]);
        char *ipv6 = kdk_nc_get_private_ipv6(cards[index]);
        char *driver = kdk_nc_get_driver(cards[index]);
        char vendor[256] = "\0", product[256] = "\0";
        kdk_nc_get_vendor_and_product(cards[index], vendor, product);
        printf("Card %zd: %s\tStatus: %s\tMac: %s\tIPv4: %s\tIPv6: %s\t Vendor: %s\t Product: %s\t Type: %s\t driver: %s\n",
               index + 1, cards[index], kdk_nc_is_up(cards[index]) == 1 ? "Up" : "Down",
               mac, ipv4, ipv6, vendor, product,
               kdk_nc_is_wireless(cards[index]) ? "wireless" : "ethernet", driver);

        if (mac)
            free(mac);
        if (ipv4)
            free(ipv4);
        if (ipv6)
            free(ipv6);
        if (driver)
            free(driver);

        char *uuid = kdk_nc_get_uuid(cards[index]);
        printf("%s\n", uuid ? uuid : "get uuid faile");
        free(uuid);

        printf("%d\n", kdk_nc_get_carrier(cards[index]));

        char *duplex = kdk_nc_get_duplex(cards[index]);
        printf("%s\n", duplex ? duplex : "get duplex failed");
        free(duplex);

        char *conn = kdk_nc_get_conn_type(cards[index]);
        printf("Conn type = %s\n", conn);

        char *wifi_name = kdk_nc_get_wifi_name(cards[index]);
        printf("wifi name = %s\n", wifi_name);
        free(wifi_name);

        char *speed = kdk_nc_get_speed(cards[index]);
        printf("speed = %s\n", speed);
        free(speed);

        int rx_packets = kdk_nc_get_rx_packets(cards[index]);
        printf("rx_packets = %d\n", rx_packets);

        int rx_bytes = kdk_nc_get_rx_bytes(cards[index]);
        printf("rx_bytes = %d\n", rx_bytes);

        int rx_errors = kdk_nc_get_rx_errors(cards[index]);
        printf("rx_errors = %d\n", rx_errors);

        int rx_dropped = kdk_nc_get_rx_dropped(cards[index]);
        printf("rx_dropped = %d\n", rx_dropped);

        int fifo_errors = kdk_nc_get_rx_fifo_errors(cards[index]);
        printf("fifo_errors = %d\n", fifo_errors);

        int frame_errors = kdk_nc_get_rx_frame_errors(cards[index]);
        printf("frame_errors = %d\n", frame_errors);
```

```
    int tx_packets = kdk_nc_get_tx_packets(cards[index]);
    printf("tx_packets = %d\n", tx_packets);

    int tx_bytes = kdk_nc_get_tx_bytes(cards[index]);
    printf("tx_bytes = %d\n", tx_bytes);

    int tx_errors = kdk_nc_get_tx_errors(cards[index]);
    printf("tx_errors = %d\n", tx_errors);

    int tx_dropped = kdk_nc_get_tx_dropped(cards[index]);
    printf("tx_dropped = %d\n", tx_dropped);

    int tx_fifo_errors = kdk_nc_get_tx_fifo_errors(cards[index]);
    printf("tx_fifo_errors = %d\n", tx_fifo_errors);

    int tx_carrier_errors = kdk_nc_get_tx_carrier_errors(cards[index]);
    printf("tx_carrier_errors = %d\n", tx_carrier_errors);

    int signal_qual = kdk_nc_get_wifi_signal_qual(cards[index]);
    printf("signal_qual = %d\n", signal_qual);

    int signal_level = kdk_nc_get_wifi_signal_level(cards[index]);
    printf("signal_level = %d\n", signal_level);

    int noise = kdk_nc_get_wifi_noise(cards[index]);
    printf("noise = %d\n", noise);

    char **list4 = kdk_nc_get_ipv4(cards[index]);
    int i = 0;
    printf("AllIpV4: ");
    while (list4 && list4[i])
    {
        printf("%s\t", list4[i]);

        char *boardaddr = kdk_nc_get_broadAddr(cards[index], list4[i]);
        printf("boardaddr = %s\n", boardaddr);
        free(boardaddr);

        char *netmask = kdk_nc_get_netmask(cards[index], list4[i]);
        printf("netmask = %s\n", netmask);
        free(netmask);

        free(list4[i++]);
    }
    printf("\n");
    char **list6 = kdk_nc_get_ipv6(cards[index]);
    printf("AllIpV6: ");
    i = 0;
    while (list6 && list6[i])
    {
        printf("%s\t", list6[i]);

        int len = kdk_nc_get_mask_length(0, cards[index], list6[i]);
        printf("netmask_length = %d\n", len);

        int scope = kdk_nc_get_scope(cards[index], list6[i]);
        printf("scope = %d\n", scope);

        free(list6[i++]);
    }
    printf("\n");

    free(list4);
    free(list6);
    index++;
}
kdk_nc_freeall(cards);

char **upcards = kdk_nc_get_upcards();
index = 0;
printf("工作网卡: \n");
```

```
    while (upcards[index])
    {
        printf("Card %zd: %s\n", index + 1, upcards[index]);
        index++;
    }
    kdk_nc_freeall(upcards);
    return 0;
}
```

```
#-----------------------------python语言示例-----------------------------
from kysdk import NetCard
netcard = NetCard()
#getNetCardName接口
netcard.getNetCardName()
#getNetCardUpcards接口
netcard.getNetCardUpcards()
#getNetCardUp接口
netcard.getNetCardUp(netcard_name)
#getNetCardPhymac接口
netcard.getNetCardPhymac(netcard_name)
#getNetCardPrivateIPv4接口
netcard.getNetCardPrivateIPvrd_name)
#getNetCardIPv4接口
netcard.getNetCardIPv4(netcard_name)
#getNetCardPrivateIPv6接口
netcard.getNetCardPrivateIPvrd_name)
#getNetCardType接口
netcard.getNetCardType(netcard_name)
#getNetCardProduct接口
netcard.getNetCardProduct(netcard_name)
#getNetCardIPv6接口
netcard.getNetCardIPv6(netcard_name)
```

```
//-----------------------------Java语言示例-----------------------------
import kylin.kysdk.java.NetCardMethod;
NetCardMethod obj = new NetCardMethod();
List<String> list = obj.getNetCardName();
System.out.println("CardUp: " + obj.getNetCardUp(list.get(0)));
System.out.println("UpCards: " + obj.getNetCardUpcards());
System.out.println("Mac: " + obj.getNetCardPhymac(list.get(0)));
System.out.println("PrivateIPv4: " + obj.getNetCardPrivateIPv4(list.get(0)));
System.out.println("Ipv4: " + obj.getNetCardIPv4(list.get(0)));
System.out.println("PrivateIPv6: " + obj.getNetCardPrivateIPv6(list.get(0)));
System.out.println("Type: " + obj.getNetCardType(list.get(0)));
System.out.println("Product: " + obj.getNetCardProduct(list.get(0)));
System.out.println("Ipv6: " + obj.getNetCardPrivateIPv6(list.get(0)));
```

```
//----------------------------websocket语言示例----------------------------
    //pid 网卡名称
    var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        new QWebChannel(websocket,function(channel){
            var netcard = channel.objects.netcard;
            //返回信息接收
            netcard.sendText.connect(function(message) {
                ...
            });
            //获取网卡名称
            netcard.getNetCardName();
            //获取网卡的UP状态
            netcard.getNetCardUp(pid);
            //获取处于link up状态的网卡列表
            netcard.getNetCardUpcards();
            //获取网卡的物理MAC地址
            netcard.getNetCardPhymac(pid);
            //获取网卡的第一个IPv4地址
            netcard.getNetCardPrivateIPv4(pid);
            //获取网卡的所有IPv4地址
            netcard.getNetCardIPv4(pid);
            //获取网卡的第一个IPv6地址
            netcard.getNetCardPrivateIPv6(pid);
            //获取网卡类型
            netcard.getNetCardType(pid);
            //获取网卡厂商型号
            netcard.getNetCardProduct(pid);
            //获取网卡的所有IPv6地址
            netcard.getNetCardIPv6(pid);
            }
        );
    }
```

```
//-----------------------------http语言示例-----------------------------
//parameter 为网卡名称参数
1.http://127.0.0.1:8888/netcard/getNetCardName
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getNetCardName()信息
    }
2.http://127.0.0.1:8888/netcard/getNetCardUp?netcardname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getNetCardUp(parameter)信息
    }
3.http://127.0.0.1:8888/netcard/getNetCardUpcards
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getNetCardUpcards()信息
    }
4.http://127.0.0.1:8888/netcard/getNetCardPhymac?netcardname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getNetCardPhymac(parameter)信息
    }
5.http://127.0.0.1:8888/netcard/getNetCardPrivateIPv4?netcardname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getNetCardPrivateIPv4(parameter)信息
    }
6.http://127.0.0.1:8888/netcard/getNetCardIPv4?netcardname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getNetCardIPv4(parameter)信息
    }
7.http://127.0.0.1:8888/netcard/getNetCardPrivateIPv6?netcardname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getNetCardPrivateIPv6(parameter)信息
    }
8.http://127.0.0.1:8888/netcard/getNetCardType?netcardname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getNetCardType(parameter)信息
    }
9.http://127.0.0.1:8888/netcard/getNetCardProduct?netcardname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getNetCardProduct(parameter)信息
    }
10.http://127.0.0.1:8888/netcard/getNetCardIPv6?netcardname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getNetCardIPv6(parameter)信息
    }
```

3.1.2.3 获取 bios 信息

封装 C 接口获取到 bios 硬件信息。

- **头文件路径:**

```
#include "kysdk/kysdk-system/libkybiosinfo.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkyhw.so
```

- **子模块信息：**

获取 bios 厂商信息(自2.0.0.0版本启用)

子模块	获取 bios 信息	
接口类型	C	
原型	extern const char *kdk_bios_get_vendor();	
描述	获取bios厂商信息	
参数	无	无
返回值	char*	成功返回bios厂商， 返回的字符串需要被kdk_bios_free释放
	NULL	获取失败
备注	无	

获取 bios 版本信息(自2.0.0.0版本启用)

子模块	获取 bios 信息	
接口类型	C	
原型	extern const char *kdk_bios_get_version();	
描述	获取bios版本信息	
参数	无	无
返回值	char*	成功返回bios版本， 返回的字符串需要被kdk_bios_free释放
	NULL	获取失败
备注	无	

释放内存(自2.0.0.0版本启用)

子模块	获取 bios 信息	
接口类型	C	
原型	extern void kdk_bios_free(char* info);	
描述	释放获取信息接口申请的内存	
参数	info	获取bios信息接口返回的指针
返回值	无	无
	无	无
备注	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

服务名称： com.kylin.kysdk.service

路径名称： /com/kylin/kysdk/bios

接口名称： com.kylin.kysdk.bios

- **python导入方法**

```
from kysdk import Bios
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.java.BiosMethod;
```

功能描述	接口类型	接口	入参	返回值
获取bios厂商信息	dbus	QString getBiosVendor()	无	QString 返回值为BIOS厂商
	python	getBiosVendor()->str	无	返回值为BIOS厂商
	websocket	bios.getBiosVendor()	无	返回值为BIOS厂商
	http	http://127.0.0.1:8888/bios/getBiosVendor	无	Json 返回值为BIOS厂商
	java	String getBiosVendor()	无	String 返回值为BIOS厂商

功能描述	接口类型	接口	入参	返回值
获取bios版本信息	dbus	QString getBiosVersion()	无	QString 返回值为BIOS版本号
	python	getBiosVersion()->str	无	返回值为BIOS版本号
	websocket	bios.getBiosVersion()	无	返回值为BIOS版本号
	http	http://127.0.0.1:8888/bios/getBiosVersion	无	Json 返回值为BIOS版本号
	java	String getBiosVersion()	无	String 返回值为BIOS版本号

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include "stdio.h"
#include "libkybios.h"
#include <stdlib.h>
int main()
{
    char *vendor = kdk_bios_get_vendor();
    char *version = kdk_bios_get_version();
    printf("vendor: %s", vendor);
    printf("version: %s",version);
    kdk_bios_free(vendor);
    kdk_bios_free(version);
    return 0;
}
```

```
#----------------------------python语言示例----------------------------
from kysdk import Bios
bios = Bios()
# getBiosVendor接口
bios.getBiosVendor()
# getBiosVersion接口
bios.getBiosVersion()
```

```
//----------------------------Java语言示例----------------------------
import kylin.kysdk.java.BiosMethod;
    BiosMethod obj = new BiosMethod();
    System.out.println("Vendor" + obj.getBiosVendor());
    System.out.println("Version" + obj.getBiosVersion());
```

```
//----------------------------websocket语言示例----------------------------
        var websocket_url = 'ws://localhost:12345';
        var websocket = null;

        if (websocket === null) {
            websocket = new WebSocket(websocket_url);
            websocket.onopen = function () {
                console.log("connect websocketserver success");
            }
        } else {
            websocket.close();
            websocket = null;
        }
        function xxx() {
            new QWebChannel(websocket,function(channel){
                 var bios = channel.objects.bios;
                 //返回信息接收
                 bios.sendText.connect(function(message) {
                     ...
                 });
                 //获取BIOS厂商
                 bios.getBiosVendor();
                 //获取BIOS版本号
                 bios.getBiosVersion();
                }
            );
        }
```

```
//----------------------------http语言示例----------------------------
1.http://127.0.0.1:8888/bios/getBiosVendor
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getBiosVendor()信息
    }
2.http://127.0.0.1:8888/bios/getBiosVersion
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getBiosVersion()信息
    }
```

3.1.2.4 获取主板信息

封装 C 接口获取到主板硬件信息。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkyboardinfo.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkyhw.so
```

- **子模块信息：**

> 获取主板产品名称信息(自2.0.0.0版本启用)

子模块	获取主板信息
接口类型	C

原型	extern const char *kdk_board_get_name();	
描述	获取主板型号	
参数	无	无
返回值	char*	成功返回主板型号，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取主板生产日期(自2.0.0.0版本启用)

子模块	获取主板信息	
接口类型	C	
原型	extern const char *kdk_board_get_date();	
描述	获取主板生产日期	
参数	无	无
返回值	char*	成功返回主板生产日期，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取主板序列号(自2.0.0.0版本启用)

子模块	获取主板信息	
接口类型	C	
原型	extern const char *kdk_board_get_serial();	
描述	获取主板序列号	
参数	无	无
返回值	char*	成功返回主板序列号，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取主板厂商信息(自2.0.0.0版本启用)

子模块	获取主板信息	
接口类型	C	
原型	extern const char *kdk_board_get_vendor();	
描述	获取主板厂商信息	
参数	无	无
返回值	char*	成功返回主板厂商信息，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

释放内存(自2.0.0.0版本启用)

子模块	获取主板信息	
接口类型	C	
原型	extern void kdk_board_free(char* info);	
描述	释放获取信息接口申请的内存	
参数	info	获取主板信息接口返回的指针
返回值	无	无
	无	无
备注	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

服务名称： com.kylin.kysdk.service

路径名称： /com/kylin/kysdk/mainboard

接口名称： com.kylin.kysdk.mainboard

- **python导入方法**

```
from kysdk import Mainboard
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.java.MainBoardMethod;
```

功能描述	接口类型	接口	入参	返回值
获取主板型号	dbus	QString getMainboardName()	无	QString 返回值为主板型号
	python	getMainboardName()->str	无	返回值为主板型号
	websocket	mainboard.getMainboardName()	无	返回值为主板型号
	http	http://127.0.0.1:8888/mainboard/getMainboardName	无	Json 返回值为主板型号
	java	String getMainboardName()	无	String 返回值为主板型号

功能描述	接口类型	接口	入参	返回值
获取主板生产日期	dbus	QString getMainboardDate()	无	QString 返回值为主板发布日期
	python	getMainboardDate()->str	无	返回值为主板发布日期
	websocket	mainboard.getMainboardDate()	无	返回值为主板发布日期
	http	http://127.0.0.1:8888/mainboard/getMainboardDate	无	Json 返回值为主板发布日期
	java	String getMainboardDate()	无	String 返回值为主板发布日期

功能描述	接口类型	接口	入参	返回值
获取主板序列号	dbus	QString getMainboardSerial()	无	QString 返回值为主板序列号
	python	getMainboardSerial()->str	无	返回值为主板序列号
	websocket	mainboard.getMainboardSerial()	无	返回值为主板序列号

	http	http://127.0.0.1:8888/mainboard/getMainboardSerial	无	Json 返回值为主板序列号
	java	String getMainboardSerial()	无	String 返回值为主板序列号

功能描述	接口类型	接口	入参	返回值
获取主板厂商信息	dbus	QString getMainboardVendor()	无	QString 返回值为主板厂商
	python	getMainboardVendor()->str	无	返回值为主板厂商
	websocket	mainboard.getMainboardVendor()	无	返回值为主板厂商
	http	http://127.0.0.1:8888/mainboard/getMainboardVendor	无	Json 返回值为主板厂商
	java	String getMainboardVendor()	无	String 返回值为主板厂商

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include "stdio.h"
#include "libkyboard.h"

int main()
{

    char *name = kdk_board_get_name();
    char *vendor = kdk_board_get_vendor();
    char *date = kdk_board_get_date();
    char *serial = kdk_board_get_serial();
    printf("name : %s", name);
    printf("vendor : %s", vendor);
    printf("date : %s", date);
    printf("serial : %s", serial);
    kdk_board_free(name);
    kdk_board_free(vendor);
    kdk_board_free(date);
    kdk_board_free(serial);
    return 0;
}
```

```
#----------------------------python语言示例----------------------------
from kysdk import Mainboard
mainboard = Mainboard()
# getMainboardName接口
mainboard.getMainboardName()
# getMainboardDate接口
mainboard.getMainboardDate()
# getMainboardSerial接口
mainboard.getMainboardSerial()
# getMainboardVendor接口
mainboard.getMainboardVendor()
```

```
//----------------------------Java语言示例----------------------------
import kylin.kysdk.java.MainBoardMethod;
MainBoardMethod obj = new MainBoardMethod();
System.out.println("Name: " + obj.getMainboardName());
System.out.println("date: " + obj.getMainboardDate());
System.out.println("Serial: " + obj.getMainboardSerial());
System.out.println("Vendor: " + obj.getMainboardVendor());
```

```
//----------------------------websocket语言示例----------------------------
        var websocket_url = 'ws://localhost:12345';
        var websocket = null;

        if (websocket === null) {
            websocket = new WebSocket(websocket_url);
            websocket.onopen = function () {
                console.log("connect websocketserver success");
            }
        } else {
            websocket.close();
            websocket = null;
        }
        function xxx() {
            new QWebChannel(websocket,function(channel){
                 var mainboard = channel.objects.mainboard;
                 //返回信息接收
                 mainboard.sendText.connect(function(message) {
                     ...
                 });
                 //获取主板型号
                 mainboard.getMainboardName();
                 //获取发布日期
                 mainboard.getMainboardDate();
                 //获取主板序列号
                 mainboard.getMainboardSerial();
                 //获取主板厂商
                 mainboard.getMainboardVendor();
                }
            );
        }
```

```
//----------------------------http语言示例----------------------------
1.http://127.0.0.1:8888/mainboard/getMainboardName
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMainboardName()信息
    }
2.http://127.0.0.1:8888/mainboard/getMainboardDate
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMainboardDate()信息
    }
3.http://127.0.0.1:8888/mainboard/getMainboardSerial
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMainboardSerial()信息
    }
4.http://127.0.0.1:8888/mainboard/getMainboardVendor
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMainboardVendor()信息
    }
```

3.1.2.5获取 usb 设备信息

封装 C 接口获取到 usb 设备硬件信息。

- **头文件路径：**

```
 #include "kysdk/kysdk-system/libkyusb.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkyhw.so
```

- **子模块信息：**

获取所有 usb 设备信息(自2.0.0.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern pDriverInfoList kdk_usb_get_list();	
描述	获取所有usb设备信息	
参数	无	无
返回值	pDriverInfoList	usb设备信息的结构体指针，具体信息自取
	NULL	获取失败

备注	typedef struct driver_info
	{
	char name[32]; //名称
	char type[2 + 1]; //类型
	char pid[4 + 1]; //设备pid
	char vid[4 + 1]; //设备vid
	char serialNo[32]; //序列号
	char devNode[32]; //路径
	} DriverInfo, *pDriverInfo;
	
	typedef struct driver_info_list
	{
	struct driver_info* info;
	struct driver_info* next;
	} *pDriverInfoList;
	接口返回成功需要调用kdk_usb_free接口释放	

释放内存(自2.0.0.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern void kdk_usb_free(pDriverInfoList list);	
描述	释放获取信息时申请的内存	
参数	list	kdk_usb_get_list() 返回的指针
返回值	无	无
	无	无
备注	无	

获取usb设备的总线信息(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern pUsbInfo kdk_usb_get_bus_info();	
描述	获取usb设备的总线地址和设备地址	
参数	无	无
返回值	pUsbInfo	usb设备总线地址和设备地址结构体
	NULL	获取失败

备注	typedef struct _UsbInfo
	{
	unsigned short busNum; //总线地址
	unsigned short devNum; //设备地址
	struct _UsbInfo* next; //链表next指针
	}UsbInfo, *pUsbInfo;
	接口返回成功需要调用kdk_usb_free_usb_info接口释放

usb硬盘类设备是否挂载(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern int kdk_usb_get_mount(int busNum, int devAddress);	
描述	usb硬盘类设备是否挂载	
参数	busNum	总线地址
	devAddress	设备地址
返回值	1	true 已挂载
	0	false 未挂载
备注	无	

获取usb设备的产品信息(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern char* kdk_usb_get_productName(int busNum, int devAddress);	
描述	获取usb设备的产品信息	
参数	busNum	总线地址
	devAddress	设备地址
返回值	char*	usb设备的产品信息， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取usb设备的厂商信息(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern char* kdk_usb_get_manufacturerName(int busNum, int devAddress);	
描述	获取usb设备厂商信息	
参数	busNum	总线地址
	devAddress	设备地址
返回值	char*	usb设备的厂商信息， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取usb设备的版本(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern char* kdk_usb_get_version(int busNum, int devAddress);	
描述	获取usb设备的版本	
参数	busNum	总线地址
	devAddress	设备地址
返回值	char*	usb设备的版本， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取usb设备的设备类(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern int kdk_usb_get_deviceClass(int busNum, int devAddress);	
描述	获取usb设备的设备类	
参数	busNum	总线地址
	devAddress	设备地址
返回值	int	usb设备的设备类
	小于0	获取失败
备注	无	

获取usb设备的设备子类(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern int kdk_usb_get_deviceSubClass(int busNum, int devAddress);	
描述	获取usb设备的设备子类	
参数	busNum	总线地址
	devAddress	设备地址
返回值	int	usb设备的设备子类
	小于0	获取失败
备注	无	

获取usb设备的协议码(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern int kdk_usb_get_deviceProtocol(int busNum, int devAddress);	
描述	获取USB设备的协议码	

参数	busNum	总线地址
	devAddress	设备地址
返回值	int	USB设备的协议码
	小于0	获取失败
备注	无	

获取usb设备的协商连接速度(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern char* kdk_usb_get_speed(int busNum, int devAddress);	
描述	获取usb设备的协商连接速度	
参数	busNum	总线地址
	devAddress	设备地址
返回值	char*	usb设备的协商连接速度，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

取usb设备的设备描述符(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern pUsbDeviceDescriptor kdk_usb_get_device_descriptor(int busNum, int devAddress);	
描述	获取usb设备的设备描述符	
参数	busNum	总线地址
	devAddress	设备地址
返回值	pUsbDeviceDescriptor	usb设备的设备描述符
	NULL	获取失败

备注	typedef struct _UsbDeviceDescriptor
	{
	unsigned short lenth; //描述符的字节数，恒为18
	unsigned short descriptorType; //描述符类型，恒为0x01
	unsigned short maxPacketSize0; //端点0的最大数据包大小
	unsigned short numConfigurations; //可能的配置数量
	}UsbDeviceDescriptor, *pUsbDeviceDescriptor;
	接口返回成功需要调用kdk_usb_free_usb_device_descriptor接口释放	

释放kdk_usb_get_bus_info返回(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern void kdk_usb_free_usb_info(pUsbInfo list);	
描述	释放kdk_usb_get_bus_info返回	
参数	list	kdk_usb_get_device_descriptor返回返回的指针
返回值	无	无

	无	无
无	无	

释放kdk_usb_get_device_descriptor返回(自2.4.1.0版本启用)

子模块	获取usb设备信息	
接口类型	C	
原型	extern void kdk_usb_free_usb_device_descriptor(pUsbDeviceDescriptor *descriptor);	
描述	释放kdk_usb_get_device_descriptor返回	
参数	descriptor	kdk_usb_get_device_descriptor返回返回的指针
返回值	无	无
	无	无
无	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

服务名称： com.kylin.kysdk.service

路径名称： /com/kylin/kysdk/peripheralsenum

接口名称： com.kylin.kysdk.peripheralsenum

- **python导入方法**

```
from kysdk import Peripheralsenum
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.PeripheralsenumMethod;
```

功能描述	接口类型	接口	入参	返回值
获取所有usb设备信息	dbus	QStringList getAllUsbInfo()	无	QStringList 返回值为所有usb设备的名称、类型、PID、VID、序列号、设备节点（若没有对应信息，输出null）
	python	getAllUsbInfo()->list	无	返回值为所有usb设备的名称、类型、PID、VID、序列号、设备节点（若没有对应信息，输出null）
	websocket	peripheralsenum.getAllUsbInfo()	无	返回值为所有usb设备的名称、类型、PID、VID、序列号、设备节点（若没有对应信息，输出null）
	http	http://127.0.0.1:8888/peripheralsenum/getAllUsbInfo	无	Json 返回值为所有usb设备的名称、类型、PID、VID、序列号、设备节点（若没有对应信息，输出null）
	java	List getAllUsbInfo()	无	List 返回值为所有usb设备的名称、类型、PID、VID、序列号、设备节点（若没有对应信息，输出null）

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include "stdio.h"
#include "libkyusb.h"

#define BUS_NUM 1
#define DEV_NUM 5

int main()
{
    pDriverInfoList list = kdk_usb_get_list();
    if(list)
    {
        pDriverInfoList tmp = list;
        while (tmp)
        {
            printf("Name: %s\t",tmp->data->name);
            printf("Class: %s\t",tmp->data->type);
            printf("PID: %s\t",tmp->data->pid);
            printf("VID: %s\t",tmp->data->vid);
            printf("Serial: %s\t",tmp->data->serialNo);
            printf("DevNode: %s\n",tmp->data->devNode);
            tmp = tmp->next;
        }

    }
    kdk_usb_free(list);

    pUsbInfo usb_info = kdk_usb_get_bus_info();
    pUsbInfo tmp = usb_info;
    while(tmp)
    {
        printf("%03u:%03u\n", tmp->busNum, tmp->devNum);
        tmp = tmp->next;
    }
    printf("%d\n", kdk_usb_get_mount(BUS_NUM,DEV_NUM));
    printf("%s\n", kdk_usb_get_productName(BUS_NUM,DEV_NUM));
    printf("%s\n", kdk_usb_get_manufacturerName(BUS_NUM,DEV_NUM));
    printf("%s\n", kdk_usb_get_version(BUS_NUM,DEV_NUM));
    printf("%d\n", kdk_usb_get_deviceClass(BUS_NUM,DEV_NUM));
    printf("%d\n", kdk_usb_get_deviceSubClass(BUS_NUM,DEV_NUM));
    printf("%d\n", kdk_usb_get_deviceProtocol(BUS_NUM,DEV_NUM));
    printf("%s\n", kdk_usb_get_speed(BUS_NUM,DEV_NUM));
    pUsbDeviceDescriptor desc = kdk_usb_get_device_descriptor(BUS_NUM,DEV_NUM);
    printf("%d %d %d %d\n",desc->lenth,desc->descriptorType, desc->maxPacketSize0, desc->numConfigurations);
    return 0;
}
```

```
#----------------------------python语言示例----------------------------
from kysdk import Peripheralsenum
peri = Peripheralsenum()
# getAllUsbInfo接口
peri.getAllUsbInfo()
```

```
//----------------------------Java语言示例----------------------------
import kylin.kysdk.PeripheralsenumMethod
PeripheralsenumMethod obj = new PeripheralsenumMethod();
System.out.println("Usb: " + obj.getAllUsbInfo());
```

```
//----------------------------websocket语言示例----------------------------
    var peripheralsenum = channel.objects.peripheralsenum;
    //返回信息接收
    peripheralsenum.sendText.connect(function(message) {
        ...
    });
    //获取外设设备信息
    peripheralsenum.getAllUsbInfo();
```

```
//----------------------------http语言示例----------------------------
1.http://127.0.0.1:8888/peripheralsenum/getAllUsbInfo
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getAllUsbInfo()信息
    }
```

3.1.2.6获取蓝牙设备信息

封装 C 接口获取到蓝牙设备硬件信息。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkybluetooth.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkybluetooth.so
```

- **子模块信息：**

> 获取蓝牙的设备id(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息	
接口类型	C	
原型	extern int** kdk_bluetooth_get_device_id();	
描述	获取蓝牙的设备id	
参数	无	无
返回值	int **	蓝牙的设备id
	0	获取失败
无	无	

> 获取蓝牙的制造商(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息	
接口类型	C	
原型	extern char* kdk_bluetooth_get_manufacturer(int num);	
描述	获取蓝牙的制造商	
参数	num	蓝牙设备的个数， 从0开始，0代表1个， 以此类推
返回值	char*	成功返回蓝牙的制造商， 返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取蓝牙的设备版本(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息	
接口类型	C	
原型	extern char* kdk_bluetooth_get_dev_version(int num);	
描述	获取蓝牙的设备版本	
参数	num	蓝牙设备的个数， 从0开始，0代表1个， 以此类推
返回值	char*	成功返回蓝牙的设备版本， 返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取蓝牙的名称(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息	
接口类型	C	
原型	extern char* kdk_bluetooth_get_name(int id);	
描述	获取蓝牙的名称	
参数	id	蓝牙的设备id
返回值	char*	成功返回蓝牙的名称， 返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取蓝牙的地址(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息	
接口类型	C	
原型	extern char* kdk_bluetooth_get_address(int id);	
描述	获取蓝牙的地址	
参数	id	蓝牙的设备id
返回值	char*	成功返回蓝牙的地址， 返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取蓝牙的连接模式(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息	
接口类型	C	
原型	extern char* kdk_bluetooth_get_link_mode(int id);	
描述	获取蓝牙的连接模式	
参数	id	蓝牙的设备id
返回值	char*	成功返回蓝牙的链接模式，

		返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取蓝牙的连接策略(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息	
接口类型	C	
原型	extern char* kdk_bluetooth_get_link_policy(int id);	
描述	获取蓝牙的连接策略	
参数	id	蓝牙的设备id
返回值	char*	成功返回蓝牙的链接策略，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取蓝牙的总线(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息	
接口类型	C	
原型	extern char* kdk_bluetooth_get_bus(int id);	
描述	获取蓝牙的总线	
参数	id	蓝牙的设备id
返回值	char*	成功返回蓝牙的总线，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取蓝牙的SCO MTU(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息	
接口类型	C	
原型	extern char* kdk_bluetooth_get_scomtu(int id);	
描述	获取蓝牙的SCO MTU	
参数	id	蓝牙的设备id
返回值	char*	成功返回蓝牙的SCO MTU，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取蓝牙的ALC MTU(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息
接口类型	C
原型	extern char* kdk_bluetooth_get_alcmtu(int id);
描述	获取蓝牙的ALC MTU

参数	id	蓝牙的设备id
返回值	char*	成功返回蓝牙的ALC MTU，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取蓝牙的数据包类型(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息	
接口类型	C	
原型	extern char* kdk_bluetooth_get_packettype(int id);	
描述	获取蓝牙的数据包类型	
参数	id	蓝牙的设备id
返回值	char*	成功返回蓝牙的数据包类型，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取蓝牙的功能(自2.2.3.5版本启用)

子模块	获取蓝牙设备信息	
接口类型	C	
原型	extern char* kdk_bluetooth_get_features(int id);	
描述	获取蓝牙的功能	
参数	id	蓝牙的设备id
返回值	char*	成功返回蓝牙的功能，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

3.1.2.7 获取显卡设备信息

封装 C 接口获取到显卡设备硬件信息。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkydisplay.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkyhw.so
```

- **子模块信息：**

获取显卡的制造商(自2.2.3.5版本启用)

子模块	获取显卡信息
接口类型	C
原型	extern char* kdk_display_get_vendor();

描述	获取显卡的制造商	
参数	无	无
返回值	char*	成功返回显卡的制造商，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显卡的型号(自2.2.3.5版本启用)

子模块	获取显卡信息	
接口类型	C	
原型	extern char* kdk_display_get_product();	
描述	获取显卡的型号	
参数	无	无
返回值	char*	成功返回显卡的型号，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显卡的说明(自2.2.3.5版本启用)

子模块	获取显卡信息	
接口类型	C	
原型	extern char* kdk_display_get_description();	
描述	获取显卡的说明	
参数	无	无
返回值	char*	成功返回显卡的说明，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显卡的物理id(自2.2.3.5版本启用)

子模块	获取显卡信息	
接口类型	C	
原型	extern char* kdk_display_get_physical_id();	
描述	获取显卡的物理id	
参数	无	无
返回值	char*	成功返回显卡的物理id，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显卡的总线地址(自2.2.3.5版本启用)

子模块	获取显卡信息

接口类型	C	
原型	extern char* kdk_display_get_bus_info();	
描述	获取显卡的总线地址	
参数	无	无
返回值	char*	成功返回显卡的总线地址，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显卡的设备版本(自2.2.3.5版本启用)

子模块	获取显卡信息	
接口类型	C	
原型	extern char* kdk_display_get_version();	
描述	获取显卡的设备版本	
参数	无	无
返回值	char*	成功返回显卡的设备版本，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显卡的数据宽度(自2.2.3.5版本启用)

子模块	获取显卡信息	
接口类型	C	
原型	extern char* kdk_display_get_width();	
描述	获取显卡的数据宽度	
参数	无	无
返回值	char*	成功返回显卡的数据宽度，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显卡的频率(自2.2.3.5版本启用)

子模块	获取显卡信息	
接口类型	C	
原型	extern char* kdk_display_get_clock();	
描述	获取显卡的频率	
参数	无	无
返回值	char*	成功返回显卡的频率，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显卡的功能(自2.2.3.5版本启用)

子模块	获取显卡信息	
接口类型	C	
原型	extern char* kdk_display_get_capabilities();	
描述	获取显卡的功能	
参数	无	无
返回值	char*	成功返回显卡的功能，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显卡的配置(自2.2.3.5版本启用)

子模块	获取显卡信息	
接口类型	C	
原型	extern char* kdk_display_get_configuration();	
描述	获取显卡的配置	
参数	无	无
返回值	char*	成功返回显卡的配置，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显卡的资源(自2.2.3.5版本启用)

子模块	获取显卡信息	
接口类型	C	
原型	extern char* kdk_display_get_resources();	
描述	获取显卡的资源	
参数	无	无
返回值	char*	成功返回显卡的资源，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

3.1.2.8 获取显示器设备信息

封装 C 接口获取到显示器设备硬件信息。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkyedid.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkyedid.so
```

- **子模块信息：**

获取显示器的当前接口(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern char** kdk_edid_get_interface();	
描述	获取显示器的当前接口	
参数	无	无
返回值	char**	成功返回显示器的当前接口，由NULL字符串表示结尾；由alloc生成，需要被kdk_edid_freeall回收
	NULL	获取失败
无	无	

获取显示器的伽马值(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern float kdk_edid_get_gamma(char *name);	
描述	获取显示器的伽马值	
参数	name	显示器的当前接口
返回值	float	成功返回显示器的伽马值
	0.00	获取失败
无	无	

获取显示器的屏幕尺寸（英寸）(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern float kdk_edid_get_size(char *name);	
描述	获取显示器的屏幕尺寸（英寸）	
参数	name	显示器的当前接口
返回值	float	成功返回显示器的屏幕尺寸（英寸）
	0.00	获取失败
无	无	

获取显示器的最大分辨率(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern char* kdk_edid_get_max_resolution(char *name);	
描述	获取显示器的最大分辨率	
参数	name	显示器的当前接口
返回值	char*	成功返回显示器的最大分辨率，返回的字符串需要被 free 释放。
	NULL	获取失败

无	无	

获取显示器的显示器型号(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern char* kdk_edid_get_model(char *name);	
描述	获取显示器的显示器型号	
参数	name	显示器的当前接口
返回值	char*	成功返回显示器的显示器型号，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显示器的可视面积(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern char* kdk_edid_get_visible_area(char *name);	
描述	获取显示器的可视面积	
参数	name	显示器的当前接口
返回值	char*	成功返回显示器的可视面积，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显示器的厂商(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern char* kdk_edid_get_manufacturer(char *name);	
描述	获取显示器的厂商	
参数	name	显示器的当前接口
返回值	char*	成功返回显示器的厂商，返回的字符串需要被 free 释放。
	NULL	获取失败
无	无	

获取显示器的生产日期/周(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern int kdk_edid_get_week(char *name);	
描述	获取显示器的生产日期/周	
参数	name	显示器的当前接口
返回值	int	成功返回显示器的生产日期/周。

	-1	获取失败
无	无	

获取显示器的生产日期/年(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern int kdk_edid_get_year(char *name);	
描述	获取显示器的生产日期/年	
参数	name	显示器的当前接口
返回值	int	成功返回显示器的生产日期/年。
	-1	获取失败
无	无	

获取是否是主显示器（是/否）(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern int kdk_edid_get_primary(char *name);	
描述	获取是否是主显示器（是/否）	
参数	name	显示器的当前接口
返回值	1	是主显示器
	0	不是主显示器
无	无	

获取分辨率(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern char* kdk_edid_get_resolution(char *name);	
描述	获取分辨率	
参数	name	显示器的当前接口
返回值	char*	成功返回分辨率， 返回的字符串需要被 free 释放
	NULL	获取失败
无	无	

获取图像高宽比(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern char* kdk_edid_get_ratio(char *name);	
描述	获取图像高宽比	
参数	name	显示器的当前接口
返回值	char*	成功返回图像高宽比， 返回的字符串需要被 free

		释放
	NULL	获取失败
无	无	

获取显示器edid未解析的字符串(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern char* kdk_edid_get_character(char *name);	
描述	获取显示器edid未解析的字符串	
参数	name	显示器的当前接口
返回值	char*	成功返回edid未解析的字符串，返回的字符串需要被 free 释放
	NULL	获取失败
无	无	

用于回收字符串列表(自2.2.3.5版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern inline void kdk_edid_freeall(char **ptr);	
描述	用于回收字符串列表	
参数	ptr	字符串列表
返回值	无	无
	无	无
备注	无	

获取显示器的最大亮度(自2.4.1.0版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern BrightnessInfo* kdk_edid_get_max_brightness(char *name);	
描述	获取显示器的最大亮度	
参数	name	显示器的当前接口
返回值	BrightnessInfo*	显示器的最大亮度结构体。返回的结构体需要由kdk_free_brightnessInfo()回收。
	NULL	获取失败
备注	成员：brightness_percentage(int)，描述：当前/最大亮度百分比 成员：brightness_value(int)，描述：当前/最大亮度值	

获取显示器的当前亮度(自2.4.1.0版本启用)

子模块	获取显示器信息
接口类型	C
原型	extern BrightnessInfo* kdk_edid_get_current_brightness(char *name);
描述	获取显示器的当前亮度

参数	name	显示器的当前接口
返回值	BrightnessInfo*	显示器的当前亮度结构体。 返回的结构体需要由kdk_free_brightnessInfo()回收。
	NULL	获取失败
备注	成员：brightness_percentage(int)，描述：当前/最大亮度百分比 成员：brightness_value(int)，描述：当前/最大亮度值	

获取显示器红色的色度坐标值(自2.4.1.0版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern ChromaticityCoordinates* kdk_edid_get_red_primary(char *name);	
描述	获取显示器红色的色度坐标值	
参数	name	显示器的当前接口
返回值	ChromaticityCoordinates*	显示器色度坐标值结构体。 返回的结构体需要由kdk_free_chromaticityCoordinates()回收。
	NULL	获取失败
备注	成员：xCoordinate(char *)，描述：色度坐标x轴 成员：yCoordinate(char *)，描述：色度坐标y轴	

获取显示器绿色的色度坐标值(自2.4.1.0版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern ChromaticityCoordinates* kdk_edid_get_green_primary(char *name);	
描述	获取显示器绿色的色度坐标值	
参数	name	显示器的当前接口
返回值	ChromaticityCoordinates*	显示器色度坐标值结构体。 返回的结构体需要由kdk_free_chromaticityCoordinates()回收。
	NULL	获取失败
备注	成员：xCoordinate(char *)，描述：色度坐标x轴 成员：yCoordinate(char *)，描述：色度坐标y轴	

获取显示器蓝色的色度坐标值(自2.4.1.0版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern ChromaticityCoordinates* kdk_edid_get_blue_primary(char *name);	
描述	获取显示器蓝色的色度坐标值	
参数	name	显示器的当前接口
返回值	ChromaticityCoordinates*	显示器色度坐标值结构体。 返回的结构体需要由kdk_free_chromaticityCoordinates()回收。

	NULL	获取失败
备注	成员：xCoordinate(char *)，描述：色度坐标x轴 成员：yCoordinate(char *)，描述：色度坐标y轴	

获取显示器白色的色度坐标值(自2.4.1.0版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern ChromaticityCoordinates* kdk_edid_get_white_primary(char *name);	
描述	获取显示器白色的色度坐标值	
参数	name	显示器的当前接口
返回值	ChromaticityCoordinates*	显示器色度坐标值结构体。 返回的结构体需要由kdk_free_chromaticityCoordinates()回收。
	NULL	获取失败
备注	成员：xCoordinate(char *)，描述：色度坐标x轴 成员：yCoordinate(char *)，描述：色度坐标y轴	

获取显示器物理水平DPI(自2.4.1.0版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern float kdk_edid_get_rawDpiX(char *name);	
描述	获取显示器物理水平DPI	
参数	name	显示器的当前接口
返回值	float	显示器物理水平DPI
	0.0	获取失败
备注	无	

获取显示器物理垂直DPI(自2.4.1.0版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern float kdk_edid_get_rawDpiY(char *name);	
描述	获取显示器物理垂直DPI	
参数	name	显示器的当前接口
返回值	float	显示器物理垂直DPI
	0.0	获取失败
备注	无	

获取显示器的刷新率(自2.4.1.0版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern char* kdk_edid_get_refreshRate(char *name);	
描述	获取显示器的刷新率	
参数	name	显示器的当前接口
返回值	char*	成功返回显示器刷新率，

		返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取显示器的方向(自2.4.1.0版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern char* kdk_edid_get_rotation(char *name);	
描述	获取显示器的方向	
参数	name	显示器的当前接口
返回值	char*	成功返回显示器的方向，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取显示器的序列号(自2.4.1.0版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern char* kdk_edid_get_serialNumber(char *name);	
描述	获取显示器的序列号	
参数	name	显示器的当前接口
返回值	char*	成功返回显示器的序列号，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

释放由kdk_edid_get_max_brightness和kdk_edid_get_current_brightness返回的显示器亮度(自2.4.1.0版本启用)

子模块	获取显示器信息	
接口类型	C	
原型	extern void kdk_free_brightnessInfo(BrightnessInfo *info);	
描述	释放由kdk_edid_get_max_brightness和kdk_edid_get_current_brightness返回的显示器亮度	
参数	info	由kdk_edid_get_max_brightness和kdk_edid_get_current_brightness返回的结构体指针
返回值	无	无
	无	无
备注	无	

释放由kdk_edid_get_red_primary、kdk_edid_get_green_primary、kdk_edid_get_blue_primary、kdk_edid_get_white_primary返回的色度坐标结构体(自2.4.1.0版本启用)

子模块	获取显示器信息
接口类型	C
原型	extern void kdk_free_chromaticityCoordinates(ChromaticityCoordinates *info);

描述	释放由kdk_edid_get_red_primary、kdk_edid_get_green_primary、kdk_edid_get_blue_primary、kdk_edid_get_white_primary返回的色度坐标结构体	
参数	info	由kdk_edid_get_red_primary、kdk_edid_get_green_primary、kdk_edid_get_blue_primary、kdk_edid_get_white_primary返回的结构体指针
返回值	无	无
	无	无
备注	无	

3.1.2.9 获取风扇设备信息

封装 C 接口获取到风扇设备硬件信息。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkyfan.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkyfan.so
```

- **子模块信息：**

获取风扇的信息(自2.2.3.5版本启用)

子模块	获取风扇设备信息	
接口类型	C	
原型	extern char** kdk_fan_get_information();	
描述	获取风扇的信息	
参数	无	无
返回值	char**	风扇的信息（名称，转速），需要被kdk_fan_freeall回收
	NULL	获取失败
备注	无	

用于回收字符串列表(自2.2.3.5版本启用)

子模块	获取风扇设备信息	
接口类型	C	
原型	extern inline void kdk_fan_freeall(char **ptr);	
描述	用于回收字符串列表	
参数	ptr	字符串列表
返回值	无	无
	无	无
备注	无	

3.1.2.10 获取键盘、鼠标、声卡、光驱、摄像头、电源信息

封装 C 接口获取到键盘、鼠标、声卡、光驱、摄像头、电源信息。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkyhw.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkyhwinfo.so
```

- **子模块信息：**

获取相关硬件信息(自2.2.3.5版本启用)

子模块	获取键盘、鼠标、声卡、光驱、摄像头、电源信息	
接口类型	C	
原型	extern struct HWInfo *kdk_hw_get_hwinfo(int type);	
描述	获取接入系统的键盘、鼠标、声卡、光驱、摄像头、 电源信息	
参数	type	3 keyboard; 5 mouse; 15 bound; 23 cdrom; 27 usb(仅摄像头)
返回值	HWInfo*	相关硬件信息， 具体参考返回结构体
	NULL	获取失败

备注	返回数据为命令hwinfo的部分返回数据，
	结构体中属性可能为空字符串
	struct HWInfo
	{
	char model[ATTRSIZE]; //模块名
	char vendor[ATTRSIZE];//厂商名
	char device[ATTRSIZE];//设备名
	char driver[ATTRSIZE];//设备驱动
	char revision[ATTRSIZE];//版本
	char busid[ATTRSIZE];//总线id
	char devicenum[ATTRSIZE];//设备号，
	例如鼠标键盘的字符设备号，块设备(光驱)的设备号
	char width[8];//声卡数据位宽
	char clock[8];//声卡时钟频率
	struct HWInfo *next;
	};	

释放由kdk_hw_get_hwinfo返回的硬件信息结构体(自2.2.3.5版本启用)

子模块	获取键盘、鼠标、声卡、光驱、摄像头、电源信息	
接口类型	C	
原型	extern void kdk_hw_free_hw(struct HWInfo *list);	
描述	释放由kdk_hw_get_hwinfo返回的硬件信息结构体	
参数	list	由kdk_hw_get_hwinfo返回的结构体指针
返回值	无	无
	无	无
备注	无	

获取电源信息(自2.2.3.5版本启用)

子模块	获取键盘、鼠标、声卡、光驱、摄像头、电源信息	
接口类型	C	
原型	extern struct Power *kdk_hw_get_powerinfo();	
描述	获取电源信息	
参数	无	无
返回值	Power*	电源信息， 具体参考返回结构体
	无	无

备注	获取电源信息返回值结构体说明：
	struct power_device
	{
	char name[256]; //设备名
	char native_path[32]; //电源设备的本机路径
	bool power_supply; //显示 "true" 表示这是一个电源设备，
	如电池或充电器
	char updated[64]; //上次更新信息的时间戳
	bool has_history; //显示 "true" 表示设备有历史信息，
	即充电和放电的历史记录
	bool has_statistics; //显示 "true" 表示设备有统计信息，
	例如充电次数、使用时间等。
	bool is_persent; //显示 "true" 表示电池存在
	bool is_rechargeable; //显示 "true" 表示电池可充电
	char state[32]; //电池的当前状态，例如
	"charging"、"discharging"、"fully-charged" 等
	char warning_level[32]; //
	电池电量低于该级别时会触发警告，
	double energy; //电池的能量信息，包括当前能量、
	满电能量等
	double energy_empty; //电池耗尽时的能量
	double energy_full; //电池充满时的能量
	double energy_full_design; //电池的设计容量
	double energy_rate; //电池的当前充电或放电速率
	double voltage; //电池的当前电压
	long time_to_empty; //电池完全放空（耗尽）所需的时间
	long time_to_full; //电池完全充满所需的时间
	double percentage; //电池的电量百分比
	（电池当前的电量相对于满电容量的百分比）
	double temperature; //电池的当前温度
	double capacity; //电池的电量百分比
	（电池当前充电状态相对于满电的百分比）
	char technology[32]; //电池所使用的技术
	bool online; //显示 "true" 表示设备当前在线（连接到电源）
	char icon_name[64]; //显示与设备状态相关联的图标名称
	char model[64]; //电池型号
	int battery_level; //电池电量等级
	int type; //充电类型
	struct power_device *next;
	};
	
	struct Power
	{
	char daemon_version[32]; //版本
	bool on_battery; //系统当前是否依赖电池供电。
	如果值为'true'，则表示系统正在使用电池供电；
	如果值为'false'，则表示系统连接到外部电源
	bool lid_is_closed; //设备的盖子（例如笔记本电脑的盖子）
	是否关闭。如果值为'true'，则表示盖子已关闭；
	如果值为'false'，则表示盖子处于打开状态
	bool lid_is_present; //设备是否具有可关闭的盖子。	

	如果值为'true'，则表示设备具有可关闭的盖子；
	如果值为'false'，则表示设备没有可关闭的盖子
	char critical_action[32]; //
	电池电量低于临界水平时系统采取的动作的方法
	struct power_device *devices;
	};

释放由kdk_hw_get_powerinfo返回的电源信息结构体(自2.2.3.5版本启用)

子模块	获取键盘、鼠标、声卡、光驱、摄像头、电源信息	
接口类型	C	
原型	extern void kdk_hw_free_power_info(struct Power *info);	
描述	释放由kdk_hw_get_powerinfo返回的电源信息结构体	
参数	info	由kdk_hw_get_powerinfo返回的结构体指针
返回值	无	无
	无	无
备注	无	

3.1.3 获取磁盘信息

- **安装命令：**

```
sudo apt-get install libkysdk-disk libkysdk-disk-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-disk
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKDISK kysdk-disk)
target_include_directories(demo PRIVATE ${KYSDKDISK_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKDISK_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKDISK_LIBRARIES})
```

3.1.3.1获取磁盘信息

封装 C 接口获取到磁盘硬件信息。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkydiskinfo.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkydiskinfo.so
```

- **子模块信息：**

获取系统中所有磁盘的列表(自1.2.0版本启用)

子模块	获取磁盘信息
接口类型	C

原型	extern char** kdk_get_disklist()	
描述	获取系统中所有磁盘的列表	
参数	无	无
返回值	char**	每个字符串表示一个磁盘的绝对路径， 结尾以NULL字符表示结束， 需要被kdk_get_disklist回收
	NULL	获取失败
备注	无	

释放由 kdk_get_disklist 返回的磁盘列表(自1.2.0版本启用)

子模块	获取磁盘信息	
接口类型	C	
原型	extern void kdk_free_disklist(char** disklist)	
描述	释放由kdk_get_disklist返回的磁盘列表	
参数	disklist	由kdk_get_disk_list返回的字符串指针
返回值	无	无
	无	无
备注	无	

获取系统中指定磁盘的磁盘信息(自1.2.0版本启用)

子模块	获取磁盘信息	
接口类型	C	
原型	extern kdk_diskinfo *kdk_get_diskinfo(const char *diskname)	
描述	获取指定磁盘的磁盘信息	
参数	diskname	指定磁盘名称， 应当是例如/dev/sda这种绝对路径， 或者是disklist中的某个元素
返回值	kdk_diskinfo*	该磁盘的详细信息； 由kdk_get_diskinfo释放
	NULL	获取失败
备注	成员：name(char *)，描述：磁盘绝对路径； 成员：sectors_num(unsigned long long)，描述： 扇区数量； 成员：sector_size(unsigned int)，描述： 每个扇区的字节数； 成员：total_size_MiB(float)，描述：磁盘容量，MiB为单位 成员：model(char *)，描述：型号； 成员：serial(char *)，描述：序列号； 成员：partition_nums(unsigned int)，描述：该磁盘/ 分区下的子分区数量 成员：disk_type(enum kdk_disk_type)，描述：磁盘类型， 固态 or 机械 or 混合； 成员：fwrev(char *)，描述：固件版本信息	

释放由 kdk_get_diskinfo 返回的磁盘信息结构体(自1.2.0版本启用)

子模块	获取磁盘信息
接口类型	C

原型	extern void kdk_free_diskinfo(kdk_diskinfo *disk);	
描述	释放由kdk_get_diskinfo返回的磁盘信息结构体	
参数	disk	由kdk_get_diskinfo返回的结构体指针
返回值	无	无
	无	无
备注	无	

获取系统接入所有硬盘(自2.2.3.5版本启用)

子模块	获取磁盘信息	
接口类型	C	
原型	extern char** kdk_get_hard_disk();	
描述	获取系统接入所有硬盘	
参数	无	无
返回值	char**	成功返回硬盘名称， 由kdk_free_disklist释放
	NULL	获取失败
备注	无	

获取硬盘大小(自2.2.3.5版本启用)

子模块	获取磁盘信息	
接口类型	C	
原型	extern char* kdk_get_hard_disk_size(const char *hardname);	
描述	获取硬盘大小	
参数	hardname	指定磁盘名称， 应当是例如/dev/sda这种绝对路径
返回值	char*	成功返回硬盘大小， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取硬盘固态版本(自2.2.3.5版本启用)

子模块	获取磁盘信息	
接口类型	C	
原型	extern char* kdk_get_hard_fwrev(const char *hardname);	
描述	获取硬盘固态版本	
参数	hardname	指定磁盘名称， 应当是例如/dev/sda这种绝对路径
返回值	char*	成功返回硬盘固态版本， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取硬盘类型(自2.2.3.5版本启用)

子模块	获取磁盘信息	
接口类型	C	
原型	extern char* kdk_get_hard_type(const char *hardname);	
描述	获取硬盘类型	
参数	hardname	指定磁盘名称， 应当是例如/dev/sda这种绝对路径
返回值	char*	成功返回硬盘类型， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取硬盘型号(自2.2.3.5版本启用)

子模块	获取磁盘信息	
接口类型	C	
原型	extern char* kdk_get_hard_model(const char *hardname);	
描述	获取硬盘型号	
参数	hardname	指定磁盘名称， 应当是例如/dev/sda这种绝对路径
返回值	char*	成功返回硬盘型号， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取硬盘序列号(自2.2.3.5版本启用)

子模块	获取磁盘信息	
接口类型	C	
原型	extern char* kdk_get_hard_serial(const char *hardname);	
描述	获取硬盘序列号	
参数	hardname	指定磁盘名称， 应当是例如/dev/sda这种绝对路径
返回值	char*	成功返回硬盘序列号， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取硬盘厂商(自2.2.3.5版本启用)

子模块	获取磁盘信息
接口类型	C
原型	extern char* kdk_get_hard_vendor(const char *hardname);

描述	获取硬盘厂商	
参数	hardname	指定磁盘名称， 应当是例如/dev/sda这种绝对路径
返回值	char*	成功返回硬盘厂商， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取硬盘运行时长(自2.4.1.0版本启用)

子模块	获取磁盘信息	
接口类型	C	
原型	extern int kdk_get_hard_running_time(const char *hardname);	
描述	获取硬盘运行时长	
参数	hardname	指定磁盘名称， 应当是例如/dev/sda这种绝对路径
返回值	int	硬盘从出厂到现在已运行时长总和
	-1	获取失败
备注	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

服务名称： com.kylin.kysdk.service

路径名称： /com/kylin/kysdk/disk

接口名称： com.kylin.kysdk.disk

- **python导入方法**

```
from kysdk import Disk
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.java.DiskMethod;
```

功能描述	接口类型	接口	入参	返回值
获取系统中所有磁盘的列表	dbus	QStringList getDiskList()	无	QStringList 返回值为磁盘列表
	python	getDiskList()->str	无	返回值为磁盘列表
	websocket	disk.getDiskList()	无	返回值为磁盘列表
	http	http://127.0.0.1:8888/disk/getDiskList	无	Json 返回值为磁盘列表
	java	List getDiskList()	无	List 返回值为磁盘列表

功能描述	接口类型	接口	入参	返回值
获取指定硬盘每个扇区的字节数	dbus	unsigned int getDiskSectorSize(const QString diskname)	diskname 指定磁盘路径	unsigned int 返回值为每个扇区的字节数
	python	getDiskSectorSize(dev_path:str)->int	dev_path 指定磁盘路径	返回值为每个扇区的字节数

	websocket	disk.getDiskSectorSize(pid)	pid 指定磁盘路径	返回值为每个扇区的字节数
	http	http://127.0.0.1:8888/disk/getDiskSectorSize?diskname=parameter	parameter 指定磁盘路径	Json 返回值为每个扇区的字节数
	java	UInt32 getDiskSectorSize(String diskname)	diskname 指定磁盘路径	UInt32 返回值为每个扇区的字节数

功能描述	接口类型	接口	入参	返回值
获取指定硬盘的磁盘容量	dbus	QString getDiskTotalSizeMiB(const QString diskname)	diskname 指定磁盘路径	QString 返回值为磁盘容量
	python	getDiskTotalSizeMiB(dev_path:str)->str	dev_path 指定磁盘路径	返回值为磁盘容量
	websocket	disk.getDiskTotalSizeMiB(pid)	pid 指定磁盘路径	返回值为磁盘容量
	http	http://127.0.0.1:8888/disk/getDiskTotalSizeMiB?diskname=parameter	parameter 指定磁盘路径	Json 返回值为磁盘容量
	java	String getDiskTotalSizeMiB(String diskname)	diskname 指定磁盘路径	String 返回值为磁盘容量

功能描述	接口类型	接口	入参	返回值
获取指定硬盘型号	dbus	QString getDiskModel(const QString diskname)	diskname 指定磁盘路径	QString 返回值为磁盘型号
	python	getDiskModel(dev_path:str)->str	dev_path 指定磁盘路径	返回值为磁盘型号
	websocket	disk.getDiskModel(pid)	pid 指定磁盘路径	返回值为磁盘型号
	http	http://127.0.0.1:8888/disk/getDiskModel?diskname=parameter	parameter 指定磁盘路径	Json 返回值为磁盘型号
	java	String getDiskModel(String diskname)	diskname 指定磁盘路径	String 返回值为磁盘型号

功能描述	接口类型	接口	入参	返回值
获取指定硬盘序列号	dbus	QString getDiskSerial(const QString diskname)	diskname 指定磁盘路径	QString 返回值为磁盘序列号
	python	getDiskSerial(dev_path:str)->str	dev_path 指定磁盘路径	返回值为磁盘序列号
	websocket	disk.getDiskSerial(pid)	pid 指定磁盘路径	返回值为磁盘序列号
	http	http://127.0.0.1:8888/disk/getDiskSerial?diskname=parameter	parameter 指定磁盘路径	Json 返回值为磁盘序列号
	java	String getDiskSerial(String diskname)	diskname 指定磁盘路径	String 返回值为磁盘序列号

功能描述	接口类型	接口	入参	返回值
获取指定硬盘的子分区数量	dbus	unsigned int getDiskPartitionNums(const QString diskname)	diskname 指定磁盘路径	unsigned int 返回值为磁盘/分区下的子分区数量
	python	getDiskPartitionNums(dev_path:str)->int	dev_path 指定磁盘路径	返回值为磁盘/分区下的子分区数量
	websocket	disk.getDiskPartitionNums(pid)	pid 指定磁盘路径	返回值为磁盘/分区下的子分区数量

	http	http://127.0.0.1:8888/disk/getDiskPartitionNums?diskname=parameter	parameter 指定磁盘路径	Json 返回值为磁盘/分区下的子分区数量
	java	UInt32 getDiskPartitionNums(String diskname)	diskname 指定磁盘路径	UInt32 返回值为磁盘/分区下的子分区数量

功能描述	接口类型	接口	入参	返回值
获取指定硬盘的类型	dbus	QString getDiskType(const QString diskname)	diskname 指定磁盘路径	QString 返回值为磁盘类型，机械(DISK_TYPE_HDD)/固态(DISK_TYPE_SSD)/其他(DISK_TYPE_OTHER)
	python	getDiskType(dev_path:str)->str	dev_path 指定磁盘路径	返回值为磁盘类型，机械(DISK_TYPE_HDD)/固态(DISK_TYPE_SSD)/其他(DISK_TYPE_OTHER)
	websocket	disk.getDiskType(pid)	pid 指定磁盘路径	返回值为磁盘类型，机械(DISK_TYPE_HDD)/固态(DISK_TYPE_SSD)/其他(DISK_TYPE_OTHER)
	http	http://127.0.0.1:8888/disk/getDiskType?diskname=parameter	parameter 指定磁盘路径	Json 返回值为磁盘类型，机械(DISK_TYPE_HDD)/固态(DISK_TYPE_SSD)/其他(DISK_TYPE_OTHER)
	java	String getDiskType(String diskname)	diskname 指定磁盘路径	String 返回值为磁盘类型，机械(DISK_TYPE_HDD)/固态(DISK_TYPE_SSD)/其他(DISK_TYPE_OTHER)

功能描述	接口类型	接口	入参	返回值
获取指定硬盘的固件版本	dbus	QString getDiskVersion(const QString diskname)	diskname 指定磁盘路径	QString 返回值为磁盘固件版本信息
	python	getDiskVersion(dev_path:str)->str	dev_path 指定磁盘路径	返回值为磁盘固件版本信息
	websocket	disk.getDiskVersion(pid)	pid 指定磁盘路径	返回值为磁盘固件版本信息
	http	http://127.0.0.1:8888/disk/getDiskVersion?diskname=parameter	parameter 指定磁盘路径	Json 返回值为磁盘固件版本信息
	java	String getDiskVersion(String diskname)	diskname 指定磁盘路径	String 返回值为磁盘固件版本信息

功能描述	接口类型	接口	入参	返回值
获取指定硬盘的扇区数量	dbus	unsigned long long getDiskSectorNum(const QString diskname)	diskname 指定磁盘路径	unsigned long long 返回值为磁盘扇区数量
	python	getDiskSectorNum(dev_path:str)->int	dev_path 指定磁盘路径	返回值为磁盘扇区数量
	websocket	disk.getDiskSectorNum(pid)	pid 指定磁盘路径	返回值为磁盘扇区数量
	http	http://127.0.0.1:8888/disk/getDiskSectorNum?diskname=parameter	parameter 指定磁盘路径	Json 返回值为磁盘固件版本信息
	java	UInt64 getDiskSectorNum(String diskname)	diskname	UInt64

			指定磁盘路径	返回值为磁盘扇区数量

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include "libkydiskinfo.h"
#include <stdio.h>

int main()
{
    char **disklist = kdk_get_disklist();
    for (int i = 0; disklist[i]; i ++)
    {
        printf("%s\n", disklist[i]);
    }

    kdk_free_disklist(disklist);

    return 0;
}
```

```
#----------------------------python语言示例----------------------------
from kysdk import Disk
disk = Disk()
dev_path = "/dev/sda"

# getDiskList接口
disk.getDiskList()
# getDiskSectorSize接口
disk.getDiskSectorSize(dev_path)
#getDiskTotalSizeMiB接口
disk.getDiskTotalSizeMiB(dev_path)
#getDiskModel接口
disk.getDiskModel(dev_path)
#getDiskSerial接口
disk.getDiskSerial(dev_path)
#getDiskPartitionNums接口
disk.getDiskPartitionNums(dev_path)
#getDiskType接口
disk.getDiskType(dev_path)
#getDiskVersion接口
disk.getDiskVersion(dev_path)
#getDiskSectorNum接口
disk.getDiskSectorNum(dev_path)
```

```
//----------------------------Java语言示例----------------------------
import kylin.kysdk.java.DiskMethod;
DiskMethod obj = new DiskMethod();
List<String> list = obj.getDiskList();
System.out.println("SecSize: " + obj.getDiskSectorSize(list.get(0)));
System.out.println("TotaSize: " + obj.getDiskTotalSizeMiB(list.get(0)));
System.out.println("Model: " + obj.getDiskModel(list.get(0)));
System.out.println("Serial: " + obj.getDiskSerial(list.get(0)));
System.out.println("PartNum: " + obj.getDiskPartitionNums(list.get(0)));
System.out.println("Type: " + obj.getDiskType(list.get(0)));
System.out.println("Version: " + obj.getDiskVersion(list.get(0)));
System.out.println("SecNum: " + obj.getDiskSectorNum(list.get(0)));
```

```
//----------------------------websocket语言示例----------------------------
    //pid 为指定磁盘路径
    var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        new QWebChannel(websocket,function(channel){
                var disk = channel.objects.disk;
                //返回信息接收
                disk.sendText.connect(function(message) {
                    ...
                });
                //获取磁盘列表
                disk.getDiskList();
                //获取扇区字节数
                disk.getDiskSectorSize(pid);
                //获取磁盘容量
                disk.getDiskTotalSizeMiB(pid);
                //获取磁盘型号
                disk.getDiskModel(pid);
                //获取磁盘序列号
                disk.getDiskSerial(pid);
                //获取子分区数量
                disk.getDiskPartitionNums(pid);
                //获取磁盘类型
                disk.getDiskType(pid);
                //获取固件版本信息
                disk.getDiskVersion(pid);
                //获取磁盘扇区数量
                disk.getDiskSectorNum(pid);
            }
        );
    }
```

```
//-----------------------------http语言示例-----------------------------
// parameter 为指定磁盘路径
1.http://127.0.0.1:8888/disk/getDiskList
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getDiskList()信息
    }
2.http://127.0.0.1:8888/disk/getDiskSectorSize?diskname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getDiskSectorSize(parameter)信息
    }
3.http://127.0.0.1:8888/disk/getDiskTotalSizeMiB?diskname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getDiskTotalSizeMiB(parameter)信息
    }
4.http://127.0.0.1:8888/disk/getDiskModel?diskname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getDiskModel(parameter)信息
    }
5.http://127.0.0.1:8888/disk/getDiskSerial?diskname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getDiskSerial(parameter)信息
    }

6.http://127.0.0.1:8888/disk/getDiskPartitionNums?diskname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getDiskPartitionNums(parameter)信息
    }
7.http://127.0.0.1:8888/disk/getDiskType?diskname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getDiskType(parameter)信息
    }
8.http://127.0.0.1:8888/disk/getDiskVersion?diskname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getDiskVersion(parameter)信息
    }
9.http://127.0.0.1:8888/disk/getDiskSectorNum?diskname=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getDiskSectorNum(parameter)信息
    }
```

3.1.4 获取包列表信息

- **安装命令：**

```
sudo apt-get install libkysdk-package libkysdk-package-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-package
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKPACKAGE kysdk-package)
target_include_directories(demo PRIVATE ${KYSDKPACKAGE_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKPACKAGE_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKPACKAGE_LIBRARIES})
```

3.1.4.1获取包列表信息

封装 C 接口获取系统中所有的包信息

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkypackages.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkypackage.so
```

- **子模块信息：**

获取系统中所有包列表(自1.2.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern kdk_package_list* kdk_package_get_packagelist()	
描述	获取系统中所有包列表(所有状态)	
参数	无	无
返回值	kdk_package_list*	包描述结构体列表； 返回的结构体需要由kdk_package_free_packagelist()回收。
	NULL	获取失败
备注	成员：name(char *)，描述：包名 成员：version(char *)，描述：版本号 成员：section(char *)，描述：包类型 成员：status(char *)，描述：状态 成员：size_kb(char *)，描述：包大小	

获取系统中指定包的版本号(自1.2.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern char* kdk_package_get_version(const char *name)	
描述	获取系统中指定包的版本号	
参数	name	软件包名
返回值	char*	成功返回版本号， 由alloc生成，需要被free
	NULL	获取失败或包不存在
备注	无	

检测指定包名的软件包是否正确安装(自1.2.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern int kdk_package_is_installed(const char *name, const char *version);	
描述	检测指定包名的软件包是否正确安装	
参数	name	软件包名
	version	版本号， 大部分情况下为NULL； 输入版本号可检测此版本的指定包是否安装
返回值	0	成功安装
	errno	失败返回错误码
备注	无	

回收由kdk_package_get_packagelist()返回的结构体(自1.2.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern void kdk_package_free_packagelist(kdk_package_list *list);	
描述	回收由kdk_package_get_packagelist()返回的结构体	
参数	list	由kdk_package_get_packagelist()返回的结构体
返回值	无	无
	无	无
备注	无	

获取指定应用的描述(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern char *kdk_package_get_description(const char *name);	
描述	获取指定应用的描述	
参数	name	应用名
返回值	char*	应用的描述信息
	NULL	获取失败
备注	无	

获取指定应用的安装目录路径(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern char **kdk_package_get_code_path(const char *name);	
描述	获取指定应用的安装目录路径	
参数	name	应用名
返回值	char**	应用的安装目录路径列表

	NULL	获取失败
备注	无	

获取指定应用的安装文件总数(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern int kdk_package_get_file_count(const char *name);	
描述	应用的安装文件总数	
参数	name	应用名
返回值	int	应用的安装文件总数
	0	获取失败
备注	无	

判断指定应用是否可以被移除(自2.4.1.0版本启用)(未启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern int kdk_package_is_removable(const char *name);	
描述	判断指定应用是否可以被移除	
参数	name	应用名
返回值	1	true
	0	false
备注	无	

判断安装deb包是否有足够的磁盘空间(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern int kdk_package_verify_disk_space(const char *path);	
描述	判断安装deb包是否有足够的磁盘空间	
参数	path	deb包路径
返回值	1	true
	0	false
备注	无	

获取默认浏览器名称(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern char* kdk_package_get_default_browser(void);	
描述	获取默认浏览器名称	
参数	无	无
返回值	char*	成功返回默认浏览器名称，返回的字符串需要被 free 释放
	NULL	获取失败

备注	无

获取默认图片查看器名称(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern char* kdk_package_get_default_image_viewer(void);	
描述	获取默认图片查看器名称	
参数	无	无
返回值	char*	成功返回默认图片查看器名称，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取默认音频播放器名称(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern char* kdk_package_get_default_audio_player(void);	
描述	获取默认音频播放器名称	
参数	无	无
返回值	char*	成功返回默认音频播放器名称，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取默认视频播放器名称(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern char* kdk_package_get_default_video_player(void);	
描述	获取默认视频播放器名称	
参数	无	无
返回值	char*	成功返回默认视频播放器名称，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取默认PDF文档查看器名称(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern char* kdk_package_get_default_pdf_viewer(void);	
描述	获取默认PDF文档查看器名称	
参数	无	无
返回值	char*	成功返回默认PDF文档查看器名称，返回的字符串需要被 free 释放
	NULL	获取失败

备注	无

获取默认WORD文档查看器名称(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern char* kdk_package_get_default_word_viewer(void);	
描述	获取默认WORD文档查看器名称	
参数	无	无
返回值	char*	成功返回默认WORD文档查看器名称，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取默认EXCEL文档查看器名称(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern char* kdk_package_get_default_excel_viewer(void);	
描述	获取默认EXCEL文档查看器名称	
参数	无	无
返回值	char*	成功返回默认EXCEL文档查看器名称，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取默认PPT文档查看器名称(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern char* kdk_package_get_default_ppt_viewer(void);	
描述	获取默认PPT文档查看器名称	
参数	无	无
返回值	char*	成功返回默认PPT文档查看器名称，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取开始菜单中的所有应用信息(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern kdk_startmenu_list* kdk_package_get_startmenu_list(void);	
描述	获取开始菜单中的所有应用信息(内容读取自desktop文件)	
参数	无	无
返回值	kdk_startmenu_list*	成功返回开始菜单中的所有应用信息，由kdk_package_free_startmenu_list释放
	NULL	获取失败

备注	typedef struct _kdk_startmenu_t
	{
	char *name; // 程序名称
	char *version; // 版本号
	char *company; // 所属公司
	char *cmd; // 可执行文件路径
	char *param; // 启动参数
	char *icon; // 图标信息
	} kdk_startmenu_t;
	
	typedef struct _kdk_startmenu_list
	{
	unsigned int nums; //list的成员个数
	kdk_startmenu_t **list; //应用信息列表
	} kdk_startmenu_list;

释放kdk_package_get_startmenu_list的返回(自2.4.1.0版本启用)

子模块	获取包列表信息	
接口类型	C	
原型	extern void kdk_package_free_startmenu_list(kdk_startmenu_list *list);	
描述	释放kdk_package_get_startmenu_list的返回	
参数	list	kdk_package_get_startmenu_list返回的指针
返回值	无	无
	无	无
备注	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

服务名称： com.kylin.kysdk.service

路径名称： /com/kylin/kysdk/packageinfo

接口名称： com.kylin.kysdk.packageinfo

- **python导入方法**

from kysdk import Packageinfo

- **websocket调用**
- **Java导入方法**

import kylin.kysdk.java.PackageInfoMethod;

功能描述	接口类型	接口	入参	返回值
获取系统中所有包列表(所有状态)	dbus	QStringList getPackageList()	无	QStringList 返回值为系统中所有包列表
	python	getPackageList()->list	无	返回值为系统中所有包列表
	websocket	packageinfo.getPackageList()	无	返回值为系统中所有包列表
	http	http://127.0.0.1:8888/packageinfo/getPackageList	无	Json 返回值为系统中所有包列表
	java	List getPackageList()	无	List 返回值为系统中所有包列表

功能描述	接口类型	接口	入参	返回值
获取系统中指定包的版本号	dbus	QString getPackageVersion(const QString	packageName	QString

		packageName)	软件包名	返回值为系统中指定
	python	getPackageVersion(package_name)->str	package_name 软件包名	返回值为系统中指定
	websocket	packageinfo.getPackageVersion(pkgname)	pkgname 软件包名	返回值为系统中指定
	http	http://127.0.0.1:8888/packageinfo/getPackageVersion?packagename=parameter	parameter为软件包名	Json 返回值为系统中指定
	java	String getPackageVersion(String packageName)	packageName 软件包名	String 返回值为系统中指定

功能描述	接口类型	接口	入参	返
检测指定包名的软件包是否正确安装	dbus	int getPackageInstalled(const QString packageName, const QString version)	packageName 软件包名 version 版本号，大部分情况下为NULL	返回值为软件 成功(1)/
	python	getPackageInstalled(package_name, version=None)->int	package_name 软件包名 version 版本号，默认为None	返回值为软件 成功(1)/
	websocket	packageinfo.getPackageInstalled(pkgname, vsname)	pkgname 软件包名 vsname 版本号，大部分情况下为NULL	返回值为软件 成功(1)/
	http	http://127.0.0.1:8888/packageinfo/getPackageInstalled?packagename=parameter&packageversion=version	parameter 为软件包名 version为版本号，大部分情况下为NULL	返回值为软件 成功(1)/
	java	int getPackageInstalled(String packageName, String version)	packageName 软件包名 version 版本号，大部分情况下为NULL	返回值为软件 成功(1)/

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include "libkypackages.h"
#include <stdio.h>

static void print_package(kdk_package_t *package)
{
    if (NULL == package)
        return;
    printf("包名: %s\t", package->name);
    printf("版本号: %s\t", package->version);
    printf("包类型: %s\t", package->section);
    printf("包状态: %s\t", package->status);
    printf("包大小: %lu\n", package->size_kb);
}

int main()
{
    kdk_package_list *list = kdk_package_get_packagelist();
    printf("系统中一共%u个软件包\n", list->nums);
    for (size_t i = 0; i < list->nums; i ++)
    {
        print_package(list->list[i]);
    }
    kdk_package_free_packagelist(list);

    printf("Evolution是否安装: %s\n", kdk_package_is_installed("evolution", NULL) == 1 ? "是" : "否");
    char *version = kdk_package_get_version("evolution");
    printf("Evolution版本号: %s\n", version);
    free(version);

    char *description = kdk_package_get_description("aapt");
    if(description != NULL)
    {
        printf("%s\n", description);
        free(description);
    }

    char **dir_list = kdk_package_get_code_path("aapt");
    if(NULL != dir_list)
    {
        int i = 0;
        while (dir_list[i])
        {
            printf("%s\n", dir_list[i++]);
        }
        for(int i = 0; dir_list[i]; i++)
        {
            free(dir_list[i]);
        }
        free(dir_list);
    }


    printf("%d\n", kdk_package_get_file_count("aapt"));

    printf("%d\n", kdk_package_verify_disk_space("/home/kylin/systemd_245.4-4kylin3.20k0.17oemhwy0.4.u_arm64.deb"));

    printf("%s\n", kdk_package_get_default_browser() ? kdk_package_get_default_browser() : "get browser failed");
    printf("%s\n", kdk_package_get_default_image_viewer() ? kdk_package_get_default_image_viewer() : "get image failed");
    printf("%s\n", kdk_package_get_default_audio_player() ? kdk_package_get_default_audio_player() : "get audio failed");
    printf("%s\n", kdk_package_get_default_video_player() ? kdk_package_get_default_video_player() : "get video failed");
    printf("%s\n", kdk_package_get_default_pdf_viewer() ? kdk_package_get_default_pdf_viewer() : "get pdf failed");
    printf("%s\n", kdk_package_get_default_word_viewer() ? kdk_package_get_default_word_viewer() : "get word failed");
    printf("%s\n", kdk_package_get_default_excel_viewer() ? kdk_package_get_default_excel_viewer() : "get execl failed");
    printf("%s\n", kdk_package_get_default_ppt_viewer() ? kdk_package_get_default_ppt_viewer() : "get ppt failed");

    kdk_startmenu_list *start_menu_list = kdk_package_get_startmenu_list();
    if (NULL != start_menu_list)
    {
        for (int i = 0; i < start_menu_list->nums; i++)
        {
```

```
            kdk_startmenu_t *tmp = start_menu_list->list[i];
            printf("%s\n", tmp->cmd);
            printf("%s\n", tmp->company);
            printf("%s\n", tmp->icon);
            printf("%s\n", tmp->name);
            printf("%s\n", tmp->param);
            printf("%s\n", tmp->version);
            printf("\n");
        }
        kdk_package_free_startmenu_list(start_menu_list);
    }

    return 0;
}
```

```
#----------------------------python语言示例----------------------------
from kysdk import Packageinfo
package = Packageinfo()
# getPackageList接口
package.getPackageList()
# getPackageVersion接口
package.getPackageVersion(package_name)
# getPackageInstalled接口
package.getPackageInstalled(package_name, version)
```

```
//----------------------------Java语言示例----------------------------
import kylin.kysdk.java.PackageInfoMethod;
PackageInfoMethod obj = new PackageInfoMethod();
List<String> list = obj.getPackageList();
System.out.println(list);
System.out.println("Version:" + obj.getPackageVersion(list.get(1)));
System.out.println("state: " + obj.getPackageInstalled(list.get(1), obj.getPackageVersion(list.get(1))));
```

```
//----------------------------websocket语言示例----------------------------
    // pkgname 为软件包名,vsname为软件包对应版本号
    var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        pkgname = $("#pkgname").val()
        vsname = $("#vsname").val()
        new QWebChannel(websocket,function(channel){
                var packageinfo = channel.objects.packageinfo;
                //返回信息接收
                packageinfo.sendText.connect(function(message) {
                    ...
                });
                //获得系统中所有包列表
                packageinfo.getPackageList();
                //获得系统中指定包的版本号
                packageinfo.getPackageVersion(pkgname);
                //软件包是否正确安装
                packageinfo.getPackageInstalled(pkgname, vsname);
            }
        );
    }
```

```
//-----------------------------http语言示例-----------------------------
// parameter 为软件包名,vertsion为软件包对应版本号
1.http://127.0.0.1:8888/packageinfo/getPackageList
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPackageList()信息
    }
2.http://127.0.0.1:8888/packageinfo/getPackageVersion?packagename=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPackageVersion(parameter)信息
    }
3.http://127.0.0.1:8888/packageinfo/getPackageInstalled?packagename=parameter&packageversion=version
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPackageInstalled(parameter,version)信息
    }
```

3.1.5 获取系统资源信息

- **安装命令：**

```
sudo apt-get install libkysdk-proc libkysdk-proc-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-proc
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKPROC kysdk-proc)
target_include_directories(demo PRIVATE ${KYSDKPROC_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKPROC_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKPROC_LIBRARIES})
```

3.1.5.1获取资源信息

封装 C 接口获取到内存、swap 分区、cpu 的使用率等资源信息。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkyrtinfo.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkyrtinfo.so
```

- **子模块信息：**

> 获取系统中物理内存总大小(自1.2.0版本启用)

子模块	获取资源信息
接口类型	C
原型	extern unsigned long kdk_rti_get_mem_res_total_KiB()

描述	获取系统中物理内存总大小	
参数	无	无
返回值	unsigned long	成功返回物理内存大小，KiB为单位
	0	获取失败
备注	无	

获取物理内存使用率(自1.2.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern float kdk_rti_get_mem_res_usage_percent()	
描述	获取物理内存使用率	
参数	无	无
返回值	float	成功返回物理内存使用率
	0.00	获取失败
备注	无	

获取物理内存使用大小(自1.2.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_res_usage_KiB()	
描述	获取物理内存使用大小， 注意Buffer/Cache被计算为已使用内存	
参数	无	无
返回值	unsigned long	成功返回物理内存使用大小，KiB为单位
	0	获取失败
备注	无	

获取实际可用的物理内存大小(自1.2.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_res_available_KiB()	
描述	获取实际可用的物理内存大小，该数值约等于Free + Buffer + Cache	
参数	无	无
返回值	unsigned long	成功返回可用物理内存大小，KiB为单位
	0	获取失败
备注	无	

获取实际空闲的物理内存大小(自1.2.0版本启用)

子模块	获取资源信息
接口类型	C

原型	extern unsigned long kdk_rti_get_mem_res_free_KiB()	
描述	获取实际空闲的物理内存大小， 注意Buffer/Cache被计算为已使用内存	
参数	无	无
返回值	unsigned long	成功返回空闲的物理内存大小，KiB为单位
	0	获取失败
备注	无	

获取所有应用申请的虚拟内存总量(自1.2.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_virt_alloc_KiB()	
描述	获取所有应用申请的虚拟内存总量	
参数	无	无
返回值	unsigned long	成功返回虚拟内存总申请量，KiB为单位
	0	获取失败
备注	无	

获取系统中 Swap 分区总大小(自1.2.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_swap_total_KiB()	
描述	获取系统中Swap分区总大小	
参数	无	无
返回值	unsigned long	成功返回Swap分区大小，KiB为单位
	0	获取失败
备注	无	

获取 Swap 分区使用率(自1.2.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern float kdk_rti_get_mem_swap_usage_percent()	
描述	获取Swap分区使用率	
参数	无	无
返回值	float	成功返回Swap分区使用率
	0.00	获取失败
备注	无	

获取 Swap 分区使用量(自1.2.0版本启用)

子模块	获取资源信息
接口类型	C

原型	extern unsigned long kdk_rti_get_mem_swap_usage_KiB()	
描述	获取Swap分区使用量	
参数	无	无
返回值	unsigned long	成功返回Swap分区使用量，KiB为单位
	0	获取失败
备注	无	

获取 Swap 分区空闲大小(自1.2.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_swap_free_KiB()	
描述	获取Swap分区空闲大小	
参数	无	无
返回值	unsigned long	成功返回Swap分区空闲大小，KiB为单位
	0	获取失败
备注	无	

获取 CPU 瞬时使用率(自1.2.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern float kdk_rti_get_cpu_current_usage()	
描述	获取CPU瞬时使用率	
参数	无	无
返回值	float	成功返回CPU瞬时使用率，该值 < 1.00
	0.00	获取失败
备注	无	

获取操作系统开机时长(自1.2.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern int kdk_rti_get_uptime(unsigned int *day, unsigned int *hour, unsigned int *min, unsigned int *sec)	
描述	获取操作系统开机时长	
参数	day	开机天数
	hour	小时数，该数值一定 < 24
	min	分钟数，该数值一定 < 60
	sec	秒数，该数值一定 < 60
返回值	0	成功
	非0	失败
备注	无	

获取共享内存大小(自2.4.1.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_shared_KiB();	
描述	获取共享内存大小	
参数	无	无
返回值	unsigned long	共享内存大小，KiB为单位
	0	获取失败
备注	无	

获取高速缓存大小(自2.4.1.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_cached_KiB();	
描述	获取高速缓存大小	
参数	无	无
返回值	unsigned long	高速缓存大小，KiB为单位
	0	获取失败
备注	无	

获取数据缓存大小(自2.4.1.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_buffers_KiB();	
描述	获取数据缓存大小	
参数	无	无
返回值	unsigned long	数据缓存大小，KiB为单位
	0	获取失败
备注	无	

获取交换缓存区大小(自2.4.1.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_swap_cached_KiB();	
描述	获取交换缓存区大小	
参数	无	无
返回值	unsigned long	交换缓存区大小，KiB为单位
	0	获取失败
备注	无	

获取活跃的缓冲文件大小(自2.4.1.0版本启用)

子模块	获取资源信息
接口类型	C

原型	extern unsigned long kdk_rti_get_mem_active_KiB();	
描述	获取活跃的缓冲文件大小	
参数	无	无
返回值	unsigned long	活跃的缓冲文件大小，KiB为单位
	0	获取失败
备注	无	

获取不活跃的缓冲文件大小(自2.4.1.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_inactive_KiB();	
描述	获取不活跃的缓冲文件大小	
参数	无	无
返回值	unsigned long	不活跃的缓冲文件大小，KiB为单位
	0	获取失败
备注	无	

获取脏页大小(自2.4.1.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_dirty_KiB();	
描述	获取脏页大小	
参数	无	无
返回值	unsigned long	脏页大小，KiB为单位
	0	获取失败
备注	无	

获取映射大小(自2.4.1.0版本启用)

子模块	获取资源信息	
接口类型	C	
原型	extern unsigned long kdk_rti_get_mem_map_KiB();	
描述	获取映射大小	
参数	无	无
返回值	unsigned long	映射大小，KiB为单位
	0	获取失败
备注	无	

获取内核数据结构缓存大小(自2.4.1.0版本启用)

子模块	获取资源信息
接口类型	C
原型	extern unsigned long kdk_rti_get_mem_slab_KiB();
描述	获取内核数据结构缓存大小

参数	无	无
返回值	unsigned long	内核数据结构缓存大小，KiB为单位
	0	获取失败
备注	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

> **服务名称：** com.kylin.kysdk.service
>
> **路径名称：** /com/kylin/kysdk/resource
>
> **接口名称：** com.kylin.kysdk.resource

- **python导入方法**

```
from kysdk import Resource
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.java.ResourceMethod;
```

功能描述	接口类型	接口	入参	返回值
获取系统中物理内存总大小	dbus	double getMemTotalKiB()	无	double 返回值为系统中物理内存总大小
	python	getMemTotalKiB()->int	无	返回值为系统中物理内存总大小
	websocket	resource.getMemTotalKiB()	无	返回值为系统中物理内存总大小
	http	http://127.0.0.1:8888/resource/getMemTotalKiB	无	Json 返回值为系统中物理内存总大小
	java	double getMemTotalKiB()	无	double 返回值为系统中物理内存总大小

功能描述	接口类型	接口	入参	返回值
获取物理内存使用率	dbus	double getMemUsagePercent()	无	double 返回值为物理内存使用率
	python	getMemUsagePercent()->int	无	返回值为物理内存使用率
	websocket	resource.getMemUsagePercent()	无	返回值为物理内存使用率
	http	http://127.0.0.1:8888/resource/getMemUsagePercent	无	Json 返回值为物理内存使用率
	java	double getMemUsagePercent()	无	double 返回值为物理内存使用率

功能描述	接口类型	接口	入参	返回值
获取物理内存使用大小，注意Buffer/Cache被计算为已使用内存	dbus	double getMemUsageKiB()	无	double 返回值为物理内存使用大小
	python	getMemUsageKiB()->int	无	返回值为物理内存使用大小
	websocket	resource.getMemUsageKiB()	无	返回值为物理内存使用大小
	http	http://127.0.0.1:8888/resource/getMemUsageKiB	无	Json 返回值为物理内存使用大小
	java	double getMemUsageKiB()	无	double 返回值为物理内存使用大小

功能描述	接口类型	接口	入参	返回值
获取实际可用的物理内存大小，该数值约等于Free + Buffer + Cache	dbus	double getMemAvailableKiB()	无	double 返回值为实际可用的物理内存大小
	python	getMemAvailableKiB()->int	无	返回值为实际可用的物理内存大小
	websocket	resource.getMemAvailableKiB()	无	返回值为实际可用的物理内存大小
	http	http://127.0.0.1:8888/resource/getMemAvailableKiB	无	Json 返回值为实际可用的物理内存大小
	java	double getMemAvailableKiB()	无	double 返回值为实际可用的物理内存大小

功能描述	接口类型	接口	入参	返回值
获取实际空闲的物理内存大小，注意Buffer/Cache被计算为已使用内存	dbus	double getMemFreeKiB()	无	double 返回值为实际空闲的物理内存大小
	python	getMemFreeKiB()->int	无	返回值为实际空闲的物理内存大小
	websocket	resource.getMemFreeKiB()	无	返回值为实际空闲的物理内存大小
	http	http://127.0.0.1:8888/resource/getMemFreeKiB	无	Json 返回值为实际空闲的物理内存大小
	java	double getMemFreeKiB()	无	double 返回值为实际空闲的物理内存大小

功能描述	接口类型	接口	入参	返回值
获取所有应用申请的虚拟内存总量	dbus	double getMemVirtAllocKiB()	无	double 返回值为所有应用申请的虚拟内存
	python	getMemVirtAllocKiB()->int	无	返回值为所有应用申请的虚拟内存
	websocket	resource.getMemVirtAllocKiB()	无	返回值为所有应用申请的虚拟内存
	http	http://127.0.0.1:8888/resource/getMemVirtAllocKiB	无	Json 返回值为所有应用申请的虚拟内存
	java	double getMemVirtAllocKiB()	无	double 返回值为所有应用申请的虚拟内存

功能描述	接口类型	接口	入参	返回值
获取系统中Swap分区总大小	dbus	double getMemSwapTotalKiB()	无	double 返回值为系统中Swap分区总大小
	python	getMemSwapTotalKiB()->int	无	返回值为系统中Swap分区总大小
	websocket	resource.getMemSwapTotalKiB()	无	返回值为系统中Swap分区总大小
	http	http://127.0.0.1:8888/resource/getMemSwapTotalKiB	无	Json 返回值为系统中Swap分区总大小
	java	double getMemSwapTotalKiB()	无	double 返回值为系统中Swap分区总大小

功能描述	接口类型	接口	入参	返回值
获取Swap分区使用率	dbus	double getMemSwapUsagePercent()	无	double 返回值为Swap分区使用率
	python	getMemSwapUsagePercent()->int	无	返回值为Swap分区使用率
	websocket	resource.getMemSwapUsagePercent()	无	返回值为Swap分区使用率
	http	http://127.0.0.1:8888/resource/getMemSwapUsagePercent	无	Json 返回值为Swap分区使用率
	java	double getMemSwapUsagePercent()	无	double

				返回值为Swap分区使用率

功能描述	接口类型	接口	入参	返回值
获取Swap分区使用率	dbus	double getMemSwapUsageKiB()	无	double 返回值为Swap分区使用量
	python	getMemSwapUsageKiB()->int	无	返回值为Swap分区使用量
	websocket	resource.getMemSwapUsageKiB()	无	返回值为Swap分区使用量
	http	http://127.0.0.1:8888/resource/getMemSwapUsageKiB	无	Json 返回值为Swap分区使用量
	java	double getMemSwapUsageKiB()	无	double 返回值为Swap分区使用量

功能描述	接口类型	接口	入参	返回值
获取Swap分区空闲大小	dbus	double getMemSwapFreeKiB()	无	double 返回值为Swap分区空闲大小
	python	getMemSwapFreeKiB()->int	无	返回值为Swap分区空闲大小
	websocket	resource.getMemSwapFreeKiB()	无	返回值为Swap分区空闲大小
	http	http://127.0.0.1:8888/resource/getMemSwapFreeKiB	无	Json 返回值为Swap分区空闲大小
	java	double getMemSwapFreeKiB()	无	double 返回值为Swap分区空闲大小

功能描述	接口类型	接口	入参	返回值
获取CPU瞬时使用率	dbus	double getCpuCurrentUsage()	无	double 返回值为CPU瞬时使用率
	python	getCpuCurrentUsage()->int	无	返回值为CPU瞬时使用率
	websocket	resource.getCpuCurrentUsage()	无	返回值为CPU瞬时使用率
	http	http://127.0.0.1:8888/resource/getCpuCurrentUsage	无	Json 返回值为CPU瞬时使用率
	java	double getCpuCurrentUsage()	无	double 返回值为CPU瞬时使用率

功能描述	接口类型	接口	入参	返回值
获取操作系统开机时长	dbus	QString getUpTime()	无	QString 返回值为开机天数
	python	getUpTime()->str	无	返回值为开机天数
	websocket	resource.getUpTime()	无	返回值为开机天数
	http	http://127.0.0.1:8888/resource/getUpTime	无	Json 返回值为开机天数
	java	String getUpTime()	无	String 返回值为开机天数

- **示例代码：**

```c
#----------------------------C语言示例----------------------------

#include "libkyrtinfo.h"
#include <stdio.h>

int main()
{
    printf("内存总大小：%lu KiB\n", kdk_rti_get_mem_res_total_KiB());
    printf("当前已用内存：%lu KiB，%f\n", kdk_rti_get_mem_res_usage_KiB(), kdk_rti_get_mem_res_usage_percent());
    printf("当前空闲内存：%lu KiB\n", kdk_rti_get_mem_res_free_KiB());
    printf("可用内存大小：%lu KiB\n", kdk_rti_get_mem_res_available_KiB());
    printf("应用总申请虚拟内存大小：%lu KiB\n", kdk_rti_get_mem_virt_alloc_KiB());
    printf("交换分区总量：%lu KiB\n", kdk_rti_get_mem_swap_total_KiB());
    printf("交换分区已用量：%lu KiB, %f\n", kdk_rti_get_mem_swap_usage_KiB(), kdk_rti_get_mem_swap_usage_percent());
    printf("交换分区剩余大小：%lu KiB\n", kdk_rti_get_mem_swap_free_KiB());
    printf("当前CPU使用率：%f\n", kdk_rti_get_cpu_current_usage());
    printf("共享内存大小：%lu KiB\n", kdk_rti_get_mem_shared_KiB());
    printf("高速缓存大小：%lu KiB\n", kdk_rti_get_mem_cached_KiB());
    printf("数据缓存大小：%lu KiB\n", kdk_rti_get_mem_buffers_KiB());
    printf("交换缓存区大小：%lu KiB\n", kdk_rti_get_mem_swap_cached_KiB());
    printf("活跃的缓冲文件大小：%lu KiB\n", kdk_rti_get_mem_active_KiB());
    printf("不活跃的缓冲文件大小：%lu KiB\n", kdk_rti_get_mem_inactive_KiB());
    printf("脏页大小：%lu KiB\n", kdk_rti_get_mem_dirty_KiB());
    printf("映射大小：%lu KiB\n", kdk_rti_get_mem_map_KiB());
    printf("内核数据结构缓存大小：%lu KiB\n", kdk_rti_get_mem_slab_KiB());
    unsigned int day, hour, min, sec;
    kdk_rti_get_uptime(&day, &hour, &min, &sec);
    printf("开机时长：%u天%u小时%u分钟%u秒\n", day, hour, min, sec);
    return 0;
}
```

```python
#----------------------------python语言示例----------------------------
from kysdk import Resource
resource = Resource()
# getMemTotalKiB接口
resource.getMemTotalKiB()
# getMemUsagePercent接口
resource.getMemUsagePercent()
# getMemUsageKiB接口
resource.getMemUsageKiB()
# getMemAvailableKiB接口
resource.getMemAvailableKiB()
# getMemFreeKiB接口
resource.getMemFreeKiB()
# getMemVirtAllocKiB接口
resource.getMemVirtAllocKiB()
# getMemSwapTotalKiB接口
resource.getMemSwapTotalKiB()
# getMemSwapUsagePercent接口
resource.getMemSwapUsagePercent()
# getMemSwapUsageKiB接口
resource.getMemSwapUsageKiB()
# getMemSwapFreeKiB接口
resource.getMemSwapFreeKiB()
# getCpuCurrentUsage接口
resource.getCpuCurrentUsage()
# getUpTime接口
resource.getUpTime()
```

```
//----------------------------Java语言示例----------------------------
import kylin.kysdk.java.ResourceMethod;
ResourceMethod obj = new ResourceMethod();
System.out.println("Total:" + obj.getMemTotalKiB());
System.out.println("UsagePer:" + obj.getMemSwapUsagePercent());
System.out.println("Usage:" + obj.getMemUsageKiB());
System.out.println("Avail:" + obj.getMemAvailableKiB());
System.out.println("Free:" + obj.getMemFreeKiB());
System.out.println("Virt:" + obj.getMemVirtAllocKiB());
System.out.println("SwapTotal:" + obj.getMemSwapTotalKiB());
System.out.println("SwapUsagePer:" + obj.getMemSwapUsagePercent());
System.out.println("SwapUsage:" + obj.getMemSwapUsageKiB());
System.out.println("SwageFree:" + obj.getMemSwapFreeKiB());
System.out.println("CpuUsage:" + obj.getCpuCurrentUsage());
System.out.println("UpTime:" + obj.getUpTime());
```

```
//----------------------------websocket语言示例----------------------------
    var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        new QWebChannel(websocket,function(channel){
                var resource = channel.objects.resource;
                //返回信息接收
                resource.sendText.connect(function(message) {
                    ...
                });
                //获得物理内存总大小
                resource.getMemTotalKiB();
                //获得物理内存使用率
                resource.getMemUsagePercent();
                //获得物理内存使用大小
                resource.getMemUsageKiB();
                //获得可用物理内存大小
                resource.getMemAvailableKiB();
                //获得空闲物理内存大小
                resource.getMemFreeKiB();
                //获得虚拟内存总量
                resource.getMemVirtAllocKiB();
                //获得Swap分区总大小
                resource.getMemSwapTotalKiB();
                //获得Swap分区使用率
                resource.getMemSwapUsagePercent();
                //获得Swap分区使用量
                resource.getMemSwapUsageKiB();
                //获得Swap分区空闲大小
                resource.getMemSwapFreeKiB();
                //获得CPU瞬时使用率
                resource.getCpuCurrentUsage();
                //获得开机天数
                resource.getUpTime();
            }
        );
    }
```

```
//-----------------------------http语言示例-----------------------------
1.http://127.0.0.1:8888/resource/getMemTotalKiB
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMemTotalKiB()信息
    }
2.http://127.0.0.1:8888/resource/getMemUsagePercent
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMemUsagePercent()信息
    }
3.http://127.0.0.1:8888/resource/getMemUsageKiB
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMemUsageKiB()信息
    }
4.http://127.0.0.1:8888/resource/getMemAvailableKiB
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMemAvailableKiB()信息
    }
5.http://127.0.0.1:8888/resource/getMemFreeKiB
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMemFreeKiB()信息
    }
6.http://127.0.0.1:8888/resource/getMemVirtAllocKiB
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMemVirtAllocKiB()信息
    }
7.http://127.0.0.1:8888/resource/getMemSwapTotalKiB
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMemSwapTotalKiB()信息
    }
8.http://127.0.0.1:8888/resource/getMemSwapUsagePercent
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMemSwapUsagePercent()信息
    }
9.http://127.0.0.1:8888/resource/getMemSwapUsageKiB
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMemSwapUsageKiB()信息
    }
10.http://127.0.0.1:8888/resource/getMemSwapFreeKiB
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMemSwapFreeKiB()信息
    }
11.http://127.0.0.1:8888/resource/getCpuCurrentUsage
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getCpuCurrentUsage()信息
    }
12.http://127.0.0.1:8888/resource/getUpTime
返回值: json
    {
```

```
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getUpTime()信息
    }
```

3.1.5.2获取进程信息

封装 C 接口获取到进程等资源信息。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkyprocess.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkyrtinfo.so
```

- **子模块信息：**

获取指定进程的 CPU 使用率(自2.0.0.0版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern float kdk_get_process_cpu_usage_percent(int proc_num)	
描述	获取指定进程的CPU使用率	
参数	proc_num	进程号
返回值	float	成功返回CPU使用率
	0.0	获取失败
备注	无	

获取指定进程的内存占用率(自2.0.0.0版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern float kdk_get_process_mem_usage_percent(int proc_num)	
描述	获取指定进程的内存占用率	
参数	proc_num	进程号
返回值	float	成功返回内存占用率
	0.0	获取失败
备注	无	

获取指定进程的进程状态(自2.0.0.0版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern char* kdk_get_process_status(int proc_num)	
描述	获取指定进程的进程状态	
参数	proc_num	进程号
返回值	char*	成功返回进程状态， 返回的字符串需要被 free 释放
	NULL	获取失败

备注	无

获取指定进程的端口号占用(自2.0.0.0版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern int kdk_get_process_port(int proc_num)	
描述	获取指定进程的端口号占用,只能取一个端口， 一个进程占用多个端口的情况在下一个版本中实现。	
参数	proc_num	进程号
返回值	int	成功返回使用的端口号
	0	获取失败
备注	无	

获取指定进程的启动时间(自2.0.0.0版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern char* kdk_get_process_start_time(int proc_num)	
描述	获取指定进程的启动时间	
参数	proc_num	进程号
返回值	char*	成功返回启动时间， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取指定进程的运行时间(自2.0.0.0版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern char* kdk_get_process_running_time(int proc_num);	
描述	获取指定进程的运行时间	
参数	proc_num	进程号
返回值	char*	成功返回运行时间， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取指定进程的 cpu 时间(自2.0.0.0版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern char* kdk_get_process_cpu_time(int proc_num)	
描述	获取指定进程的cpu时间	
参数	proc_num	进程号
返回值	char*	成功返回cpu时间， 返回的字符串需要被 free 释放

	NULL	获取失败
备注	无	

获取指定进程的 Command(自2.0.0.0版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern char* kdk_get_process_command(int proc_num)	
描述	获取指定进程的Command	
参数	proc_num	进程号
返回值	char*	成功返回Command， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取指定进程的属主(自2.0.0.0版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern char* kdk_get_process_user(int proc_num)	
描述	获取指定进程的属主	
参数	proc_num	进程号
返回值	char*	成功返回属主， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取指定进程的信息(自2.0.0.0版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern char** kdk_procname_get_process_infomation(char *proc_name)	
描述	获取指定进程的信息	
参数	proc_num	进程号
返回值	char**	进程的信息列表， 以NULL表示结尾， 由alloc生成， 需要被kdk_proc_freeall回收
	NULL	获取失败
备注	无	

获取系统中所有进程的信息(自2.0.0.0版本启用)

子模块	获取进程信息
接口类型	C
原型	char** kdk_get_process_all_information()
描述	获取系统中所有进程的信息

参数	无	无
返回值	char**	进程所有信息列表， 以NULL表示结尾， 由alloc生成， 需要被kdk_proc_freeall回收
	NULL	获取失败
备注	无	

回收字符串列表(自2.0.0.0版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern inline void kdk_proc_freeall(char **ptr)	
描述	回收字符串列表	
参数	ptr	字符串列表
返回值	无	无
	无	无
备注	无	

获取某一进程的名称(自2.2.3.5版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern char* kdk_get_process_name(int proc_num);	
描述	获取某一进程的名称	
参数	proc_num	进程号
返回值	char*	成功返回进程的名称， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取某一进程的id(自2.2.3.5版本启用)

子模块	获取进程信息	
接口类型	C	
原型	extern int kdk_get_process_id(char *proc_name);	
描述	获取某一进程的id，只能取一个id, 同名进程id的情况在下一个版本中实现	
参数	proc_name	进程的名称
返回值	int	成功返回进程号（id）
	0	获取失败
备注	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

服务名称： com.kylin.kysdk.service

路径名称： /com/kylin/kysdk/process

接口名称： com.kylin.kysdk.process

- **python导入方法**

```
from kysdk import Process
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.java.ProcessMethod;
```

功能描述	接口类型	接口	入参	返回值
获取指定进程的CPU使用率	dbus	double getProcInfoCpuUsage(int pid)	pid 进程号	double 返回值为进程CPU瞬时
	python	getProcInfoCpuUsage(pid)->int	pid 进程号	返回值为进程CPU瞬时
	websocket	process.getProcInfoCpuUsage(pid)	pid 进程号	返回值为进程CPU瞬时
	http	http://127.0.0.1:8888/process/getProcInfoCpuUsage?processid=parameter	parameter为进程号	Json 返回值为进程CPU瞬时
	java	double getProcInfoCpuUsage(int pid)	pid 进程号	double 返回值为进程CPU瞬时

功能描述	接口类型	接口	入参	返回值
获取指定进程的内存占用率	dbus	double getProcInfoMemUsage(int pid)	pid 进程号	double 返回值为进程内存占用
	python	getProcInfoMemUsage(pid)->int	pid 进程号	double 返回值为进程内存占用
	websocket	process.getProcInfoMemUsage(pid)	pid 进程号	返回值为进程内存占用
	http	http://127.0.0.1:8888/process/getProcInfoMemUsage?processid=parameter	parameter为进程号	Json 返回值为进程内存占用
	java	double getProcInfoMemUsage(int pid)	pid 进程号	double 返回值为进程内存占用

功能描述	接口类型	接口	入参	返回值
获取指定进程的进程状态	dbus	QString getProcInfoStatus(int pid)	pid 进程号	QString 返回值为进程状态
	python	getProcInfoStatus(pid)->str	pid 进程号	返回值为进程状态
	websocket	process.getProcInfoStatus(pid)	pid 进程号	
	http	http://127.0.0.1:8888/process/getProcInfoStatus?processid=parameter	parameter为进程号	Json 返回值为进程状态
	java	String getProcInfoStatus(int pid)	pid 进程号	String 返回值为进程状态

功能描述	接口类型	接口	入参	返回值
获取指定进程的端口号占用，只能取一个端口，一个进程占用多个端口的情况在下一个版本中实现。	dbus	int getProcInfoPort(int pid)	pid 进程号	int 返回值为进程使用的端口号
	python	getProcInfoPort(pid)->int	pid 进程号	返回值为进程使用的端口号
	websocket	process.getProcInfoPort(pid)	pid 进程号	返回值为进程使用的端口号
	http	http://127.0.0.1:8888/process/getProcInfoPort?processid=parameter	parameter为进程号	Json 返回值为进程使用的端口号
	java	int getProcInfoPort(int pid)	pid 进程号	int 返回值为进程使用的端口号

功能描述	接口类型	接口	入参	返回值
获取指定进程的启动时间	dbus	QString getProcInfoStartTime(int pid)	pid 进程号	QString 返回值为进程的启动时间
	python	getProcInfoStartTime(pid)->int	pid 进程号	返回值为进程的启动时间
	websocket	process.getProcInfoStartTime(pid)	pid 进程号	返回值为进程的启动时间
	http	http://127.0.0.1:8888/process/getProcInfoStartTime?processid=parameter	parameter为进程号	Json 返回值为进程的启动时间
	java	String getProcInfoStartTime(int pid)	pid 进程号	String 返回值为进程的启动时间

功能描述	接口类型	接口	入参	返回值
获取指定进程的运行时间	dbus	QString getProcInfoRunningTime(int pid)	pid 进程号	QString 返回值为进程的运行时间
	python	getProcInfoRunningTime(pid)->str	pid 进程号	返回值为进程的运行时间
	websocket	process.getProcInfoRunningTime(pid)	pid 进程号	返回值为进程的运行时间
	http	http://127.0.0.1:8888/process/getProcInfoRunningTime?processid=parameter	parameter为进程号	Json 返回值为进程的运行时间
	java	String getProcInfoRunningTime(int pid)	pid 进程号	String 返回值为进程的运行时间

功能描述	接口类型	接口	入参	返回值
获取指定进程的cpu时间	dbus	QString getProcInfoCpuTime(int pid)	pid 进程号	double 返回值为进程CPU时间
	python	getProcInfoCpuTime(pid)->int	pid 进程号	返回值为进程CPU时间
	websocket	process.getProcInfoCpuTime(pid)	pid 进程号	返回值为进程CPU时间
	http	http://127.0.0.1:8888/process/getProcInfoCpuTime?processid=parameter	parameter为进程号	Json 返回值为进程CPU时间
	java	String getProcInfoCpuTime(int pid)	pid 进程号	double 返回值为进程CPU时间

功能描述	接口类型	接口	入参	返回值
获取指定进程的Command	dbus	QString getProcInfoCmd(int pid)	pid 进程号	QString 返回值为进程的cmd
	python	getProcInfoCmd(pid)->str	pid 进程号	返回值为进程的cmd
	websocket	process.getProcInfoCmd(pid)	pid 进程号	返回值为进程的cmd
	http	http://127.0.0.1:8888/process/getProcInfoCmd?processid=parameter	parameter为进程号	Json 返回值为进程的cmd
	java	String getProcInfoCmd(int pid)	pid 进程号	String 返回值为进程的cmd

功能描述	接口类型	接口	入参	返回值
获取指定进程的属主	dbus	QString getProcInfoUser(int pid)	pid 进程号	QString 返回值为进程的属主
	python	getProcInfoUser(pid)->str	pid 进程号	返回值为进程的属主
	websocket	process.getProcInfoUser(pid)	pid 进程号	返回值为进程的属主
	http	http://127.0.0.1:8888/process/getProcInfoUser?processid=parameter	parameter为进程号	Json 返回值为进程的属主
	java	String getProcInfoUser(int pid)	pid 进程号	String 返回值为进程的属主

功能描述	接口类型	接口	入参	返回值
获取指定进程的信息	dbus	QStringList getProcInfo(const QString procName)	procName 进程名	QStringList 返回值为某进程所有信息
	python	getProcInfo(process_name)->list	process_name 进程名	返回值为某进程所有信息
	websocket	process.getProcInfo(vid)	vid 进程号	返回值为某进程所有信息
	http	http://127.0.0.1:8888/process/getProcInfo?processname=name	parameter为进程号	Json 返回值为某进程所有信息
	java	List getProcInfo(const QString procName)	procName 进程名	List 返回值为某进程所有信息

功能描述	接口类型	接口	入参	返回值
系统中所有进程信息	dbus	QStringList getAllProcInfo()	无	QStringList 返回值为系统中所有进程信息
	python	getAllProcInfo()->list	无	返回值为系统中所有进程信息
	websocket	process.getAllProcInfo()	无	返回值为系统中所有进程信息
	http	http://127.0.0.1:8888/process/getAllProcInfo	无	Json 返回值为系统中所有进程信息
	java	List getAllProcInfo()	无	List 返回值为系统中所有进程信息

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include "libkyprocess.h"
#include <stdio.h>
#include <stdlib.h>

int main()
{
    size_t index = 0;
    char *run_time = kdk_get_process_running_time(13366);
    char *cpu_time = kdk_get_process_cpu_time(13366);
    char *cmd = kdk_get_process_command(13366);
    char *start_time = kdk_get_process_start_time(13366);
    char *status = kdk_get_process_status(13366);
    char *user = kdk_get_process_user(13366);
    int *port = kdk_get_process_port_nums(13366);
    printf("获取某一进程的CPU利用率: %0.1f\n", kdk_get_process_cpu_usage_percent(13366));
    printf("获取某一进程的内存占用率: %0.1f\n", kdk_get_process_mem_usage_percent(13366));
    printf("获取某一进程的进程状态: %s\n", status);
    printf("获取某一进程的进程端口号: %d\n", kdk_get_process_port(3458));
    printf("获取某一进程的启动时间: %s\n", start_time);
    printf("获取某一进程的运行时间: %s\n", run_time);
    printf("获取某一进程的CPU时间: %s\n", cpu_time);
    printf("获取某一进程的Command: %s\n", cmd);
    printf("获取某一进程的属主: %s\n", user);
    free(status);
    free(run_time);
    free(cpu_time);
    free(cmd);
    free(start_time);
    free(user);

    if (port)
    {
        while (port[index])
        {
            printf("获取某一进程的进程端口号: %d\n", port[index]);
            index++;
        }
    }

    char** pid =  kdk_procname_get_process_infomation("systemd");
    if (NULL != pid)
    {
        index = 0;
        while (pid[index])
        {
            printf("pid %s\n", pid[index]);
            index++;
        }
        kdk_proc_freeall(pid);
    }

    char** info = kdk_get_process_all_information();
    size_t count = 0;
    while (info[count])
    {
        printf("No. %d\t %s\n", count + 1, info[count]);
        count ++;
    }
    kdk_proc_freeall(info);

    char *name = kdk_get_process_name(3458);
    printf("name = %s\n", name);
    free(name);

    int id =  kdk_get_process_id("systemd");
    printf("pid %d\n", id);

    return 0;
}
```

```
#----------------------------python语言示例----------------------------
from kysdk import Process
process = Process()
# getAllProcInfo接口",
process.getAllProcInfo()
# getProcInfo接口",
process.getProcInfo(process_name)
# getProcInfoCpuUsage接口",
process.getProcInfoCpuUsage(pid)
# getProcInfoMemUsage接口",
process.getProcInfoMemUsage(pid)
# getProcInfoStatus接口",
process.getProcInfoStatus(pid)
# getProcInfoPort接口",
process.getProcInfoPort(pid)
# getProcInfoStartTime接口",
process.getProcInfoStartTime(pid)
# getProcInfoRunningTime接口",
process.getProcInfoRunningTime(pid)
# getProcInfoCpuTime接口",
process.getProcInfoCpuTime(pid)
# getProcInfoCmd接口",
process.getProcInfoCmd(pid)
# getProcInfoUser接口",
process.getProcInfoUser(pid)
```

```
//----------------------------Java语言示例----------------------------
import kylin.kysdk.java.ProcessMethod;
ProcessMethod obj = new ProcessMethod();
System.out.println("CpuUsage:" + obj.getProcInfoCpuUsage(proc));
System.out.println("MemUsage:" + obj.getProcInfoMemUsage(proc));
System.out.println("State:" + obj.getProcInfoStatus(proc));
System.out.println("Port:" + obj.getProcInfoPort(proc));
System.out.println("Start:" + obj.getProcInfoStartTime(proc));
System.out.println("Running:" + obj.getProcInfoRunningTime(proc));
System.out.println("CpuTime:" + obj.getProcInfoCpuTime(proc));
System.out.println("Cmd:" + obj.getProcInfoCmd(proc));
System.out.println("User:" + obj.getProcInfoUser(proc));
System.out.println(obj.getProcInfo("code"));
System.out.println(obj.getAllProcInfo());
```

```
//----------------------------websocket语言示例----------------------------
    //pid 进程号
    var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        new QWebChannel(websocket,function(channel){
            var process = channel.objects.process;
            //返回信息接收
            process.sendText.connect(function(message) {
                ...
            });
            //获取CPU瞬时使用率
            process.getProcInfoCpuUsage(pid);
            //获取实时网速
            process.getProcInfoIoUsage(pid);
            //获取内存占用率
            process.getProcInfoMemUsage(pid);
            //获取进程状态
            process.getProcInfoStatus(pid);
            //获取进程使用的端口号
            process.getProcInfoPort(pid);
            //获取进程的启动时间
            process.getProcInfoStartTime(pid);
            //获取进程的运行时间
            process.getProcInfoRunningTime(pid);
            //获取CPU时间
            process.getProcInfoCpuTime(pid);
            //获取cmd
            process.getProcInfoCmd(pid);
            //获取属主
            process.getProcInfoUser(pid)
            //获取进程的属主
            process.getProcInfo(vid)
            //获取某进程所有信息
            process.getAllProcInfo()
            //获取系统中所有进程信息
        }
    }
```

```
//-----------------------------http语言示例-----------------------------
// parameter 为进程号
1.http://127.0.0.1:8888/process/getAllProcInfo
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getAllProcInfo()信息
    }
2.http://127.0.0.1:8888/process/getProcInfoCpuUsage?processid=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getProcInfoCpuUsage(parameter)信息
    }
3.http://127.0.0.1:8888/process/getProcInfoMemUsage?processid=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getProcInfoMemUsage(parameter)信息
    }
4.http://127.0.0.1:8888/process/getProcInfoStatus?processid=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getProcInfoStatus(parameter)信息
    }
5.http://127.0.0.1:8888/process/getProcInfoPort?processid=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getProcInfoPort(parameter)信息
    }
6.http://127.0.0.1:8888/process/getProcInfoStartTime?processid=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getProcInfoStartTime(parameter)信息
    }
7.http://127.0.0.1:8888/process/getProcInfoRunningTime?processid=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getProcInfoRunningTime(parameter)信息
    }
8.http://127.0.0.1:8888/process/getProcInfoCpuTime?processid=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getProcInfoCpuTime(parameter)信息
    }
9.http://127.0.0.1:8888/process/getProcInfoCmd?processid=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getProcInfoCmd(parameter)信息
    }
10.http://127.0.0.1:8888/process/getProcInfoUser?processid=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getProcInfoUser(parameter)信息
    }
// name为进程名
11.http://127.0.0.1:8888/process/getProcInfo?processname=name
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getProcInfo(name)信息
    }
```

3.1.6 获取操作系统基础信息

- **安装命令：**

```
sudo apt-get install libkysdk-sysinfo libkysdk-sysinfo-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-sysinfo
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKSYSINFO kysdk-sysinfo)
target_include_directories(demo PRIVATE ${KYSDKSYSINFO_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKSYSINFO_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKSYSINFO_LIBRARIES})
```

3.1.6.1获取操作系统基础信息

封装 C 接口获取系统名称、版本号、激活信息等。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkysysinfo.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkysysinfo.so
```

- **子模块信息：**

> 获取操作系统架构信息(自1.2.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_architecture()	
描述	获取系统架构信息	
参数	无	无
返回值	char*	成功返回系统架构，例：x86_64； 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

> 获取操作系统名称(自1.2.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_systemName()	
描述	获取操作系统名称	
参数	无	无

返回值	char*	成功返回系统名称，例：Kylin；返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取系统出厂详略版本号(自1.2.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_version(bool verbose)	
描述	获取系统出厂详略版本号	
参数	verbose	0获取简略版本号， 1获取详细版本号
返回值	char*	成功返回系统出厂版本号， 例：系统简略版本： xxxx桌面操作系统； 系统详细版本：Desktop-V10-Professional-Release-Build1-210203； 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取操作系统激活状态(自1.2.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern int kdk_system_get_activationStatus(int *status_error_num, int *date_error_num);	
描述	获取操作系统激活状态	
参数	status_error_num	用于接收激活状态的错误信息
	date_error_num	用于接收获取技术服务状态错误信息
返回值	int	2表示已过期；0表示未激活， 处于试用期；1表示已激活
	-1	接口内部错误
备注	无	

获取操作系统服务序列号(自1.2.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_serialNumber();	
描述	获取操作系统服务序列号	
参数	无	无
返回值	char*	成功返回操作系统服务序列号； 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取内核版本号(自1.2.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_kernelVersion()	
描述	获取内核版本号	
参数	无	无
返回值	char*	成功返回内核版本号； 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取当前登录用户的用户名(自1.2.0版本启用)
注：2.3.0.0版本后修改为系统当前登录的用户名，跟应用程序的启动用户没有关系，root用户启动的应用程序调用这个接口也不会返回root。

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_eUser()	
描述	获取当前登录用户的用户名（Effect User）	
参数	无	无
返回值	char*	成功返回当前登录用户的用户名； 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取当前登录用户的登录时间(自2.3.0.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_eUser_login_time();	
描述	获取当前登录用户的登录时间	
参数	无	无
返回值	char*	成功返回当前登录用户的登录时间； 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取操作系统项目编号名(自1.2.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_projectName();	
描述	获取操作系统项目编号名	
参数	无	无
返回值	char*	成功返回操作系统项目编号名； 返回的字符串需要被 free 释放
	NULL	获取失败

备注	无

获取操作系统项目子编号名(自1.2.1版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_projectSubName();	
描述	获取操作系统项目子编号名	
参数	无	无
返回值	char*	成功返回操作系统项目子编号名； 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取操作系统产品标识码(自1.2.1版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern unsigned int kdk_system_get_productFeatures();	
描述	获取操作系统产品标识码	
参数	无	无
返回值	unsigned int	返回标志码 0000：信息异常 0001：仅PC特性 0010：仅平板特性 0011：支持平板与PC特性
	无	无
备注	无	

获取操作系统宿主机的虚拟机类型(自1.2.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_hostVirtType();	
描述	获取操作系统宿主机的虚拟机类型	
参数	无	无
返回值	char*	成功返回一个字符串，字符串内容如下： [none, qemu, kvm, zvm, vmware,hyper-v, orcale virtualbox, xen, bochs, uml,parallels,bhyve,qnx,arcn,openvz,lxc,lxc-libvirt,systemd-nspawn,docker,podman,rkt,wsl] 其中 none 表示运行在物理机环境中； 其他字符串代表具体的虚拟环境类型。 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取操作系统宿主机的云平台类型(自1.2.0版本启用)

子模块	获取操作系统基础信息
接口类型	C

原型	extern char* kdk_system_get_hostCloudPlatform();	
描述	获取操作系统宿主机的云平台类型（天翼云在2.2.2.0版本开始支持， 华为特殊标识在2.3.0.0版本开始支持）	
参数	无	无
返回值	char*	成功返回一个字符串，字符串内容如下： [none, huawei, FC, HCS, HWC, HCSO , SCE, ctyun] 其中 none 表示运行在物理机或未知的云平台环境中； 其它字符串代表不同的云平台；'ctyun'代表天翼云，'huawei',代表华为云，'FC', 'HCS', 'HWC', 'HCSO' , 'SCE'是华为云特殊标识。 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

判断当前操作系统是否为专用机系统(自1.2.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern bool kdk_system_is_zyj(void);	
描述	判断当前镜像系统否为专用机系统	
参数	无	无
返回值	true	是
	false	不是
备注	无	

获取系统分辨率信息(自2.0.0.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char** kdk_system_get_resolving_power()	
描述	获取系统分辨率信息	
参数	无	无
返回值	char**	分辨率信息列表， 以NULL表示结尾， 由alloc生成， 需要被kdk_resolving_freeall回收
	NULL	获取失败
备注	无	

回收字符串列表(自2.0.0.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern inline void kdk_resolving_freeall(char **ptr)	
描述	回收字符串列表	
参数	ptr	字符串列表
返回值	无	无

	无	无
备注	无	

获取显示系统硬件版本类别(自2.1.1.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_systemCategory()	
描述	获取显示系统版本类别	
参数	无	无
返回值	char*	成功返回字符串系统版本硬件类别，版本类别有{Tablet, MaxTablet},Tablet-平板，MaxTablet-大屏； 未读到文件或字段返回字符串none。返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取系统版本号/补丁版本号(自1.2.1版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern version_t kdk_system_get_version_detaile();	
描述	获取系统版本号/补丁版本号	
参数	无	无
返回值	version_t	该系统版本号的详细信息结构体
	NULL	获取失败
备注	成员：os_version(char *)，描述：系统版本，例如：2303 成员：update_version(char *)，描述：补丁版本，例如：2303	

获取系统开机时间(自2.2.3.5版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char** kdk_system_get_startup_time();	
描述	获取系统开机时间	
参数	无	无
返回值	char**	成功返回开机时间，以NULL表示结尾，由alloc生成，需要被kdk_resolving_freeall回收
	NULL	获取失败
备注	无	

获取系统当日每次关机时间(自2.2.3.5版本启用)

子模块	获取操作系统基础信息
接口类型	C

原型	extern char** kdk_system_get_shutdown_time();	
描述	获取系统当日每次关机时间	
参数	无	无
返回值	char**	成功返回当日所有关机时间列表，以NULL表示结尾，由alloc生成，需要被kdk_resolving_freeall回收
	NULL	获取失败
备注	无	

获取整机制造商(自2.2.3.5版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char *kdk_get_host_vendor();	
描述	获取整机制造商	
参数	无	无
返回值	char*	成功返回整机制造商，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取整机型号(自2.2.3.5版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char *kdk_get_host_product();	
描述	获取整机型号	
参数	无	无
返回值	char*	成功返回整机型号，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取整机序列号(自2.2.3.5版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char *kdk_get_host_serial();	
描述	获取整机序列号	
参数	无	无
返回值	char*	成功返回整机序列号，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取主机名(自2.2.3.5版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char *kdk_system_get_hostName();	
描述	获取主机名	
参数	无	无
返回值	char*	成功返回主机名， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取操作系统位数(自2.2.3.5版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern int kdk_system_get_word();	
描述	获取操作系统位数	
参数	无	无
返回值	int	成功返回系统位数
	无	无
备注	无	

获取操作系统构建时间(自2.2.3.5版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char *kdk_system_get_buildTime();	
描述	获取操作系统构建时间	
参数	无	无
返回值	char*	成功返回操作系统构建时间， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取总线信息(自2.2.3.5版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern struct KPci *kdk_hw_get_pci_info();	
描述	获取总线信息	
参数	无	无
返回值	KPci*	pci总线信息结构体
	NULL	获取失败
备注	成员：slot_path(char)，描述：总线路径 成员：class_name(char)，描述：设备类 成员：product_name(char)，描述：设备名 成员：rev(unsigned char)，描述：版本	

	成员：ss_name(char)，描述：subsystem 成员：driver_use(char)，描述：使用的内核驱动 成员：modules(char**)，描述：内核模块 成员：module_count(int)，描述：内核模块个数

释放由kdk_hw_get_pci_info返回的pci总线信息结构体(自2.2.3.5版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern void kdk_hw_free_pci_info(struct KPci *info);	
描述	释放由kdk_hw_get_pci_info返回的pci总线信息结构体	
参数	info	由kdk_hw_get_pci_info返回的结构体指针
返回值	无	无
	无	无
备注	无	

获取应用场景(自2.2.3.5版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char *kdk_system_get_appScene();	
描述	获取应用场景	
参数	无	无
返回值	char*	成功返回字符串应用场景，应用场景有{EDU}， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取操作系统启动耗时(自2.4.1.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern char* kdk_system_get_startup_takeTime();	
描述	获取操作系统启动耗时	
参数	无	无
返回值	char*	成功返回操作系统启动耗时， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取用户注销状态(自2.4.1.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern bool kdk_system_user_logout_status(char *name);	
描述	获取用户注销状态	
参数	name	用户名

返回值	true	已注销
	false	未注销
备注	无	

注册监听用户切换事件的回调函数(自2.4.1.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern int kdk_system_register_switch_user_handle(CallBack call_back, void *user_data);	
描述	注册监听用户切换事件的回调函数	
参数	call_back	指向回调函数的函数指针
	user_data	传递给回调函数的用户数据结构指针
返回值	1	成功
	0	失败
备注	无	

注销监听用户切换事件的回调函数(自2.4.1.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern void kdk_system_unregister_switch_user_handle();	
描述	注册监听用户切换事件的回调函数	
参数	无	无
返回值	无	无
	无	无
备注	无	

获取系统中文件描述符数(自2.4.1.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern unsigned int kdk_system_get_file_descriptor();	
描述	获取系统中文件描述符数	
参数	无	无
返回值	unsigned int	成功返回cpu的文件描述符数
	0	获取失败
备注	无	

获取系统中进程数(自2.4.1.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern unsigned int kdk_system_get_process_nums();	
描述	获取系统中进程数	
参数	无	无
返回值	unsigned int	成功返回系统中进程数

	0	获取失败
备注	无	

获取系统中线程数(自2.4.1.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern unsigned int kdk_system_get_thread_nums();	
描述	获取系统中线程数	
参数	无	无
返回值	unsigned int	成功返回系统中线程数
	0	获取失败
备注	无	

获取CPU负载均衡(自2.4.1.0版本启用)

子模块	获取操作系统基础信息	
接口类型	C	
原型	extern kdk_loadavg kdk_system_get_loadavg();	
描述	获取CPU负载均衡	
参数	无	无
返回值	kdk_loadavg	返回cpu的1,5,15分钟负载均衡
	NULL	获取失败
备注	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

服务名称： com.kylin.kysdk.service

路径名称： /com/kylin/kysdk/sysinfo

接口名称： com.kylin.kysdk.sysinfo

- **python导入方法**

```
from kysdk import Sysinfo
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.java.SysInfoMethod;
```

功能描述	接口类型	接口	入参	返回值
获取系统架构信息	dbus	QString getSystemArchitecture()	无	QString 返回值为操作系统架构信息
	python	getSystemArchitecture()->str	无	返回值为操作系统架构信息
	websocket	sysinfo.getSystemArchitecture()	无	返回值为操作系统架构信息
	http	http://127.0.0.1:8888/sysinfo/getSystemArchitecture	无	Json 返回值为操作系统架构信息
	java	String getSystemArchitecture()	无	String 返回值为操作系统架构信息

功能描述	接口类型	接口	入参	返回值
获取操作系统名称	dbus	QString getSystemName()	无	double 返回值为操作系统名称
	python	getSystemName()->str	无	返回值为操作系统名称
	websocket	sysinfo.getSystemName()	无	返回值为操作系统名称
	http	http://127.0.0.1:8888/sysinfo/getSystemName	无	Json 返回值为操作系统名称
	java	String getSystemName()	无	String 返回值为操作系统名称

功能描述	接口类型	接口	入参	返回值
获取系统出厂详略版本号	dbus	QString getSystemVersion(bool verbose)	verbose 0获取简略版本号， 1获取详细版本号	QString 返回值为操作系统版本号
	python	getSystemVersion(defatul=0)->str	defatul 0获取简略版本号， 1获取详细版本号	返回值为操作系统版本号
	websocket	sysinfo.getSystemVersion(pid)	pid 0获取简略版本号， 1获取详细版本号	返回值为操作系统版本号
	http	http://127.0.0.1:8888/sysinfo/getSystemVersion?systemversion=parameter	parameter为 0获取简略版本号， 1获取详细版本号	Json 返回值为操作系统版本号
	java	String getSystemVersion(bool verbose)	verbose 0获取简略版本号， 1获取详细版本号	String 返回值为操作系统版本号

功能描述	接口类型	接口	入参	返回值
获取操作系统激活状态	dbus	int getSystemActivationStatus()	无	int 返回值为操作系统激活状态； 2表示已过期； 0表示未激活，处于试用期； 1表示已激活；-1表示接口内部错误
	python	getSystemActivationStatus()->int	无	返回值为操作系统激活状态； 2表示已过期； 0表示未激活，处于试用期； 1表示已激活；-1表示接口内部错误
	websocket	sysinfo.getSystemActivationStatus()	无	返回值为操作系统激活状态； 2表示已过期； 0表示未激活，处于试用期； 1表示已激活；-1表示接口内部错误
	http	http://127.0.0.1:8888/sysinfo/getSystemActivationStatus	无	Json 返回值为操作系统激活状态； 2表示已过期； 0表示未激活，处于试用期； 1表示已激活；-1表示接口内部错误
	java	int getSystemActivationStatus()	无	int 返回值为操作系统激活状态； 2表示已过期； 0表示未激活，处于试用期；

				1表示已激活；-1表示接口内部错误

功能描述	接口类型	接口	入参	返回值
获取操作系统服务序列号	dbus	QString getSystemSerialNumber()	无	QString 返回值为操作系统服务序列号
	python	getSystemSerialNumber()->str	无	返回值为操作系统服务序列号
	websocket	sysinfo.getSystemSerialNumber()	无	返回值为操作系统服务序列号
	http	http://127.0.0.1:8888/sysinfo/getSystemSerialNumber	无	Json 返回值为操作系统服务序列号
	java	String getSystemSerialNumber()	无	String 返回值为操作系统服务序列号

功能描述	接口类型	接口	入参	返回值
获取操作系统产品标识码	dbus	unsigned int getSystemProductFeatures()	无	unsigned int 返回值为操作系统产品标识码; 0000 信息异常; 0001 仅PC特性; 0010 仅平板特性; 0011 支持平板与PC特性
	python	getSystemProductFeatures()->int	无	返回值为操作系统产品标识码; 0000 信息异常; 0001 仅PC特性; 0010 仅平板特性; 0011 支持平板与PC特性
	websocket	sysinfo.getSystemProductFeatures()	无	返回值为操作系统产品标识码; 0000 信息异常; 0001 仅PC特性; 0010 仅平板特性; 0011 支持平板与PC特性
	http	http://127.0.0.1:8888/sysinfo/getSystemProductFeatures	无	Json 返回值为操作系统产品标识码; 0000 信息异常; 0001 仅PC特性; 0010 仅平板特性; 0011 支持平板与PC特性
	java	UInt32 getSystemProductFeatures()	无	UInt32 返回值为操作系统产品标识码; 0000 信息异常; 0001 仅PC特性; 0010 仅平板特性; 0011 支持平板与PC特性

功能描述	接口类型	接口	入参	返回值
获取操作系统宿主机的虚拟机类型	dbus	QString getSystemHostVirtType()	无	QString 返回值为操作系统宿主机的虚拟机
	python	getSystemHostVirtType()->str	无	返回值为操作系统宿主机的虚拟机
	websocket	sysinfo.getSystemHostVirtType()	无	返回值为操作系统宿主机的虚拟机
	http	http://127.0.0.1:8888/sysinfo/getSystemHostVirtType	无	Json 返回值为操作系统宿主机的虚拟机
	java	String getSystemHostVirtType()	无	String 返回值为操作系统宿主机的虚拟机

功能描述	接口类型	接口	入参	返回值
获取操作系统宿主机的云平台类型（天翼云在2.2.2.0版本开始支持，华为特殊标识在2.3.0.0版本开始支持）	dbus	QString getSystemHostCloudPlatform()	无	QString 返回值为操作系统宿主机的
	python	getSystemHostCloudPlatform()->str	无	返回值为操作系统宿主机的

	websocket	sysinfo.getSystemHostCloudPlatform()	无	返回值为操作系统宿主机的
	http	http://127.0.0.1:8888/sysinfo/getSystemHostCloudPlatform	无	Json 返回值为操作系统宿主机的
	java	String getSystemHostCloudPlatform()	无	String 返回值为操作系统宿主机的

功能描述	接口类型	接口	入参	返回值
获取系统版本号	dbus	QString getSystemOSVersion()	无	QString 返回值为操作系统OS版本号
	python	getSystemOSVersion()->str	无	返回值为操作系统OS版本号
	websocket	sysinfo.getSystemOSVersion()	无	返回值为操作系统OS版本号
	http	http://127.0.0.1:8888/sysinfo/getSystemOSVersion	无	Json 返回值为操作系统OS版本号
	java	String getSystemOSVersion()	无	String 返回值为操作系统OS版本号

功能描述	接口类型	接口	入参	返回值
获取系统补丁版本号	dbus	QString getSystemUpdateVersion()	无	QString 返回值为操作系统Update版本号
	python	getSystemUpdateVersion()->str	无	返回值为操作系统Update版本号
	websocket	sysinfo.getSystemUpdateVersion()	无	返回值为操作系统Update版本号
	http	http://127.0.0.1:8888/sysinfo/getSystemUpdateVersion	无	Json 返回值为操作系统Update版本号
	java	String getSystemUpdateVersion()	无	String 返回值为操作系统Update版本号

功能描述	接口类型	接口	入参	返回值
判断当前镜像系统否为专用机系统	dbus	bool getSystemIsZYJ()	无	bool 返回值为镜像系统是否为专用机系统; true代表是/false代表不是
	python	getSystemIsZYJ()->bool	无	返回值为镜像系统是否为专用机系统; true代表是/false代表不是
	websocket	sysinfo.getSystemIsZYJ()	无	返回值为镜像系统是否为专用机系统; true代表是/false代表不是
	http	http://127.0.0.1:8888/sysinfo/getSystemIsZYJ	无	Json 返回值镜像系统是否为专用机系统
	java	boolean getSystemIsZYJ()	无	boolean 返回值为镜像系统是否为专用机系统; true代表是/false代表不是

功能描述	接口类型	接口	入参	返回值
获取屏幕分辨率	session dbus	QStringList getSysLegalResolution()	无	QStringList 返回值为屏幕， 屏幕设置的分辨率， 屏幕支持的分辨率
	python	getSysLegalResolution()->list	无	返回值为屏幕， 屏幕设置的分辨率， 屏幕支持的分辨率
	websocket	sysinfo.getSysLegalResolution()	无	返回值为屏幕， 屏幕设置的分辨率， 屏幕支持的分辨率

	http	http://127.0.0.1:8888/sysinfo/getSysLegalResolution	无	Json 返回值为屏幕，屏幕设置的分辨率，屏幕支持的分辨率
	java	List getSysLegalResolution()	无	List 返回值为屏幕，屏幕设置的分辨率，屏幕支持的分辨率

- **示例代码：**

```c
#-----------------------------C语言示例-----------------------------
#include <libkysysinfo.h>
#include <stdio.h>
#include <stdlib.h>

void callbacke(char *old_user, char *new_user, void *userdata)
{
    printf("%s  %s\n", old_user, new_user);
}

int main()
{
    char *res = NULL;
    res = kdk_system_get_architecture();
    printf("架构: %s\n", res);
    free(res);

    res = kdk_system_get_systemName();
    printf("系统名称: %s\n", res);
    free(res);

    res = kdk_system_get_version(0);
    printf("系统简略版本: %s\n", res);
    free(res);

    res = kdk_system_get_version(1);
    printf("系统详细版本: %s\n", res);
    free(res);

    res = kdk_system_get_kernelVersion();
    printf("内核版本: %s\n", res);
    free(res);

    res = kdk_system_get_serialNumber();
    printf("序列号: %s\n", res);
    free(res);

    int p;
    int d;
    int act = kdk_system_get_activationStatus(&p,&d);
    printf("--------------\n");
    printf("激活状态码: %d\n",act);
    printf("激活状态: %s\n", act == 1 ? "已激活" : act == 0 ? "未激活" : "已过期");

    res = kdk_system_get_eUser();
    printf("当前用户: %s\n", res);
    // free(res);

    res = kdk_system_get_eUser_login_time();
    printf("登录时间: %s\n", res);

    res = kdk_system_get_eUser_login_time();
    printf("登录时间: %s\n", res);
    free(res);

    res = kdk_system_get_projectName();
    printf("项目编号名: %s\n", res);
    free(res);

    int zyj = kdk_system_is_zyj();
    printf("专用机: %s\n", zyj == 0 ? "非专用机":"专用机");

    res = kdk_system_get_hostVirtType();
    printf("虚拟机类型: %s\n", res);
    free(res);

    res = kdk_system_get_hostCloudPlatform();
    printf("云平台类型: %s\n", res);
    free(res);

    version_t test = kdk_system_get_version_detaile( );
```

```
printf("test.os_version = %s\n",test.os_version);
printf("test.update_version = %s\n",test.update_version);
free(test.os_version);
free(test.update_version);

char** name = kdk_system_get_resolving_power();
size_t count = 0;
if(NULL != name)
{
    // printf("name = %s\n", name);
    while (name[count])
    {
        printf("No. %d\t %s\n", count + 1, name[count]);
        count++;
    }
    kdk_resolving_freeall(name);
}

char *ver = kdk_system_get_systemCategory();
printf("系统硬件版本类别 = %s\n", ver);
free(ver);

// ver = kdk_system_get_cloudPlatformType();
// printf("云平台类型 = %s\n", ver);
// free(ver);

char **ud_time = kdk_system_get_startup_time();
count = 0;
while (ud_time[count])
{
    printf("开机时间 = %s\n", ud_time[count]);
    count++;
}
kdk_resolving_freeall(ud_time);

ud_time = kdk_system_get_shutdown_time();
count = 0;
while (ud_time[count])
{
    printf("关机时间 = %s\n", ud_time[count]);
    count++;
}
kdk_resolving_freeall(ud_time);

res = kdk_get_host_vendor();
printf("整机制造商 = %s\n", res);
free(res);

res = kdk_get_host_product();
printf("整机型号 = %s\n", res);
free(res);

res = kdk_get_host_serial();
printf("整机序列号 = %s\n", res);
free(res);

res = kdk_system_get_buildTime();
printf("构建时间: %s\n", res);
free(res);

res = kdk_system_get_hostName();
printf("主机名: %s\n", res);
free(res);

printf("系统位数: %d\n", kdk_system_get_word());

struct KPci *pci = kdk_hw_get_pci_info();
struct KPci *tmp = pci;
while (tmp)
{
    printf("slot_path : %s\n", tmp->slot_path);
```

```
        printf("\tclass name :%s\n", tmp->class_name);
        printf("\tproduct name :%s\n", tmp->product_name);
        printf("\trev :%02x\n", tmp->rev);
        printf("\tsubsystem name :%s\n", tmp->ss_name);
        printf("\tdriver user :%s\n", tmp->driver_use);
        printf("\tmodules :");
        for(int i = 0; i < tmp->module_count; i++)
        {
            printf("\t%s", tmp->modules[i]);
        }
        printf("\n");
        tmp = tmp->next;
    }
    kdk_hw_free_pci_info(pci);

    res = kdk_system_get_appScene();
    printf("应用场景: %s\n", res);
    free(res);

    bool status = kdk_system_user_logout_status("zm");
    printf("status = %d\n", status);

    kdk_system_register_switch_user_handle(callbacke,NULL);
    sleep(5);
    kdk_system_unregister_switch_user_handle();

    char* takeTime =kdk_system_get_startup_takeTime();
    printf("takeTime = %s\n", takeTime);
    free(takeTime);

    printf("系统中文件描述符数: %d\n", kdk_system_get_file_descriptor());
    printf("系统中进程数: %d\n", kdk_system_get_process_nums());
    printf("系统中线程数: %d\n", kdk_system_get_thread_nums());
    kdk_loadavg loadAvg = kdk_system_get_loadavg();
    printf("CPU 负载均衡 1m %0.2f, 5m %0.2f, 15m %0.2f\n", loadAvg.loadavg_1m, loadAvg.loadavg_5m, loadAvg.loadavg_15m);

    return 0;
}
```

```python
#----------------------------python语言示例----------------------------
from kysdk import Sysinfo
sysinfo = Sysinfo()
# getSystemArchitecture接口
sysinfo.getSystemArchitecture()
# getSystemName接口
sysinfo.getSystemName()
# getSystemActivationStatus接口
sysinfo.getSystemActivationStatus()
# getSystemSerialNumber接口
sysinfo.getSystemSerialNumber()
# getSystemKernelVersion接口
sysinfo.getSystemKernelVersion()
# getSystemEffectUser接口
sysinfo.getSystemEffectUser()
# getSystemProjectName接口
sysinfo.getSystemProjectName()
# getSystemProjectSubName接口
sysinfo.getSystemProjectSubName()
# getSystemProductFeatures接口
sysinfo.getSystemProductFeatures()
# getSystemHostVirtType接口
sysinfo.getSystemHostVirtType()
# getSystemHostCloudPlatform接口
sysinfo.getSystemHostCloudPlatform()
# getSystemOSVersion接口
sysinfo.getSystemOSVersion()
# getSystemUpdateVersion接口
sysinfo.getSystemUpdateVersion()
# getSystemIsZYJ接口
sysinfo.getSystemIsZYJ()
# getSysLegalResolution接口
sysinfo.getSysLegalResolution()
# getSystemVersion接口
sysinfo.getSystemVersion(defatul)

from kysdk import GetSysLegalResolution
resolution = GetSysLegalResolution()
resolution.getSysLegalResolution()
```

```java
//----------------------------Java语言示例----------------------------
import kylin.kysdk.java.SysInfoMethod;
SysInfoMethod obj = new SysInfoMethod();
System.out.println("Arch:" + obj.getSystemArchitecture());
System.out.println("Name:" + obj.getSystemName());
System.out.println("Version:" + obj.getSystemVersion(false));
System.out.println("Activation:" + obj.getSystemActivationStatus());
System.out.println("Serial:" + obj.getSystemSerialNumber());
System.out.println("KernelVer:" + obj.getSystemKernelVersion());
System.out.println("Effect:" + obj.getSystemEffectUser());
System.out.println("ProName:" + obj.getSystemProjectName());
System.out.println("SubName:" + obj.getSystemProjectSubName());
System.out.println("Product:" + obj.getSystemProductFeatures());
System.out.println("Virt:" + obj.getSystemHostVirtType());
System.out.println("Paltform:" + obj.getSystemHostCloudPlatform());
System.out.println("OSVer:" + obj.getSystemOSVersion());
System.out.println("UpdateVer:" + obj.getSystemUpdateVersion());
System.out.println("ZYJ:" + obj.getSystemIsZYJ

import kylin.kysdk.java.SysInfoMethod;
SysInfoMethod obj = new SysInfoMethod();
System.out.println("Resolution:" + obj.getSysLegalResolution());
```

```
//----------------------------websocket语言示例----------------------------
    //pid 版本号状态
    var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        new QWebChannel(websocket,function(channel){
            var sysinfo = channel.objects.sysinfo;
            //返回信息接收
            sysinfo.sendText.connect(function(message) {
                ...
            });
            //获取操作系统架构信息
            sysinfo.getSystemArchitecture();
            //获取操作系统名称
            sysinfo.getSystemName();
            //获取操作系统版本号
            sysinfo.getSystemVersion(pid);
            //获取操作系统激活状态
            sysinfo.getSystemActivationStatus();
            //获取操作系统服务序列号
            sysinfo.getSystemSerialNumber();
            //获取内核版本号
            sysinfo.getSystemKernelVersion();
            //获取当前登录用户的用户名
            sysinfo.getSystemEffectUser();
            //获取操作系统项目编号名
            sysinfo.getSystemProjectName();
            //获取操作系统项目子编号名
            sysinfo.getSystemProjectSubName();
            //获取操作系统产品标识码
            sysinfo.getSystemProductFeatures();
            //获取操作系统宿主机的虚拟机类型
            sysinfo.getSystemHostVirtType();
            //获取操作系统宿主机的云平台类型
            sysinfo.getSystemHostCloudPlatform();
            //获取操作系统OS版本号
            sysinfo.getSystemOSVersion();
            //获取操作系统Update版本号
            sysinfo.getSystemUpdateVersion();
            //获取镜像系统是否为专用机系统
            sysinfo.getSystemIsZYJ();
            }
        );
    }

    var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        new QWebChannel(websocket,function(channel){
            var sysinfo = channel.objects.sysinfo;
```

```
            //返回信息接收
            sysinfo.sendText.connect(function(message) {
                ...
            });
            //获取屏幕设置的分辨率
            sysinfo.getSysLegalResolution();
            }
        );
    }
```

```
//-----------------------------http语言示例-----------------------------
// parameter 为版本号状态
1.http://127.0.0.1:8888/sysinfo/getSystemArchitecture
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemArchitecture()信息
    }
2.http://127.0.0.1:8888/sysinfo/getSystemName
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemName()信息
    }
3.http://127.0.0.1:8888/sysinfo/getSystemVersion?systemversion=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemVersion(parameter)信息
    }
4.http://127.0.0.1:8888/sysinfo/getSystemActivationStatus
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemActivationStatus()信息
    }
5.http://127.0.0.1:8888/sysinfo/getSystemSerialNumber
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemSerialNumber()信息
    }
6.http://127.0.0.1:8888/sysinfo/getSystemKernelVersion
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemKernelVersion()信息
    }
7.http://127.0.0.1:8888/sysinfo/getSystemEffectUser
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemEffectUser()信息
    }
8.http://127.0.0.1:8888/sysinfo/getSystemProjectName
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemProjectName()信息
    }
9.http://127.0.0.1:8888/sysinfo/getSystemProjectSubName
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemProjectSubName()信息
    }
10.http://127.0.0.1:8888/sysinfo/getSystemProductFeatures
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemProductFeatures()信息
    }
11.http://127.0.0.1:8888/sysinfo/getSystemHostVirtType
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemHostVirtType()信息
    }
12.http://127.0.0.1:8888/sysinfo/getSystemHostCloudPlatform
返回值: json
```

```
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemHostCloudPlatform()信息
    }
13.http://127.0.0.1:8888/sysinfo/getSystemOSVersion
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemOSVersion()信息
    }
14.http://127.0.0.1:8888/sysinfo/getSystemUpdateVersion
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemUpdateVersion()信息
    }
15.http://127.0.0.1:8888/sysinfo/getSystemIsZYJ
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSystemIsZYJ()信息
    }


1.http://127.0.0.1:8888/sysinfo/getSysLegalResolution
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getSysLegalResolution()信息
    }
```

3.1.7 获取网络信息

- **安装命令：**

```
sudo apt-get install libkysdk-net libkysdk-net-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-net
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKNET kysdk-net)
target_include_directories(demo PRIVATE ${KYSDKNET_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKNET_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKNET_LIBRARIES})
```

3.1.7.1获取网络信息

封装 C 接口获取到网络信息。

- **头文件路径：**

```
 #include "kysdk/kysdk-system/libkynetinfo.h"
```

- **so库路径：**

```
 /usr/lib/aarch64-linux-gnu/libkynetinfo.so
```

- **dbus信息**
  - dbus 服务名称：com.kylin.kysdk.Net
  - 路径名称：/com/kylin/kysdk/Net
  - Interfaces：com.kylin.kysdk.Net
  - 信号：
    - 网络连接状态改变信号：NetStateChangeSignal(int state)
    - state：网络链接状态
- **子模块信息：**

获取网络端口状态(自2.0.0.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern int kdk_net_get_port_stat(int port);	
描述	获取端口状态	
参数	port	端口号
返回值	int	成功返回端口状态 FREE：0 TCP_ESTABLISHED:1 TCP_SYN_SENT:2 TCP_SYN_RECV:3 TCP_FIN_WAIT1:4 TCP_FIN_WAIT2:5 TCP_TIME_WAIT:6 TCP_CLOSE:7 TCP_CLOSE_WAIT:8 TCP_LAST_ACL:9 TCP_LISTEN:10 TCP_CLOSING:11
	-1	获取失败
备注	无	

获取多个网络端口状态(自2.0.0.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	int kdk_net_get_multiple_port_stat(int start, int end, int *result);	
描述	获取[start,end]的端口状态	
参数	start	开始端口号
	end	结束端口号
	result	具有足够空间的int数组接受端口状态
返回值	0	成功
	-1	失败
备注	无	

获取默认网关(自2.0.0.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern prouteMapList kdk_net_get_route();	
描述	获取默认网关	
参数	无	无

返回值	prouteMapList	网关信息
	NULL	获取失败

备注	获取网关信息返回值结构体说明：
	typedef struct route
	{
	char name[32]; //网卡名
	char addr[16]; //网关地址
	struct route *next;
	} routeMapList, *prouteMapList;	

获取防火墙信息(自2.0.0.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern pChain kdk_net_get_iptable_rules();	
描述	获取防火墙信息	
参数	无	无
返回值	pChain	防火墙信息
	NULL	获取失败

备注	获取防火墙信息返回值结构体说明：
	typedef struct chain_data
	{ char target[8]; //数据包匹配规则后应用的目标动作，
	目标可以是 ACCEPT（接受）、DROP（丢弃）
	或其他定义的动作
	char prot[4 + 1]; //数据包使用的协议，可能是 TCP、UDP、
	ICMP 等
	char opt[4 + 1]; //与规则关联的附加选项
	char source[32]; //数据包的源地址
	char destination[64]; //数据包的目标地址
	char option[128]; //其它选项(匹配条件，动作等)
	struct chain_data *next;
	} chainData, *pChainData;
	
	typedef struct chain
	{
	char total[16]; //链名
	char policy[32]; //链的规则
	pChainData data;
	struct chain *next;
	} Chain, *pChain;	

获取子网信息(自2.2.3.5版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern void kdk_net_get_netmask(IN const char *nc, OUT char *mask);	
描述	获取防火墙信息	
参数	nc	网卡名称，如eno1
	mask	子网掩码
返回值	无	无
	无	无
备注	无	

获取进程对应端口(自2.2.3.5版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char **kdk_net_get_proc_port();	
描述	获取进程对应端口	
参数	无	无
返回值	char **	成功返回进程对应端口, 以NULL表示结尾， 由alloc生成， 需要被kdk_net_freeall回收
	NULL	获取失败
备注	无	

获取所有UP状态的端口号(自2.2.3.5版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char **kdk_net_get_up_port();	
描述	获取所有UP状态的端口号	
参数	无	无
返回值	char **	成功返回所有UP状态的端口号, 以NULL表示结尾， 由alloc生成， 需要被kdk_net_freeall回收
	NULL	获取失败
备注	无	

获取hosts配置(自2.2.3.5版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char* kdk_net_get_hosts();	
描述	获取hosts配置	
参数	无	无
返回值	char *	成功返回hosts配置， 返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取hosts配置的域名映射(自2.2.3.5版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char* kdk_net_get_hosts_domain();	
描述	获取hosts配置的域名映射	
参数	无	无
返回值	char *	成功返回hosts配置的域名映射，

		返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取DNS配置文件(自2.2.3.5版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char** kdk_net_get_resolv_conf();	
描述	获取DNS配置文件	
参数	无	无
返回值	char**	成功返回DNS配置文件， 以NULL表示结尾， 由alloc生成， 需要被kdk_net_freeall回收
	NULL	获取失败
备注	无	

用于回收字符串列表(自2.2.3.5版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern inline void kdk_net_freeall(char **ptr);	
描述	用于回收字符串列表	
参数	ptr	字符串列表
返回值	无	无
	无	无
备注	无	

释放由kdk_net_get_route返回的网关信息结构体(自2.0.0.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern void kdk_net_free_route(prouteMapList list);	
描述	释放由kdk_net_get_route返回的网关信息结构体	
参数	list	由kdk_net_get_route返回的结构体
返回值	无	无
	无	无
备注	无	

释放由kdk_net_get_iptable_rules返回的防火墙信息结构体(自2.0.0.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern void kdk_net_free_chain(pChain list);	
描述	释放由kdk_net_get_iptable_rules返回的防火墙信息结构体	
参数	list	由kdk_net_get_iptable_rules返回的结构体
返回值	无	无

	无	无
备注	无	

获取网络连接类型(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern int kdk_net_get_link_type();	
描述	获取网络连接类型	
参数	无	无
返回值	int	成功返回网络连接类型
	无	无

备注	enum NetLinkType
	{
	LINK_TYPE_UNKNOWN,
	LINK_TYPE_WIFI,
	LINK_TYPE_ETHERNET,
	LINK_TYPE_WIFI_ETHERNET
	};	

检查指定网络连接的连接状态和可用性(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern int kdk_net_get_link_status(char *ip, char *port);	
描述	检查指定网络连接的连接状态和可用性	
参数	ip	ip地址
	port	端口号
返回值	1	连接
	0	未连接
备注	无	

获取网络当前已连接的所有网卡名称(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char** kdk_net_get_link_ncNmae();	
描述	获取网络当前已连接的所有网卡名称	
参数	无	无
返回值	char**	成功返回网卡名列表
	NULL	获取失败
备注	无	

获取网络当前使用网卡的网络协议(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char* kdk_net_get_primary_conType();	
描述	获取网络当前使用网卡的网络协议	

参数	无	无
返回值	char*	成功返回网络当前使用网卡的网络协议，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取无线设备的工作模式(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char* kdk_net_get_wifi_mode(char *nc);	
描述	获取无线设备的工作模式	
参数	nc	无线网卡名
返回值	char*	成功返回无线设备的工作模式，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取无线设备的工作频率(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char* kdk_net_get_wifi_freq(char *nc);	
描述	获取无线设备的工作频率	
参数	nc	无线网卡名
返回值	char*	成功返回无线设备的工作频率，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取无线设备的频道(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char* kdk_net_get_wifi_channel(char *nc);	
描述	获取无线设备的频道	
参数	nc	无线网卡名
返回值	char*	成功返回获取无线设备的频道，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取无线设备的工作速率(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char* kdk_net_get_wifi_rate(char *nc);	
描述	获取无线设备的工作速率	

参数	nc	无线网卡名
返回值	char*	成功返回无线设备的工作速率，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

获取无线设备的接收灵敏度的下限(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char* kdk_net_get_wifi_sens(char *nc);	
描述	获取无线设备的接收灵敏度的下限	
参数	nc	无线网卡名
返回值	char*	成功返回无线设备的接收灵敏度的下限，返回的字符串需要被 free 释放
	NULL	获取失败
备注	无	

域名转换为ip地址(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char** kdk_net_get_addr_by_name(char *name);	
描述	域名转换为ip地址	
参数	name	域名
返回值	char**	成功返回ip地址列表
	NULL	获取失败
备注	无	

ip地址转换为域名(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern char* kdk_net_get_name_by_addr(char *ip);	
描述	ip地址转换为域名	
参数	ip	ip地址
返回值	char*	成功返回域名
	NULL	获取失败
备注	无	

获取ipv4的dhcp配置(无线连接)(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern Dhcp4Config *kdk_net_get_ipv4_dhcp_config();	
描述	获取ipv4的dhcp配置(无线连接)	
参数	无	无
返回值	Dhcp4Config*	配置信息结构体

	NULL	获取失败

备注	typedef struct _Dhcp4Config
	{
	char *broadcast_address; // 广播地址
	int dad_wait_time; // 指定客户端在接口上等待重复地址检测（DAD）完成的最长时间（以秒为单位）
	int dhcp_lease_time; // 动态ip使用时限
	int dhcp_message_type; // DHCP报文的类型
	char *dhcp_server_identifier; // DNS服务器标识地址
	char *domain_name_servers; // 域名服务
	char *expiry; // 分配的IP地址的租约有效期限
	char *ip_address; // ip地址
	char *network_number; // 网络号
	char *next_server; // 下一个网络服务器IP地址
	int requested_broadcast_address; // 是否请求广播地址
	int requested_domain_name; // 是否请求域名
	int requested_domain_name_servers; // 是否请求域名服务器地址列表
	int requested_domain_search; // 是否请求域名搜索列表
	int requested_host_name; // 是否请求主机名
	int requested_interface_mtu; // 是否请求接口最大传输单元（MTU）
	int requested_ms_classless_static_routes; // 是否请求无类别静态路由选项
	int requested_netbios_name_servers; // 是否请求NetBIOS名称服务器地址
	int requested_netbios_scope; // 是否请求NetBIOS 范围
	int requested_ntp_servers; // 是否请求 NTP 服务器地址
	int requested_rfc3442_classless_static_routes; // 是否请求 RFC 3442 类无类别静态路由选项
	int requested_root_path; // 是否请求根路径信息
	int requested_routers; // 是否请求路由器（网关）地址
	int requested_static_routes; // 是否请求静态路由信息
	int requested_subnet_mask; // 是否请求子网掩码
	int requested_time_offset; // 是否请求时间偏移量
	int requested_wpad; // 是否请求 Web 代理自动发现（WPAD）
	char *routers; // 路由
	char *server_name; // 服务器名称
	char *subnet_mask; // 子网掩码
	} Dhcp4Config;	

获取ipv6的dhcp配置(无线连接)(自2.4.1.0版本启用)

子模块	获取网络信息	
接口类型	C	
原型	extern Dhcp6Config *kdk_net_get_ipv6_dhcp_config();	
描述	获取ipv6的dhcp配置(无线连接)	
参数	无	无
返回值	Dhcp6Config*	配置信息结构体
	NULL	获取失败

备注	typedef struct _Dhcp6Config
	{
	int dad_wait_time; // 指定客户端在接口上等待重复地址检测（DAD）完成的最长时间（以秒为单位）
	char *dhcp6_client_id; // 客户端标识符
	char *dhcp6_name_servers; // 域名服务器地址
	char *dhcp6_server_id; // DHCPv6 服务器标识符	

	int requested_dhcp6_client_id; // 是否请求客户端标识符
	int requested_dhcp_domain_search; //
	是否请求域名搜索列表
	int requested_dhp6_name_servers; //
	是否请求域名服务器地址
	} Dhcp6Config;

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

服务名称： com.kylin.kysdk.service

路径名称： /com/kylin/kysdk/net

接口名称： com.kylin.kysdk.net

- **python导入方法**

```
from kysdk import Net
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.java.NetMethod;
```

功能描述	接口类型	接口	入参	返回值
获取端口状态	dbus	int getPortState(int port)	port 端口号	int 返回值为网络端口状态（0-FREE， 1-TCP_ESTABLISHED, 2-TCP_SYN_SENT, 3-TCP_SYN_RECV, 4-TCP_FIN_WAIT1， 5-TCP_FIN_WAIT2， 6-TCP_TIME_WAIT, 7-TCP_CLOSE, 8-TCP_CLOSE_WAIT, 9-TCP_LAST_ACL， 10-TCP_LISTEN, 11-TCP_CLOSING）
	python	getPortState(port)->int	port 端口号	返回值为网络端口状态（0-FREE， 1-TCP_ESTABLISHED, 2-TCP_SYN_SENT, 3-TCP_SYN_RECV, 4-TCP_FIN_WAIT1， 5-TCP_FIN_WAIT2， 6-TCP_TIME_WAIT, 7-TCP_CLOSE, 8-TCP_CLOSE_WAIT, 9-TCP_LAST_ACL， 10-TCP_LISTEN, 11-TCP_CLOSING）
	websocket	net.getPortState(pid)	pid 端口号	返回值为网络端口状态（0-FREE， 1-TCP_ESTABLISHED, 2-TCP_SYN_SENT, 3-TCP_SYN_RECV, 4-TCP_FIN_WAIT1， 5-TCP_FIN_WAIT2， 6-TCP_TIME_WAIT, 7-TCP_CLOSE, 8-

				TCP_CLOSE_WAIT, 9-TCP_LAST_ACL， 10-TCP_LISTEN, 11-TCP_CLOSING）
	http	http://127.0.0.1:8888/net/getPortState?port=parameter	parameter为端口号	Json 返回值为网络端口状态（0-FREE， 1-TCP_ESTABLISHED, 2-TCP_SYN_SENT, 3-TCP_SYN_RECV, 4-TCP_FIN_WAIT1,5-TCP_FIN_WAIT2, 6-TCP_TIME_WAIT, 7-TCP_CLOSE, 8-TCP_CLOSE_WAIT, 9-TCP_LAST_ACL,10-TCP_LISTEN, 11-TCP_CLOSING）
	java	int getPortState(int port)	port 端口号	int 返回值为网络端口状态（0-FREE， 1-TCP_ESTABLISHED, 2-TCP_SYN_SENT, 3-TCP_SYN_RECV, 4-TCP_FIN_WAIT1， 5-TCP_FIN_WAIT2， 6-TCP_TIME_WAIT, 7-TCP_CLOSE, 8-TCP_CLOSE_WAIT, 9-TCP_LAST_ACL， 10-TCP_LISTEN, 11-TCP_CLOSING）

功能描述	接口类型	接口	入参	返回值
获取[start,end]的端口状态	dbus	QStringList getMultiplePortStat(int start, int end)	start 开始端口号 end 结束端口号	QStringList 返回值为[start,end]的端口状态
	python	getMultiplePortStat(start_port, end_port)->list	start_port 开始端口号 end_port 结束端口号	返回值为[start,end]的端口状态
	websocket	net.getMultiplePortStat(beginpid, endpid)	beginpid 开始端口号 endpid 结束端口号	返回值为[start,end]的端口状态
	http	http://127.0.0.1:8888/net/getMultiplePortStat?startid=start&endid=end	start 开始端口号 end 结束端口号	Json 返回值为[start,end]的端口状态
	java	List getMultiplePortStat(int start, int end)	start 开始端口号 end 结束端口号	List 返回值为[start,end]的端口状态

功能描述	接口类型	接口	入参	返回值
获取默认网关	dbus	QStringList getGatewayInfo()	无	QStringList 返回值为网关信息-名称，地址
	python	getGatewayInfo()->list	无	返回值为网关信息-名称，地址
	websocket	net.getGatewayInfo()	无	返回值为网关信息-名称，地址
	http	http://127.0.0.1:8888/net/getGatewayInfo	无	Json 返回值为网关信息-名称，地址
	java	List getGatewayInfo()	无	List 返回值为网关信息-

				名称，地址

功能描述	接口类型	接口	入参	返回值
获取防火墙信息	dbus	QStringList getFirewallState()	无	QStringList 返回值为防火墙信息
	python	getFirewallState()->list	无	返回值为防火墙信息
	websocket	net.getFirewallState()	无	返回值为防火墙信息
	http	http://127.0.0.1:8888/net/getFirewallState	无	Json 返回值为防火墙信息
	java	List getFirewallState()	无	List 返回值为防火墙信息

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include "libkynetinfo.h"
#include "stdio.h"
#include "stdlib.h"

int main(int argc, char *argv[])
{
    prouteMapList list = kdk_net_get_route(), tmp = list;
    while (tmp)
    {
        printf("%s default route %s\n", tmp->name, tmp->addr);
        tmp = tmp->next;
    }
    kdk_net_free_route(list);
    if (argc < 2)
        printf("please input test port num\n");
    else
        printf("port:%s's state %d\n", argv[1], kdk_net_get_port_stat(atoi(argv[1])));

    pChain chain = kdk_net_get_iptable_rules();
    pChain tmpchain = chain;
    while(tmpchain)
    {
        printf("Chain: %s\tpolice: %s\n",tmpchain->total,tmpchain->policy);
        pChainData tmpData = tmpchain->data;
        while (tmpData)
        {
            printf("target: %s\tprot: %s\topt: %s\tsource: %s\tdestination: %s\toption: %s\n",tmpData->target,tmpData->prot,tmpData
                                tmpData->source,tmpData->destination,tmpData->option);
            tmpData = tmpData->next;
            printf("\n");
        }
        tmpchain = tmpchain->next;
    }
    kdk_net_free_chain(chain);

    int result[65536];
    int net = kdk_net_get_multiple_port_stat(0, 1000, result);
    if(net == 0)
    {
        size_t count = 0;
        for(;count < 1000 ;count++)
        {
            printf("%d\t", result[count]);
        }
    }
    else
        printf("Failed With %d", net);
     printf("\n");

    char mask[32] = "\0";
    kdk_net_get_netmask("enaftgm1i0", mask);
    printf("mask = %s\n", mask);

    char **port = kdk_net_get_proc_port();
    size_t index = 0;
    while(port[index])
    {
        printf("No.%ld, port = %s\n", index+1, port[index]);
        index++;
    }
    kdk_net_freeall(port);

    port = kdk_net_get_up_port();
    index = 0;
    while(port[index])
    {
        printf("No.%ld, port = %s\n", index+1, port[index]);
        index++;
    }
```

```
    kdk_net_freeall(port);

    char *hosts = kdk_net_get_hosts();
    if (hosts != NULL)
    {
        printf("hosts = %s\n", hosts);
        free(hosts);
    }

    char *domain = kdk_net_get_hosts_domain();
    if (domain != NULL)
    {
        printf("domain = %s\n", domain);
        free(domain);
    }

    char **resolv = kdk_net_get_resolv_conf();
    index = 0;
    while(resolv[index])
    {
        printf("No.%ld, resolv = %s\n", index+1, resolv[index]);
        index++;
    }
    kdk_net_freeall(resolv);

    int link_type = kdk_net_get_link_type();
    printf("Link type : %d\n", link_type);

    char **link_list = kdk_net_get_link_ncNmae();
    if (NULL != link_list)
    {
        for (int i = 0; link_list[i]; i++)
        {
            printf("up interface :%s\n", link_list[i]);
            free(link_list[i]);
        }
        free(link_list);
    }

    char *contype = kdk_net_get_primary_conType();
    if (contype)
    {
        printf("PrimaryConnectionType :%s\n", contype);
        free(contype);
    }

#define WIRELESSNC "wlx004bf3de83b5"
    char *mode = kdk_net_get_wifi_mode(WIRELESSNC);
    if (mode)
    {
        printf("Wifi mode :%s\n", mode);
        free(mode);
    }

    char *freq = kdk_net_get_wifi_freq(WIRELESSNC);
    if (freq)
    {
        printf("Wifi freq :%s\n", freq);
        free(freq);
    }

    char *cahnnel = kdk_net_get_wifi_channel(WIRELESSNC);
    if (cahnnel)
    {
        printf("Wifi cahnnel :%s\n", cahnnel);
        free(cahnnel);
    }

    char *rate = kdk_net_get_wifi_rate(WIRELESSNC);
    if (rate)
    {
```

```
        printf("Wifi rate :%s\n", rate);
        free(rate);
    }

    char *sens = kdk_net_get_wifi_sens(WIRELESSNC);
    if (sens)
    {
        printf("Wifi sens :%s\n", sens);
        free(sens);
    }
#undef WIRELESSNC

    char **addrs = kdk_net_get_addr_by_name("www.google.com");
    if (NULL != addrs)
    {
        for (int i = 0; addrs[i]; i++)
        {
            printf("addr :%s\n", addrs[i]);
            free(addrs[i]);
        }
        free(addrs);
    }

    char *h_name = kdk_net_get_name_by_addr("108.160.170.26");
    if (h_name)
    {
        printf("h_name : %s\n", h_name);
        free(h_name);
    }

    Dhcp4Config *dhcp_4_config = kdk_net_get_ipv4_dhcp_config();
    if(NULL != dhcp_4_config)
    {
        printf("DHCP4 broadcast_address:%s\n",                          dhcp_4_config->broadcast_address);
        printf("DHCP4 dad_wait_time:%d\n",                              dhcp_4_config->dad_wait_time);
        printf("DHCP4 dhcp_lease_time:%d\n",                            dhcp_4_config->dhcp_lease_time);
        printf("DHCP4 dhcp_message_type:%d\n",                          dhcp_4_config->dhcp_message_type);
        printf("DHCP4 dhcp_server_identifier:%s\n",                     dhcp_4_config->dhcp_server_identifier);
        printf("DHCP4 domain_name_servers:%s\n",                        dhcp_4_config->domain_name_servers);
        printf("DHCP4 expiry:%s\n",                                     dhcp_4_config->expiry);
        printf("DHCP4 ip_address:%s\n",                                 dhcp_4_config->ip_address);
        printf("DHCP4 network_number:%s\n",                             dhcp_4_config->network_number);
        printf("DHCP4 next_server:%s\n",                                dhcp_4_config->next_server);
        printf("DHCP4 requested_broadcast_address:%d\n",                dhcp_4_config->requested_broadcast_address);
        printf("DHCP4 requested_domain_name:%d\n",                      dhcp_4_config->requested_domain_name);
        printf("DHCP4 requested_domain_name_servers:%d\n",              dhcp_4_config->requested_domain_name_servers);
        printf("DHCP4 requested_domain_search:%d\n",                    dhcp_4_config->requested_domain_search);
        printf("DHCP4 requested_host_name:%d\n",                        dhcp_4_config->requested_host_name);
        printf("DHCP4 requested_interface_mtu:%d\n",                    dhcp_4_config->requested_interface_mtu);
        printf("DHCP4 requested_ms_classless_static_routes:%d\n",       dhcp_4_config->requested_ms_classless_static_routes);
        printf("DHCP4 requested_netbios_name_servers:%d\n",             dhcp_4_config->requested_netbios_name_servers);
        printf("DHCP4 requested_netbios_scope:%d\n",                    dhcp_4_config->requested_netbios_scope);
        printf("DHCP4 requested_ntp_servers:%d\n",                      dhcp_4_config->requested_ntp_servers);
        printf("DHCP4 requested_rfc3442_classless_static_routes:%d\n",  dhcp_4_config->requested_rfc3442_classless_static_routes);
        printf("DHCP4 requested_root_path:%d\n",                        dhcp_4_config->requested_root_path);
        printf("DHCP4 requested_routers:%d\n",                          dhcp_4_config->requested_routers);
        printf("DHCP4 requested_static_routes:%d\n",                    dhcp_4_config->requested_static_routes);
        printf("DHCP4 requested_subnet_mask:%d\n",                      dhcp_4_config->requested_subnet_mask);
        printf("DHCP4 requested_time_offset:%d\n",                      dhcp_4_config->requested_time_offset);
        printf("DHCP4 requested_wpad:%d\n",                             dhcp_4_config->requested_wpad);
        printf("DHCP4 routers:%s\n",                                    dhcp_4_config->routers);
        printf("DHCP4 server_name:%s\n",                                dhcp_4_config->server_name);
        printf("DHCP4 subnet_mask:%s\n",                                dhcp_4_config->subnet_mask);
    }

    Dhcp6Config *dhcp_6_config = kdk_net_get_ipv6_dhcp_config();
    if(NULL != dhcp_6_config)
    {
        printf("DHCP6 dad_wait_time:%d\n",                  dhcp_6_config->dad_wait_time);
        printf("DHCP6 dhcp6_client_id:%s\n",                dhcp_6_config->dhcp6_client_id);
        printf("DHCP6 dhcp6_name_servers:%s\n",             dhcp_6_config->dhcp6_name_servers);
```

```
        printf("DHCP6 dhcp6_server_id:%s\n",                dhcp_6_config->dhcp6_server_id);
        printf("DHCP6 requested_dhcp6_client_id:%d\n",      dhcp_6_config->requested_dhcp6_client_id);
        printf("DHCP6 requested_dhcp_domain_search:%d\n",   dhcp_6_config->requested_dhcp_domain_search);
        printf("DHCP6 requested_dhp6_name_servers:%d\n\n",  dhcp_6_config->requested_dhp6_name_servers);
    }

    return 0;
}
```

```
#-----------------------------python语言示例-----------------------------
from kysdk import Net
net = Net()
# getGatewayInfo接口
net.getGatewayInfo()
# getFirewallState接口
net.getFirewallState()
# getPortState接口
net.getPortState(port)
# getMultiplePortStat接口
net.getMultiplePortStat(start_port, end_port)
```

```
//-----------------------------Java语言示例-----------------------------
import kylin.kysdk.java.NetMethod;
NetMethod obj = new NetMethod();
System.out.println("Port:" + obj.getPortState(5868));
System.out.println("Multiple:" + obj.getMultiplePortStat(0,500));
System.out.println("Gateway:" + obj.getGatewayInfo());
System.out.println("Firewall:" + obj.getFirewallState());
```

```
//-----------------------------websocket语言示例-----------------------------
    //pid 端口号
    var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        new QWebChannel(websocket,function(channel){
                var net = channel.objects.net;
                //返回信息接收
                net.sendText.connect(function(message) {
                    ...
                });
                //获取端口状态
                net.getPortState(pid);
                //获取接受端口状态
                net.getMultiplePortStat(beginpid, endpid);
                //获取网关信息
                net.getGatewayInfo();
                //获取防火墙信息
                net.getFirewallState();
            }
        );
    }
```

```
//-----------------------------http语言示例-----------------------------
// parameter 为端口号
1.http://127.0.0.1:8888/net/getPortState?port=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPortState(parameter)信息
    }
//start为开始端口号，end为结束端口号
2.http://127.0.0.1:8888/net/getMultiplePortStat?startid=start&endid=end
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getMultiplePortStat(start,end)信息
    }
3.http://127.0.0.1:8888/net/getGatewayInfo
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getGatewayInfo()信息
    }
4.http://127.0.0.1:8888/net/getFirewallState
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getFirewallState()信息
    }
```

3.1.8 获取系统运行时信息

- **安装命令：**

```
sudo apt-get install libkysdk-realtime libkysdk-realtime-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-realtime
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKREALTIME kysdk-realtime)
target_include_directories(demo PRIVATE ${KYSDKREALTIME_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKREALTIME_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKREALTIME_LIBRARIES})
```

3.1.8.1获取系统运行时信息

封装C接口获取网速、cpu 温度、硬盘温度等。

- **头文件路径：**

```
 #include "kysdk/kysdk-system/libkyrealtimeinfo.h"
```

- **so库路径：**

```
 /usr/lib/aarch64-linux-gnu/libkyrealtime.so
```

- **子模块信息：**

获取上传的瞬时网速(自2.0.0.0版本启用)

子模块	获取系统运行时信息	
接口类型	C	
原型	extern float kdk_real_get_net_speed(const char *nc);	
描述	获取上传的瞬时网速信息	
参数	nc	网卡名称，如eno1
返回值	float	成功返回上传的瞬时网速
	-1	获取失败
备注	无	

获取下载的瞬时网速信息(自2.2.3.5版本启用)

子模块	获取系统运行时信息	
接口类型	C	
原型	extern float kdk_real_get_if_speed(const char *nc);	
描述	获取下载的瞬时网速信息	
参数	nc	网卡名称，如eno1
返回值	float	成功返回下载的瞬时网速
	-1	获取失败
备注	无	

获取cpu温度(自2.0.0.0版本启用)

子模块	获取系统运行时信息	
接口类型	C	
原型	extern double kdk_real_get_cpu_temperature();	
描述	获取CPU温度	
参数	无	无
返回值	double	成功返回CPU温度
	-1	获取失败
备注	无	

获取磁盘温度(自2.0.0.0版本启用)

子模块	获取系统运行时信息	
接口类型	C	
原型	extern int kdk_real_get_disk_temperature(const char *name);	
描述	获取磁盘温度	
参数	name	磁盘名称， 应当是例如/dev/sda这种绝对路径
返回值	int	成功返回磁盘温度
	-1	获取失败
备注	无	

获取磁盘转速(自2.0.0.0版本启用)

子模块	获取系统运行时信息	
接口类型	C	
原型	extern int kdk_real_get_disk_rate(const char *name);	
描述	获取磁盘转速	
参数	name	磁盘名称， 应当是例如/dev/sda这种绝对路径
返回值	int	成功返回磁盘转速
	-1	获取失败
备注	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

> **服务名称：** com.kylin.kysdk.service
>
> **路径名称：** /com/kylin/kysdk/runinfo
>
> **接口名称：** com.kylin.kysdk.runinfo

- **python导入方法**

```
from kysdk import Runinfo
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.java.RunInfoMethod;
```

功能描述	接口类型	接口	入参	返回值
获取上传的瞬时网速信息	dbus	double getNetSpeed(const QString nc)	nc 网卡名	double 返回值为实时网速
	python	getNetSpeed(netcard)->int	netcard 网卡名	返回值为实时网速
	websocket	runinfo.getNetSpeed(pid)	pid 网卡名	返回值为实时网速
	http	http://127.0.0.1:8888/runinfo/getNetSpeed?netspeed=parameter	parameter为网卡名	Json 返回值为实时网速
	java	double getNetSpeed(String nc)	nc 网卡名	double 返回值为实时网速

功能描述	接口类型	接口	入参	返回值
获取磁盘转速	dbus	int getDiskRate(const QString diskpath)	diskpath 硬盘绝对路径	int 返回值为磁盘转速
	python	getDiskRate(disk_path)->int	disk_path 硬盘绝对路径	返回值为磁盘转速
	websocket	runinfo.getDiskRate(pid)	pid 硬盘绝对路径	返回值为磁盘转速
	http	http://127.0.0.1:8888/runinfo/getDiskRate?diskpath=parameter	parameter为硬盘绝对路径	Json 返回值为磁盘转速
	java	int getDiskRate(String diskpath)	diskpath 硬盘绝对路径	int 返回值为磁盘转速

功能描述	接口类型	接口	入参	返回值
获取CPU温度	dbus	double getCpuTemperature()	无	double 返回值为实时CPU温度
	python	getCpuTemperature()->int	无	返回值为实时CPU温度
	websocket	runinfo.getCpuTemperature()	无	返回值为实时CPU温度

	http	http://127.0.0.1:8888/runinfo/getCpuTemperature	无	Json 返回值为实时CPU温度
	java	double getCpuTemperature()	无	double 返回值为实时CPU温度

功能描述	接口类型	接口	入参	返回值
获取磁盘温度	dbus	double getDiskTemperature(const QString diskpath)	diskpath 硬盘绝对路径	double 返回值为实时硬盘温度
	python	getDiskTemperature(disk_path)->int	disk_path 硬盘绝对路径	返回值为实时硬盘温度
	websocket	runinfo.getDiskTemperature(pid)	pid 硬盘绝对路径	返回值为实时硬盘温度
	http	http://127.0.0.1:8888/runinfo/getDiskTemperature?diskpath=parameter	diskpath为硬盘绝对路径	Json 返回值为实时硬盘温度
	java	double getDiskTemperature(String diskpath)	diskpath 硬盘绝对路径	double 返回值为实时硬盘温度

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include "stdio.h"
#include "libkyrealtimeinfo.h"

int main()
{
    printf("netSpeed : %f\n", kdk_real_get_net_speed("enaftgm1i0"));
    printf("cpuTemp: %f\n", kdk_real_get_cpu_temperature());
    printf("diskTemp : %d\n", kdk_real_get_disk_temperature("/dev/nvme0n1"));
    printf("diskRate : %d\n", kdk_real_get_disk_rate("/dev/nvme0n1"));
    printf("ifnetSpeed : %f\n", kdk_real_get_if_speed("enaftgm1i0"));
    return 0;
}
```

```
#----------------------------python语言示例----------------------------
from kysdk import Runinfo
runinfo = Runinfo()
# getCpuTemperature接口
runinfo.getCpuTemperature()
# getNetSpeed接口
runinfo.getNetSpeed(netcard)
# getDiskRate接口
runinfo.getDiskRate(disk_path)
# getDiskTemperature接口
runinfo.getDiskTemperature(disk_path)
```

```
//----------------------------Java语言示例----------------------------
import kylin.kysdk.java.RunInfoMethod;
RunInfoMethod obj = new RunInfoMethod();
System.out.println("NetSpeed:" + obj.getNetSpeed("enp3s0"));
System.out.println("Rate:" + obj.getDiskRate("/dev/sda"));
System.out.println("CpuTemp:" + obj.getCpuTemperature());
System.out.println("DiskTemp:" + obj.getDiskTemperature("/dev/sda"));
```

```
//-----------------------------websocket语言示例-----------------------------
    //pid 网卡名
    var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        new QWebChannel(websocket,function(channel){
                var runinfo = channel.objects.runinfo;
                //返回信息接收
                runinfo.sendText.connect(function(message) {
                    ...
                });
                //获取实时网速
                runinfo.getNetSpeed(pid);
                //获取磁盘转速
                runinfo.getDiskRate(pid);
                //获取实时CPU温度
                runinfo.getCpuTemperature();
                //获取实时硬盘温度
                runinfo.getDiskTemperature(pid);
            }
        );
    }
```

```
//-----------------------------http语言示例-----------------------------
// parameter 为网卡名
1.http://127.0.0.1:8888/runinfo/getNetSpeed?netspeed=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getNetSpeed(parameter)信息
    }
// parameter 为硬盘绝对路径
2.http://127.0.0.1:8888/runinfo/getDiskRate?diskpath=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getDiskRate(parameter)信息
    }
3.http://127.0.0.1:8888/runinfo/getCpuTemperature
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getCpuTemperature()信息
    }
// parameter 为硬盘绝对路径
4.http://127.0.0.1:8888/runinfo/getDiskTemperature?diskpath=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getDiskTemperature(parameter)信息
    }
```

3.1.9 获取当前地理信息

- **安装命令:**

```
sudo apt-get install libkysdk-location libkysdk-location-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-location
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKLOCATION kysdk-location)
target_include_directories(demo PRIVATE ${KYSDKLOCATION_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKLOCATION_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKLOCATION_LIBRARIES})
```

3.1.9.1获取当前地理信息

封装 C 接口获取当前 ip 地址地理信息。

- **头文件路径：**

```
#include "kysdk/kysdk-system/libkylocation.h"
```

- **so库路径：**

```
/usr/lib/aarch64-linux-gnu/libkylocation.so
```

- **子模块信息：**

> 获取本机地理位置(自2.0.0.0版本启用)

子模块	获取当前地理信息	
接口类型	C	
原型	extern char *kdk_location_get();	
描述	获取本机的地理位置	
参数	无	无
返回值	char*	成功返回描述地理位置的json字符串
	NULL	获取失败
备注	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

> **服务名称：** com.kylin.kysdk.service
>
> **路径名称：** /com/kylin/kysdk/gps
>
> **接口名称：** com.kylin.kysdk.gps

- **python导入方法**

```
from kysdk import Gps
```

- **websocket调用**

- **Java导入方法**

```
import kylin.kysdk.java.GpsMethod;
```


功能描述	接口类型	接口	入参	返回值
获取本机的地理位置	dbus	QString getGPSInfo()	无	QString 返回值为国家城市经纬度
	python	getGPSInfo()->str	无	返回值为国家城市经纬度
	websocket	gps.getGPSInfo()	无	返回值为国家城市经纬度
	http	http://127.0.0.1:8888/gps/getGPSInfo	无	Json 返回值为国家城市经纬度
	java	String getGPSInfo()	无	String 返回值为国家城市经纬度

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include "stdio.h"
#include "libkylocation.h"

int main()
{
    char *location = kdk_loaction_get();
    printf("location: %s\n", location);
    if(location)
        free(location);
    return 0;
}
```

```
#----------------------------python语言示例----------------------------
from kysdk import Gps

gps = Gps()
# getCpuTemperature接口
gps.getGPSInfo()
```

```
//----------------------------Java语言示例----------------------------
import kylin.kysdk.java.GpsMethod;
GpsMethod obj = new GpsMethod();
System.out.println(obj.getGPSInfo());
```

```
//----------------------------websocket语言示例----------------------------
    var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        new QWebChannel(websocket,function(channel){
                var gps = channel.objects.gps;
                //返回信息接收
                gps.sendText.connect(function(message) {
                    ...
                });
                //接收国家城市经纬度
                gps.getGPSInfo();
            }
        );
    }
```

```
//----------------------------http语言示例----------------------------
1.http://127.0.0.1:8888/gps/getGPSInfo
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败)
        ResultMessage:返回dbus方法 getGPSInfo()信息
    }
```

3.2 电源管理

该层设计主要为应用电源管理接口。

- 安装命令

```
sudo apt install libkysdk-powermanagement libkysdk-powermanagement-dev
```

- 构建示例

（1）.pro 文件构建项目

```
qt 项目.pro 文件中增加:
```

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-powermanagement
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKPOWER kysdk-powermanagement)
target_include_directories(demo PRIVATE ${KYSDKPOWER_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKPOWER_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKPOWER_LIBRARIES})
```

3.2.1 锁屏设置

封装 C++接口提供锁屏设置能力。

- **引用头文件**

```
#include libkylockscreen.h
```

- **库文件路径**

`/usr/lib/*/libpowermanager.so`

'*'代表不同架构的架构目录名称，例如x86_64-linux-gnu

- **子模块信息**

> 设置禁止锁屏(自1.2.0版本启用)

子模块	锁屏设置	
接口类型	C++	
原型	uint32_t kdk_set_inhibit_lockscreen(const char *appName , const char *reason)	
描述	设置禁止锁屏	
参数	appName	应用名
	reason	禁止锁屏原因
返回值	0	失败
	int(>0)	成功
备注	无	

取消禁止锁屏(自1.2.0版本启用)

子模块	锁屏设置	
接口类型	C++	
原型	int kdk_un_inhibit_lockscreen(uint32_t flag)	
描述	取消禁止锁屏	
参数	flag	禁止锁屏接口的返回值
返回值	0	成功
	-1	失败
备注	无	

3.2.2 电源管理

封装c接口提供电源管理功能

- **引用头文件**

```
#include libkylockscreen.h
```

- **库文件路径**

`/usr/lib/*/libpowermanager.so`

'*'代表不同架构的架构目录名称，例如x86_64-linux-gnu

- **DBus服务**
  - 封装 C接口提供电源信息接口。
  - dbus 服务名称：com.kylin.kysdk.PowerManagerServer
  - 路径名称：/com/kylin/kysdk/PowerManager
  - Interfaces：com.kylin.kysdk.PowerManagerInterface
  - 信号：
    - 电源状态改变信号：PowerStateChangeSignal
- **子模块信息**

检测当前设备经过多长时间后息屏(自2.4.1.0版本启用)

子模块	电源管理模块	
接口类型	C	
原型	int kdk_power_get_screenidle_timeout()	
描述	检测当前设备经过多长时间后息屏	
参数	无	
返回值	int	设备经过多长时间后息屏，单位秒
备注	无	

检测当前设备是否处于活动状态(自2.4.1.0版本启用)

子模块	电源管理模块	
接口类型	C	
原型	int kdk_power_is_active()	
描述	检测当前设备是否处于活动状态	
参数	无	
返回值	0	活动状态
	int	非活动状态
备注	无	

获取当前设备的电源模式(自2.4.1.0版本启用)

子模块	电源管理模块	
接口类型	C	
原型	int kdk_power_get_mode()	
描述	获取当前设备的电源模式	
参数	无	
返回值	0	最佳性能
	1	平衡
	2	最佳能效
备注	无	

获取电源支持的休眠方式(自2.4.1.0版本启用)

子模块	电源管理模块	
接口类型	C	
原型	char* kdk_power_is_hibernate()	
描述	获取电源支持的休眠方式	
参数	无	
返回值	char*	成功返回电源支持的休眠方式，失败返回NULL
备注	返回值需要释放	

获取控制休眠的操作模式（挂起到磁盘）(自2.4.1.0版本启用)

子模块	电源管理模块	
接口类型	C	
原型	char* kdk_power_get_control_disk_status()	
描述	获取控制休眠的操作模式（挂起到磁盘）	
参数	无	
返回值	char*	成功返回控制休眠的操作模式，失败返回NULL
备注	返回值需要释放	

3.3 文件管理

该模块设计主要为开发者监控文件系统中文件变化，提供接口定义；减少

系统版本间由于部分文件系统变化而导致的差异；

- **安装命令**

```
$ sudo apt install libkysdk-filesystem libkysdk-filesystem-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-filesystem
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKFILESYSTEM kysdk-filesystem)target_inclu
```

3.3.1 文件监听功能

封装C++类提供文件监听功能

- **引入头文件**

```
#include /usr/include/kysdk/kysdk-system/libkyfilewatcher.hpp
```

- **库文件路径**

`/usr/lib/*/libkykyfilewatcher.so`

'*'代表不同架构的架构目录名称，例如x86_64-linux-gnu

- **子模块信息**

> 增加需要监听的文件路径(自1.2.0版本启用)

子模块	文件管理	
接口类型	C++	
原型	QStringList addWatchTargetRecursive(QString url, FileWatcherType type = PERIODIC, int attr = OPEN`|`CLOSE`|`MODIFY`|`DELETE, int maxdepth = 5, int recurType = DIR`|`REGULAR)	
描述	增加需要监听的文件路径，对于目录， 默认启用递归监听子目录	
参数	url	文件路径
	type	监听类型
	attr	监听属性
	maxdepth	递归最大深度
	recurType	递归类型
返回值	QStringList	监听文件列表
备注	无	

> 增加需要监听的文件列表(自1.2.0版本启用)

子模块	文件管理	
接口类型	C++	
原型	QStringList addWatchTargetListRecursive(QStringList urlList, FileWatcherType type = PERIODIC, int attr = OPEN`|`CLOSE`|`MODIFY`|`DELETE, int maxdepth = 5, int recurType = DIR`|`REGULAR)	
描述	增加需要监听的文件列表	
参数	urlList	文件列表
	type	监听类型
	attr	监听属性
	maxdepth	递归最大深度
	recurType	递归类型
返回值	QStringList	监听文件列表
备注	无	

从监听列表中移除指定的文件(自1.2.0版本启用)

子模块	文件管理	
接口类型	C++	
原型	QStringList removeWatchTargetRecursive(QString url, int maxdepth = 5, int recurType = ALLFILE)	
描述	从监听列表中移除指定的文件	
参数	url	文件路径
	maxdepth	递归最大深度
	recurType	递归类型
返回值	QStringList	监听文件列表
备注	无	

清空监听列表(自1.2.0版本启用)

子模块	文件管理
接口类型	C++
原型	void clearWatchList()
描述	清空监听列表
参数	无
返回值	无
备注	无

暂停文件监听(自1.2.0版本启用)

子模块	文件管理
接口类型	C++
原型	void pauseWatcher()
描述	暂停文件监听
参数	无
返回值	无
备注	无

恢复文件监听(自1.2.0版本启用)

子模块	文件管理
接口类型	C++
原型	void restartWatcher()
描述	恢复文件监听
参数	无
返回值	无
备注	无

3.4 AI 能力

该层设计主要为应用提供 AI 识别功能接口，为 OS 增加 OCR
文字识别功能；屏蔽需

引入 AI 功能带来的开发复杂性与调试难度。

- **安装命令**

```
$ sudo apt install libkysdk-ocr libkysdk-ocr-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-ocr
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKOCR kysdk-ocr)target_include_directories
```

3.4.1 OCR文字识别功能

- **引入头文件**

```
#include "kysdk/kysdk-system/libkyocr.hpp"
```

- **库文件路径**
  `/usr/lib/*/libkyocr.so`
  '*'代表不同架构的架构目录名称，例如x86_64-linux-gnu
- **子模块信息**

封装 C++接口提供 OCR 文字识别功能。

> 获取文字框(自1.2.0版本启用)

子模块	OCR文字识别功能	
接口类型	C++	
原型	Std::vector>> kdk::kdkOCR::getRect(const std::string &imagePath)	
描述	获取文字框	
参数	String	图片文件路径
返回值	Vector	文字框点的坐标矩阵、以及文字框的个数
备注	无	

> 获取文字内容(自1.2.0版本启用)

子模块	OCR文字识别功能	
接口类型	C++	
原型	Std::vector>>getCls(const std::string &imagePath,int nums)	
描述	获取文字内容	
参数	String	图片文件路径
	Vector	图片中的文字字符串
返回值	Int	同时处理的文字栈个数
备注	无	

3.5 打印机管理

该层主要为应用提供打印机管理接口，可以实现打印任务的下发，打印方式，打印任

务的取消。

- **安装命令**

```
$ sudo apt-get install libkysdk-hardware libkysdk-hardware-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-hardware
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKHARDWARE kysdk-hardware)target_include_d
```

3.5.1 获取打印机信息

- **引入头文件**

```
#include "kysdk/kysdk-system/libkyprinter.h"
```

- **库文件路径**

`/usr/lib/*/libkyprinter.so`

'*'代表不同架构的架构目录名称，例如x86_64-linux-gnu

- **子模块信息**

> 获取系统打印机列表(自2.0.0.0版本启用)

子模块	打印机管理	
接口类型	C	
原型	char** kdk_printer_get_list()	
描述	获取打印机列表	
参数	无	
返回值	char**	打印机名称列表， 由NULL字符串表示结尾； 失败返回NULL
备注	返回值需要被kdk_printer_freeall回收	

> 获取本地的所有可用打印机(打印机处于idle状态)(自2.0.0.0版本启用)

子模块	打印机管理	
接口类型	C	
原型	char** kdk_printer_get_available_list()	
描述	获取本地的所有可用打印机(打印机处于idle状态)	
参数	无	
返回值	char**	打印机名称列表， 由NULL字符串表示结尾； 失败返回NULL
备注	返回值需要被kdk_printer_freeall回收	

> 设置打印参数(自2.0.0.0版本启用)

子模块	打印机管理
接口类型	C

原型	void kdk_printer_set_options(int number_up,const char *media,const char *number_up_layout,const char *sides)	
描述	设置打印参数	
参数	number_up	一张纸打印几页
	media	纸张类型
	number_up_layout	cups属性
	sides	one-sided 单面 two_sided_long_dege 双面（长边翻转） two_sided_short_dege 双面（短边翻转）
返回值	无	
备注	无	

打印文件(自2.0.0.0版本启用)

子模块	打印机管理	
接口类型	C	
原型	int kdk_printer_print_local_file(const char *printername, char *filepath)	
描述	打印文件	
参数	printername	打印机名
	filepath	打印文件绝对路径
返回值	int	打印作业的id
备注	无	

取消打印作业(自2.0.0.0版本启用)

子模块	打印机管理	
接口类型	C	
原型	int kdk_printer_cancel_all_jobs(const char *printername)	
描述	取消打印作业	
参数	printername	打印机名
返回值	0	成功
	-1	失败
备注	无	

获取打印机状态(状态不是实时更新)(自2.0.0.0版本启用)

子模块	打印机管理	
接口类型	C	
原型	int kdk_printer_get_status(const char *printername)	
描述	获取打印机状态(状态不是实时更新)	
参数	printername	打印机名
返回值	int	打印机状态码
备注	无	

从url中解析下载的文件名(自2.0.0.0版本启用)

子模块	打印机管理	
接口类型	C	
原型	char* kdk_printer_get_filename(const char *url)	
描述	从url解析下载的文件名	
参数	url	下载链接
返回值	char *	解析的文件名
备注	无	

获取当前打印任务状态(自2.0.0.0版本启用)

子模块	打印机管理	
接口类型	C	
原型	int kdk_printer_get_job_status(const char *printername, int jobid)	
描述	获取当前打印任务状态 （发送打印任务之后下需要等待一会再获取状态, 状态不是实时更新）	
参数	printername	打印机名
	jobid	打印作业的id
返回值	int	打印任务状态码
备注	无	

下载网络文件到本地(自2.0.0.0版本启用)

子模块	打印机管理	
接口类型	C	
原型	int kdk_printer_print_download_remote_file(const char *url, const char *filepath)	
描述	下载网络文件到本地	
参数	url	网络文件
	filepath	要保存的文件路径， 用户自定义
返回值	int	下载状态码
备注	无	

用于回收字符串列表(自2.2.3版本启用)

子模块	打印机管理	
接口类型	C	
原型	void kdk_printer_freeall(char **ptr)	
描述	用于回收字符串列表	
参数	ptr	字符串列表
返回值	无	
备注	无	

- **其他接口类型接口：**
- **引用方法**
- **dbus服务名**

服务名称： com.kylin.kysdk.service

路径名称： /com/kylin/kysdk/print

接口名称： com.kylin.kysdk.print

- **python导入方法**

```
from kysdk import Print
```

- **websocket调用**
- **Java导入方法**

```
import kylin.kysdk.java.PrintMethod;
```

功能描述	接口类型	接口	入参	返回值
获取打印机列表	dbus	QStringList getPrinterList()	无	QStringList 返回值为系统打印机列表
	python	getPrinterList()->list	无	返回值为系统打印机列表
	websocket	printer.getPrinterList()	无	返回值为系统打印机列表
	http	http://127.0.0.1:8888/printer/getPrinterList	无	Json 返回值为系统打印机列表
	java	List getPrinterList()	无	List 返回值为系统打印机列表

功能描述	接口类型	接口	入参	返回值
获取本地的所有可用打印机 (打印机处于idle状态)	dbus	QStringList getPrinterAvailableList()	无	QStringList 返回值为系统可用打印机列表
	python	getPrinterAvailableList()->str	无	返回值为系统可用打印机列表
	websocket	printer.getPrinterAvailableList()	无	返回值为系统可用打印机列表
	http	http://127.0.0.1:8888/printer/getPrinterAvailableList	无	Json 返回值为系统可用打印机列表
	java	List getPrinterAvailableList()	无	List 返回值为系统可用打印机列表

功能描述	接口类型	接口	入参	返回
设置打印参数	dbus	void setPrinterOptions(int number_up, const QString media, const QString number_up_layout, const QString sides)	无	numbe 一张纸打 media 纸 number_u cups属性 单面:one-si (长边翻转):two g_ed 双面 (短边翻转):two rt_ed
	python	setPrinterOptions(num, paper_type, cups, option)	无	num 一张纸 paper_type cups cups属 option打印选项 sided, (长边翻转):two g_edge, (短边翻转):two rt_ed
	websocket	printer.setPrinterOptions(command)	无	parame 一张纸打

				parameter2 parameter3 如lrt parameter4 页 例如单面:one- (长边翻转):two g_ed 双面 (短边翻转):two rt_ed
	http	http://127.0.0.1:8888/printer/setPrinterOptions?pagenum=parameter1&pagetype=parameter2&cputype=parameter3&sidetype=parameter4	无	parame 一张纸打 parameter2 parameter3 如lrt parameter4 页 例如单面:one- (长边翻转):two g_ed 双面 (短边翻转):two rt_ed
	java	void setPrinterOptions(int number_up, String media, String number_up_layout, String sides)	无	numbe 一张纸打 media 纸 number_up cups属性 单面:one-si (长边翻转):two g_ed 双面 (短边翻转):two rt_ed

功能描述	接口类型	接口	入参	返回值
打印文件	dbus	int getPrinterPrintLocalFile(const QString printername, const QString filepath)	printername 打印机名 filepath 打印文件绝对路径	int 打印作业的id
	python	getPrinterPrintLocalFile(print_name, file_path)->int	print_name 打印机名 file_path 打印文件绝对路径	打印作业的id
	websocket	printer.getPrinterPrintLocalFile(command)	printername 打印机名 ","+ filepath 打印文件绝对路径	打印作业的id
	http	http://127.0.0.1:8888/printer/getPrinterPrintLocalFile?printername=parameter1&filepath=parameter2	parameter1 打印机名 parameter2 打印文件绝对路径	Json 返回打印文件的id
	java	int getPrinterPrintLocalFile(String printername, String filepath)	printername 打印机名 filepath 打印文件绝对路径	打印作业的id

功能描述	接口类型	接口	入参	返回值
取消打印作业	dbus	int getPrinterCancelAllJobs(const QString printername)	printername 打印机名	int 成功：0/ 失败：-1
	python	getPrinterCancelAllJobs(print_name)->int	print_name 打印机名	成功：0/失败：-1
	websocket	printer.getPrinterCancelAllJobs(printer_type)	printer_type 打印机名	成功：0/失败：-1

	http	http://127.0.0.1:8888/printer/getPrinterCancelAllJobs?printername=parameter	parameter为打印机名	Json 返回值为成功：0/失败：-1
	java	int getPrinterCancelAllJobs(String printername)	printername 打印机名	int 成功：0/失败：-1

功能描述	接口类型	接口	入参	返回值
获取打印机状态 (状态不是实时更新)	dbus	int getPrinterStatus(const QString printername)	printername 打印机名	int 打印机状态码
	python	getPrinterStatus(print_name)->int	print_name 打印机名	打印机状态码
	websocket	printer.getPrinterStatus(printer_type)	printer_type 打印机名	打印机状态码
	http	http://127.0.0.1:8888/printer/getPrinterStatus?printername=parameter	parameter为打印机名	Json 返回值为打印机状态码
	java	int getPrinterStatus(String printername)	printername 打印机名	int 打印机状态码

功能描述	接口类型	接口	入参	返回值
从url解析下载的文件名	dbus	QString getPrinterFilename(const QString url)	url 下载链接	QString 解析出来的文件名
	python	getPrinterFilename(url)->str	url 下载链接	解析出来的文件名
	websocket	printer.getPrinterFilename(download_link)	download_link 下载链接	解析出来的文件名
	http	http://127.0.0.1:8888/printer/getPrinterFilename?printername=parameter	parameter为下载链接	Json 返回值为下载的文件名
	java	String getPrinterFilename(String url)	url 下载链接	String 解析出来的文件名

功能描述	接口类型	接口	入参	返回值
获取当前打印任务状态（发送打印任务之后下需要等待一会再获取状态,状态不是实时更新）	dbus	int getPrinterJobStatus(const QString printername, int jobid)	printername 打印机名 jobid 打印作业的id	int 打印任务状态码
	python	getPrinterJobStatus(print_name, jobid)->int	print_name 打印机名 jobid 打印作业的id	打印任务状态码
	websocket	printer.getPrinterJobStatus(command)	printername 打印机名 jobid 打印作业的id	打印任务状态码
	http	http://127.0.0.1:8888/printer/getPrinterJobStatus?printername=parameter1&jobid=parameter2	parameter1 为打印机名 parameter2 为打印作业的id	Json 返回值为打印任务状态码
	java	int getPrinterJobStatus(String printername, int jobid)	printername 打印机名 jobid 打印作业的id	int 打印任务状态码

功能描述	接口类型	接口	入参	返回值
下载网络文件到本地	dbus	int getPrinterDownloadRemoteFile(const QString url, const QString filepath)	url 网络文件 filepath 要保存的文件路径,用户自定义	int 下载状态码
	python	getPrinterDownloadRemoteFile(url, path)->int	url 网络文件 path	int 下载状态码

			要保存的文件路径, 用户自定义	
	websocket	printer.getPrinterDownloadRemoteFile(command)	url 网络文件 filepath 要保存的文件路径, 用户自定义	下载状态码
	http	http://127.0.0.1:8888/printer/getPrinterDownloadRemoteFile? file=parameter1&filepath=parameter2	parameter1 网络文件 parameter2 要保存的文件路径, 用户自定义	Json 返回值为下载状态码
	java	int getPrinterDownloadRemoteFile(String url, String filepath)	url 网络文件 filepath 要保存的文件路径, 用户自定义	int 下载状态码

- **示例代码：**

```c
#----------------------------C语言示例----------------------------
#include "libkyprinter.h"
#include <stdio.h>

int main()
{
    int index = 0;
    //获取打印机列表
    char **printers = kdk_printer_get_available_list();
    while (printers[index])
    {
        printf("%zd: %s\n", index + 1, printers[index]);

        //取消当前打印机所有打印任务
        kdk_printer_cancel_all_jobs(printers[index]);

        //获取url对应的文件名
        char *url = "http://www.cztouch.com/upfiles/soft/testpdf.pdf";
        int res = -1;
        char *filename = kdk_printer_get_filename(url); //从完整路径名中解析出文件名称，例如：/home/test/abc.txt,解析完成后为abc.txt
        printf("filename = %s\n", filename);

        //设置打印参数
        kdk_printer_set_options(2, "A4", "lrtb", "two-sided-long-edge");

        //下载
        res = kdk_printer_print_download_remote_file(url, filename);
        printf("[%s] res = %d\n", __FUNCTION__, res);
        printf("[%s] filepath = %s\n", __FUNCTION__, filename);

        //打印
        int job_id = kdk_printer_print_local_file(printers[index], filename);
        if (job_id == 0)
        {
            //打印失败
            printf("[%s] create print job fail\n", __FUNCTION__);
        }
        else
        {
            //打印成功，等待一会获取任务状态
            sleep(10);
            int status = kdk_printer_get_job_status(printers[index], job_id);
        }

        index++;
        free(filename);
    }
    kdk_printer_freeall(printers);
    return 0;
}
```

```python
#----------------------------python语言示例----------------------------
from kysdk import Print
printer = Print()
# getPrinterList接口
 printer.getPrinterList()
# getPrinterAvailableList接口
 printer.getPrinterAvailableList()
# getPrinterCancelAllJobs接口
 printer.getPrinterCancelAllJobs(print_name)
# getPrinterStatus接口
 printer.getPrinterStatus(print_name)
# getPrinterFilename接口
 printer.getPrinterFilename(url)
# getPrinterPrintLocalFile接口
 printer.getPrinterPrintLocalFile(print_name, file_path)
# getPrinterJobStatus接口
 printer.getPrinterJobStatus(print_name, jobid))
# getPrinterDownloadRemoteFile接口
 printer.getPrinterDownloadRemoteFile(url, path)
# setPrinterSetOptions接口
printer.setPrinterSetOptions(num, paper_type, cups, option)
```

```java
//----------------------------Java语言示例----------------------------
import kylin.kysdk.java.PrintMethod;
PrintMethod obj = new PrintMethod();
System.out.println("PrinterList:" + obj.getPrinterList());
System.out.println("AvailableList:" + obj.getPrinterAvailableList());
obj.setPrinterOptions(1,"A4","","one-sided");
System.out.println("LocalFile:" + obj.getPrinterPrintLocalFile("",""));
System.out.println("CancleJob:" + obj.getPrinterCancelAllJobs(""));
System.out.println("Statue:" + obj.getPrinterStatus(""));
System.out.println("FileName:" + obj.getPrinterFilename(""));
System.out.println("JobStatus:" + obj.getPrinterJobStatus("",0));
System.out.println("RemoteFile:" + obj.getPrinterDownloadRemoteFile("",""));
```

```
//----------------------------websocket语言示例----------------------------
    var websocket_url = 'ws://localhost:12345';
    var websocket = null;

    if (websocket === null) {
        websocket = new WebSocket(websocket_url);
        websocket.onopen = function () {
            console.log("connect websocketserver success");
        }
    } else {
        websocket.close();
        websocket = null;
    }
    function xxx() {
        new QWebChannel(websocket,function(channel){
                var printer = channel.objects.print;
                //返回消息接收
                printer.sendText.connect(function(message) {
                ...
                });
                //获取打印机列表
                printer.getPrinterList();
                //获取可用打印机列表
                printer.getPrinterAvailableList();
                //设置打印参数
                printer.setPrinterOptions(command);
                //打印本地文件
                printer.getPrinterPrintLocalFile(command);
                //取消所有作业
                printer.getPrinterCancelAllJobs(printer_type);
                //获取打印机状态
                printer.getPrinterStatus(printer_type);
                //打印文件名
                printer.getPrinterFilename(download_link);
                //获取打印机作业状态
                printer.getPrinterJobStatus(command);
                //打印远程文件
                printer.getPrinterDownloadRemoteFile(command);
            }
        );
    }
```

```
//-----------------------------http语言示例-----------------------------
1.http://127.0.0.1:8888/printer/getPrinterList
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPrinterList()信息
    }
2.http://127.0.0.1:8888/printer/getPrinterAvailableList
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPrinterAvailableList()信息
    }
//parameter1 为打印页数; parameter2 为纸张类型; parameter3 为cups属性; parameter4 单双面打印设置
3.http://127.0.0.1:8888/printer/setPrinterOptions?pagenum=parameter1&pagetype=parameter2&cputype=parameter3&sidetype=parameter4
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPrinterAvailableList(parameter1,parameter2,parameter3,parameter4)信息
    }
//parameter1 为打印机名; parameter2 为打印文件绝对路径
4.http://127.0.0.1:8888/printer/getPrinterPrintLocalFile?printername=parameter1&filepath=parameter2
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPrinterPrintLocalFile(parameter1,parameter2)信息
    }
//parameter 为打印机名
5.http://127.0.0.1:8888/printer/getPrinterCancelAllJobs?printername=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPrinterCancelAllJobs(parameter)信息
    }
//parameter 为打印机名
6.http://127.0.0.1:8888/printer/getPrinterStatus?printername=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPrinterStatus(parameter)信息
    }
//parameter 为下载链接
7.http://127.0.0.1:8888/printer/getPrinterFilename?printername=parameter
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPrinterFilename(parameter)信息
    }
//parameter1 为打印机名; parameter2 为打印作业的id
8.http://127.0.0.1:8888/printer/getPrinterJobStatus?printername=parameter1&jobid=parameter2
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPrinterJobStatus(parameter1,parameter2)信息
    }
//parameter1 为网络文件; parameter2 为要保存的文件路径，用户自定义
9.http://127.0.0.1:8888/printer/getPrinterDownloadRemoteFile?file=parameter1&filepath=parameter2
返回值: json
    {
        Result:连接dbus服务是否成功(0 成功 -1 失败),
        ResultMessage:返回dbus方法 getPrinterDownloadRemoteFile(parameter1,parameter2)信息
    }
```

3.6 进程资源管理

整个进程资源管理模块自2.0.0.0版本启用

该层主要主要负责对系统中的进程资源进行管理，目前仅有设置关键进程的oom分值的接口，来防止系统触发oom的时候，关键进程被杀掉。

- **安装命令**

```
$ sudo apt install libkysdk-processmanage
```

3.6.1 关键进程防杀

调用该dbus方法的进程需要将进程信息添加进白名单，才可调用成功，白名单现在由sdk统一管理。

- **dbus服务**
  - dbus服务名称：com.kylin.processStateManage
  - 路径名称：com/kylin/processStateManage
  - 接口名：com.kylin.processStateManage
  - 方法
    - SetOomScore

子模块	进程资源管理	
接口类型	C	
原型	int SetOomScore(int score)	
描述	设置当前进程的oom分值	
参数	score	进程的oom分值， 分值范围通常是（-1000，1000）
返回值	int	成功返回0
备注	无	

```
- SetOomScoreChild
```

子模块	进程资源管理	
接口类型	dbus接口	
原型	int SetOomScoreChild(int score)	
描述	设置当前进程的子进程的oom分值	
参数	score	进程的oom分值， 分值范围通常是（-1000，1000）
返回值	int	成功返回0
备注	无	

3.7 jpeg编码

该层主要为应用提供jpeg编码器接口，可以实现初始化jpeg编码器，将rgb格式的纯字符内容的图片编码成jpg图片，释放编码器内存。

- **安装命令**

```
$ sudo apt-get install libkysdk-imageproc libkysdk-imageproc-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-imageproc
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKIMAGE kysdk-imageproc)target_include_dir
```

3.7.1 jpeg编码器

- **引入头文件**

```
#include "kysdk/kysdk-system/libkyimageproc.h"
```

- **库文件路径**

`/usr/lib/*/libkyimageproc.so`

'*'代表不同架构的架构目录名称，例如x86_64-linux-gnu

- **子模块信息**

> 初始化jpeg编码器(自2.3.0.0版本启用)

子模块	jpeg编码	
接口类型	C	
原型	void* kdk_jpeg_encoder_init()	
描述	初始化jpeg编码器	
参数	无	
返回值	void*	返回编码器句柄
备注	无	

> jpeg图编码接口(自2.3.0.0版本启用)

子模块	jpeg编码	
接口类型	C	
原型	int kdk_jpeg_encode_to_file(void *handle, char *srcFile, char *destFile)	
描述	jpeg图编码接口	
参数	handle	编码器句柄
	srcFile	源文件
	destFile	输出文件
返回值	0	编码成功
	-1	编码失败
备注	无	

> 释放编码器内存(自2.3.0.0版本启用)

子模块	jpeg编码	
接口类型	C	
原型	void kdk_jpeg_encoder_release(void* handle)	
描述	释放编码器内存	
参数	handle	编码器句柄
返回值	无	
备注	无	

3.8 系统设置

该层主要提供系统设置接口。

- **安装命令**

```
$ sudo apt-get install libkysdk-accounts libkysdk-accounts-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-accounts
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKACCOUNTS kysdk-accounts)target_include_d
```

3.8.1 系统设置功能

- **引入头文件**

```
#include "kysdk/kysdk-system/libkyaccounts.h"
```

- **库文件路径**

`/usr/bin/*/libkysdk-accounts.so`

'*'代表不同架构的架构目录名称，例如x86_64-linux-gnu

- **子模块信息**

> 创建用户，并创建家目录(自2.4.1.0版本启用)

子模块	系统设置	
接口类型	C	
原型	char* kdk_system_create_user(char* name, char* fullName, int accountType)	
描述	创建用户，并创建家目录	
参数	name	用户名
	fullName	用户全名
	accountType	创建的类型 0 标准用户 1 管理员用户
返回值	char*	成功返回用户家目录路径， 失败返回NULL
备注	返回的字符串需要被 free 释放	

> 修改用户密码(自2.4.1.0版本启用)

子模块	系统设置	
接口类型	C	
原型	bool kdk_system_change_password(char* username, char* password, unsigned int *err_num)	
描述	修改用户密码	
入参	username	用户名
	password	新的用户密码
出参	err_num	0 修改密码成功 1 身份验证失败 2 用户不存在

		3 密码不规范 4 参数为空
返回值	bool	是否成功
备注	无	

获取用户是否存在(自2.4.1.0版本启用)

子模块	系统设置	
接口类型	C	
原型	bool kdk_system_check_has_user(char *username)	
描述	获取用户是否存在	
参数	username	用户名
返回值	bool	是否存在
备注	无	

获取加密使用的公钥(自2.4.1.0版本启用)

子模块	系统设置	
接口类型	C	
原型	char* kdk_login_get_public_encrypt()	
描述	获取加密使用的公钥	
参数	无	
返回值	char*	返回非对称加密使用的公钥，加密使用rsa算法
备注	无	

发送用户名和密码进行登录(自2.4.1.0版本启用)

子模块	系统设置	
接口类型	C	
原型	bool kdk_login_send_password(const char* username, unsigned char* password, int length)	
描述	发送用户名和密码进行登录	
参数	username	用户名
	password	加密后的密码
	length	密码长度
返回值	bool	是否成功
备注	无	

3.9 电池信息

该层主要提供获取电池信息接口。

- **安装命令**

```
$ sudo apt-get install libkysdk-battery libkysdk-battery-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-battery
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKBATTERY kysdk-battery)target_include_dir
```

3.9.1 获取电池信息

- **引入头文件**

```
#include "kysdk/kysdk-system/libkybattery.h"
```

- **库文件路径**

`/usr/lib/*/libkybattery.so`

'*'代表不同架构的架构目录名称，例如x86_64-linux-gnu

- **子模块信息**

> 获取当前主机电池剩余电量百分比(自2.4.1.0版本启用)

子模块	电池信息	
接口类型	C	
原型	float kdk_battery_get_soc()	
描述	获取当前主机电池剩余电量百分比	
参数	无	
返回值	float	电池剩余电量百分比
备注	无	

> 获取当前主机电池充电状态(自2.4.1.0版本启用)

子模块	电池信息	
接口类型	C	
原型	BatteryChargeState kdk_battery_get_charge_state()	
描述	获取当前主机电池充电状态	
参数	无	
返回值	BatteryChargeState	电池充电状态枚举变量; 以下是枚举值和描述: NONE = 0 电池充电状态未知 ENABLE = 1 电池充电状态为使能状态(充电) DISABLE = 2 电池充电状态为停止状态(放电) FULL = 3 电池充电状态为已充满状态 NOT_CHARGING = 4 电池充电状态为未充电
备注	无	

> 获取当前主机电池健康状态(自2.4.1.0版本启用)

子模块	电池信息

接口类型	C	
原型	BatteryHealthState kdk_battery_get_health_state()	
描述	获取当前主机电池健康状态	
参数	无	
返回值	BatteryHealthState	电池健康状态枚举变量； 以下是枚举值和描述: UNKNOWN = 0 电池健康状态未知 GOOD = 1 电池健康状态为正常 OVERHEAT = 2 电池健康状态为过热 OVERVOLTAGE = 3 电池健康状态为过压 COLD = 4 电池健康状态为低温 DEAD = 5 电池健康状态为僵死状态 UNSPEC_FAILURE = 6 电池的健康状态为未指定的故障 WATCHDOG_TIMER_EXPIRE = 7 看门狗定时器到期 SAFETY_TIMER_EXPIRE = 8 安全定时器到期 OVERCURRENT = 9 过电流 CALIBRATION_REQUIRED = 10 需要校准 WARM = 11 电池的健康状态为温暖； 正常状态 COOL = 12 电池的健康状态为凉爽； 正常状态
备注	无	

获取当前主机连接的充电器类型(自2.4.1.0版本启用)

子模块	电池信息	
接口类型	C	
原型	BatteryPluggedType kdk_battery_get_plugged_type()	
描述	获取当前主机连接的充电器类型	
参数	无	
返回值	BatteryPluggedType	充电器类型枚举变量，以下是枚举值和描述: TYPE_NONE = 0 连接充电器类型未知 BATTERY = 1 连接的充电器类型为电池 UPS = 2 连接的充电器类型为不间断电源 MAINS = 3 连接的充电器类型为交流电源 USB = 4 连接的充电器类型为USB USB_DCP = 5 连接的充电器类型为专用于充电的USB端口， 提供更大的电流 USB_CDP = 6 通过USB接口连接的可用于下游设备充电的端口 USB_ACA = 7 USB附件充电适配器 USB_TYPE_C = 8 USB Type-C接口 USB_PD = 9 USB Type-C接口快充 USB_PD_DRP = 10 USB Type- C接口的工作模式， 可以充当主机或设备的角色，具有双重作用

		APPLE_BRICK_ID = 11 苹果充电方式 WIRELESS = 12 连接的充电器类型为无线充电器
备注	无	

获取当前主机电池电压(自2.4.1.0版本启用)

子模块	电池信息	
接口类型	C	
原型	float kdk_battery_get_voltage()	
描述	获取当前主机电池电压	
参数	无	
返回值	float	电池电压,单位伏
备注	无	

获取当前主机电池技术型号(自2.4.1.0版本启用)

子模块	电池信息	
接口类型	C	
原型	char* kdk_battery_get_technology()	
描述	获取当前主机电池技术型号	
参数	无	
返回值	char*	成功返回电池技术型号， 失败返回NULL
备注	无	

获取当前主机电池温度(自2.4.1.0版本启用)

子模块	电池信息	
接口类型	C	
原型	float kdk_battery_get_temperature()	
描述	获取当前主机电池温度	
参数	无	
返回值	float	电池温度,单位摄氏度
备注	无	

检查当前主机是否支持电池(自2.4.1.0版本启用)

子模块	电池信息	
接口类型	C	
原型	bool kdk_battery_is_present()	
描述	检查当前主机是否支持电池	
参数	无	
返回值	bool	是否支持
备注	无	

获取当前主机电池电量等级(自2.4.1.0版本启用)

子模块	电池信息
接口类型	C

原型	int kdk_battery_get_capacity_level()	
描述	获取当前主机电池电量等级	
参数	无	
返回值	int	电池电量等级
备注	无	

3.10 存储模块

获取存储模块信息

- **安装命令**

```
$ sudo apt install libkysdk-storage libkysdk-storage-dev
```

- **构建示例**

  i. pro文件构建示例

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-systime
```

  ii. CMakeLists文件构建示例

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKSYSTIME kysdk-systime)target_incl
```

3.10.1 获取存储信息

- **引用头文件**

```
#include /usr/include/kysdk/kysdk-base/libkystorage.h
```

- **库文件路径**

`/usr/lib/*/libkystorage.so`

'*'代表不同架构的架构目录名称，例如x86_64-linux-gnu

- **子模块信息**

获取默认浏览器 Cookie 的路径(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_get_default_cookie_path()	
描述	获取默认浏览器 Cookie 的路径	
参数	无	
返回值	char *	浏览器 Cookie 的路径
备注	返回值需要释放 目前仅支持qax，chrome，chromium和firefox, 默认浏览器不是这四款会返回NULL	

获取当前用户桌面文件夹的路径(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_get_default_desktop_path()	
描述	获取当前用户桌面文件夹的路径	
参数	无	
返回值	char *	当前用户桌面文件夹的路径
备注	返回值需要释放	

获取当前用户文档文件夹的路径(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_get_default_documents_path()	
描述	获取当前用户文档文件夹的路径	
参数	无	
返回值	char *	当前用户文档文件夹的路径
备注	返回值需要释放	

获取当前用户指定浏览器的Internet 缓存文件夹的路径(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_get_default_internetCache_path(char *name)	
描述	获取当前用户指定浏览器的Internet 缓存文件夹的路径	
参数	name	浏览器名 qax chrome chromium firefox
返回值	char *	浏览器的Internet 缓存文件夹的路径
备注	返回值需要释放 目前仅支持qax，chrome，chromium和firefox	

获取当前用户下载文件夹的路径(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_get_default_download_path()	
描述	获取当前用户下载文件夹的路径	
参数	无	
返回值	char *	当前用户下载文件夹的路径
备注	返回值需要释放	

获取文件内容的 MIME 类型(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_get_file_ContentType(char *filename)	
描述	获取文件内容的 MIME 类型	
参数	filename	文件路径
返回值	char *	文件的 MIME 类型
备注	返回值需要释放	

获取文件的最近改动的日期和时间(自2.4.1.0版本启用)

子模块	存储模块

接口类型	C	
原型	char *kdk_storage_get_file_dateChange(char *filename)	
描述	获取文件的最近改动的日期和时间	
参数	filename	文件路径
返回值	char *	文件的最近改动的日期和时间
备注	返回值需要释放	

获取文件最近更改的日期和时间(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_get_file_dateModify(char *filename)	
描述	获取文件最近更改的日期和时间	
参数	filename	文件路径
返回值	char *	文件最近更改的日期和时间
备注	返回值需要释放	

获取文件最近访问的日期和时间(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_get_file_dateAccess(char *filename)	
描述	获取文件最近访问的日期和时间	
参数	filename	文件路径
返回值	char *	文件最近访问的日期和时间
备注	返回值需要释放	

获取文件大小(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	long kdk_storage_get_file_size(char *filename)	
描述	获取文件大小	
参数	filename	文件路径
返回值	long	文件大小
	0	失败
备注	无	

读取文件内容(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	int kdk_storage_read_file(const char *filename, char *buffer, size_t size)	
描述	读取文件内容到缓冲区中	
参数	filename	文件路径
	buffer	缓冲区

	size	缓冲区大小
返回值	int	成功返回写入数据的长度
	-1	失败
备注	无	

将数据追加到指定的文件中(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	int kdk_storage_write_file(const char *filename, const char *data, size_t length)	
描述	将数据追加到指定的文件中	
参数	filename	文件路径
	data	数据
	length	数据长度
返回值	int	成功返回写入数据的长度
	-1	失败
备注	无	

移动文件(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	int kdk_storage_move_file(const char *source, const char *destination)	
描述	移动文件	
参数	source	源文件路径
	destination	目标文件路径
返回值	0	成功
	-1	失败
备注	无	

获取文件权限(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_get_file_permissions(const char *filename)	
描述	获取文件权限	
参数	filename	文件的路径
返回值	char *	文件权限
备注	返回值需要释放	

获取文件所有者(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_get_file_owner(const char *filename)	

描述	获取文件所有者	
参数	filename	文件的路径
返回值	char *	文件所有者
备注	返回值需要释放	

获取文件所属组(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_get_file_group(const char *filename)	
描述	获取文件所属组	
参数	filename	文件的路径
返回值	char *	文件所属组
备注	返回值需要释放	

列出指定路径下的所有文件和文件夹(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	FileInfo *kdk_storage_list_files(const char *path)	
描述	列出指定路径下的所有文件和文件夹	
参数	path	要列出的路径
返回值	FileInfo *	结构体链表， 结构体定义见后文
备注	返回值需要释放,可通过sdk提供的接口释放返回值	

结构体

```
typedef struct _FileInfo{char name[256];int is_dir;struct _FileInfo *next;}FileInfo;
```

结构体释放函数

子模块	存储模块	
接口类型	C	
原型	void kdk_storage_free_file_info(FileInfo *info)	
描述	释放FileInfo链表	
参数	info	FileInfo链表
返回值	无	
备注	无	

解析指定文件的符号链接（软链接），获取实际文件路径(自2.4.1.0版本启用)

子模块	存储模块	
接口类型	C	
原型	char *kdk_storage_resolve_symbolic_link(const char *filename)	
描述	解析指定文件的符号链接（软链接），获取实际文件路径	
参数	filename	指定的文件名
返回值	char*	实际文件路径

备注	返回值需要释放

4 应用支撑 SDK

kysdk 应用支撑层中，所有的包均为向图形化应用提供图形化开发功能，例如图形化控件、主题风格等；该层通常与某个特定的开发框架有所绑定 ， 如 QT 、 GTK ； 应 用 支 撑层 SDK 安装时需安装 两个虚包：libkysdk-application、libkysdk-applications-dev，其他包按照功能分类生成多个实包，比如：QT 控件类： libkysdk-qtwidgets，wayland 显示协议兼容接口 libkysdk-waylandhelper，应用通用模块 libkysdk-kabase 等。

安装命令：

```
$ sudo apt install libkysdk-applications libkysdk-applications-dev
```

4.1 QT 扩展控件

QT 扩展控件 kysdk-qtwidgets 属于 kysdk-application 的子模块，安装方式如下：

```
sudo apt install libkysdk-qtwidgets libkysdk-qtwidgets-dev
```

根据不同项目类型，可参考以下 demo：

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-qtwidgets
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(Qt5 COMPONENTS Widgets REQUIRED)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKQTWIDGETS_PKG kysdk-qtwidgets)
target_include_directories(demo PRIVATE ${KYSDKQTWIDGETS_PKG_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKQTWIDGETS_PKG_LIBRARY_DIRS})
target_link_libraries(demo Qt5::Widgets ${KYSDKQTWIDGETS_PKG_LIBRARIES})
```

在具体项目中，需在代码中引入对应的头文件以及命名空间，如：

```
#include "kwidget.h"
using namespace kdk;
```

部分控件用到翻译文件，可以在 main()函数中加载翻译文件，目前支持中

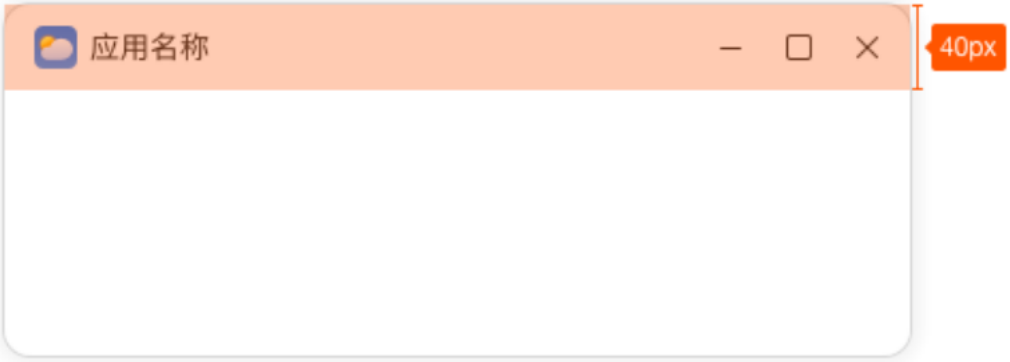
文、藏文、英文三种语言，翻译文件已经编进动态库中，加载方式如下：

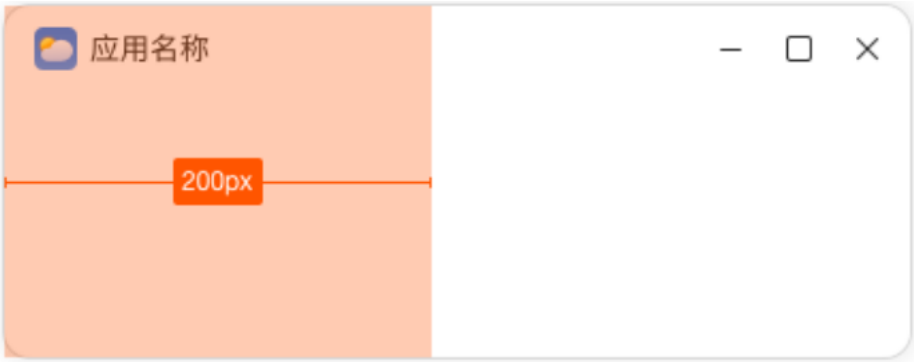
```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QTranslator trans;
    QString locale = QLocale::system().name();
    if(locale == "zh_CN")
    {
        if(trans.load(":/translations/gui_zh_CN.qm"))
        {
            a.installTranslator(&trans);
        }
    }
    if(locale == "bo_CN")
    {
        if(trans.load(":/translations/gui_bo_CN.qm"))
        {
            a.installTranslator(&trans);
        }
    }
    Widget w;
    w.show();
    return a.exec();
}
```

4.1.1 窗体模块

4.1.1.1 基础窗体

功能描述：KWidget，继承自 QWidget,支持响应主题背景切换，响应图标主题切换，标题颜色响应窗口激活状态，窗口按钮样式符合 ukui3.1 的设计风格，分为四个组成部分iconBar,windowButtonBar,sideBar,baseBar
iconBar,windowButtonBar 默认高度为 40px，sideBar 默认宽度为 200px。





枚举类型	enum LayoutType{VerticalType, HorizontalType, MixedType };

子模块	libkysdk-qtwidgets
接口类型	C++
原型	QWidget* sideBar();

描述	获取左边栏widget,通过setlayout添加自定义内容	
参数	参数名称	参数说明
	无	无
返回值	QWidget*	返回侧边widget
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QWidget* baseBar();	
描述	获取主内容区widget,通过setlayout添加自定义内容。	
参数	参数名称	参数说明
	无	无
返回值	QWidget*	返回主内容区widget
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIcon(const QIcon& icon);	
描述	通过icon添加窗体图标	
参数	参数名称	参数说明
	QIcon	传入图片
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIcon(const QString& iconName);	
描述	设置窗体图标，iconName 需要直接指定系统目录中的图标名称，如"kylin-music"	
参数	参数名称	参数说明
	QString	传入图片名称
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setWidgetName(const QString& widgetName);	
描述	设置窗体名称。	
参数	参数名称	参数说明
	QString	传入需要设置的名称
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets

接口类型	C++	
原型	KWindowButtonBar* windowButtonBar();	
描述	获取窗口三联组合控件，以控制是否显示最大化、 最小化按钮和下拉菜单按钮。	
参数	参数名称	参数说明
	无	无
返回值	KWindowButtonBar*	返回界面右上角的windowbuttonbar
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KIconBar* iconBar();	
描述	获取窗口标题、图标组合控件，以控制相关样式。	
参数	参数名称	参数说明
	无	无
返回值	KIconBar*	返回界面左上角的iconbar
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setLayoutType(LayoutType type);	
描述	设置布局结构类型。	
参数	参数名称	参数说明
	LayoutType	设置窗体的布局结构， 见上方枚举值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setWindowFlags(Qt::WindowFlags type);	
描述	设置窗口标志，1.2版本添加	
参数	参数名称	参数说明
	Qt::WindowType	Qt::WindowFlags枚举值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setWindowFlag(Qt::WindowType flag,bool on =true);	
描述	设置窗口标志，1.2版本添加	
参数	参数名称	参数说明
	Qt::WindowType	Qt::WindowType枚举值
	布尔值	确定是否启用设置的窗口标志

返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setSidebarFollowMode(bool flag);	
描述	设置sidebar是否遵循系统更改宽度，1.2版本添加	
参数	参数名称	参数说明
	布尔值	true遵循默认策略， false遵循自主策略
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool sidebarFollowMode();	
描述	获取sidebar是否遵循系统更改宽度，1.2版本添加	
参数	参数名称	参数说明
	无	无
返回值	布尔值	返回kwidget是否跟随默认策略更改sidebar宽度
备注	无	

4.1.1.2 KBubbleWidget

> 功能描述：KBubbleWidget，继承自 QWidget，是一个带有气泡尾部的窗体，可以指定气泡尾部的尺寸、显示方向和显示位置，还可以设置窗体的圆角、毛玻璃以及透明度。自 2.0.0.0 版本启用。

枚举类型	enum TailDirection{ TopDirection, LeftDirection, BottomDirection, RightDirection, None };
	enum TailLocation{ LeftLocation, MiddleLocation, RightLocation };

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTailSize(const QSize& size);	
描述	设置气泡尾部尺寸	
参数	参数名称	参数说明
	QSize	尺寸大小
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++
原型	QSize tailSize();

描述	获取气泡尾部尺寸	
参数	参数名称	参数说明
	无	无
返回值	QSize	返回气泡尾部的尺寸
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTailPosition(TailDirection dirType, TailLocation locType=MiddleLocation);	
描述	设置气泡尾部显示位置	
参数	参数名称	参数说明
	TailDirection	气泡显示的方向，上下左右
	TailLocation	气泡显示的位置，左 中 右
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	TailDirection tailDirection();	
描述	获取气泡尾部显示方向（左、上、右、下	
参数	参数名称	参数说明
	无	无
返回值	TailDirection	返回气泡的显示方向
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	TailLocation tailLocation();	
描述	获取气泡尾部显示位置（居左、居中、居右）	
参数	参数名称	参数说明
	无	无
返回值	TailLocation	返回气泡的显示位置
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius(int bottomLeft,int topLeft,int topRight,int bottomRight);	
描述	设置窗体圆角半径	
参数	参数名称	参数说明
	int	左下角半径
	int	左上角半径
	int	右上角半径

	int	右下角半径
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius(int radius);	
描述	设置窗体圆角半径	
参数	参数名称	参数说明
	int	窗体四个圆角半径
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setEnableBlur(bool flag);	
描述	设置是否启用毛玻璃效果	
参数	参数名称	参数说明
	布尔值	true启用毛玻璃， false不启用毛玻璃
返回值	五	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool enableBlur();	
描述	获取是否已启用毛玻璃效果	
参数	参数名称	参数说明
	无	无
返回值	布尔值	ture 启用毛玻璃，false 未启用毛玻璃
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setOpacity(qreal opacity);	
描述	设置透明度	
参数	参数名称	参数说明
	qreal	透明度值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++
原型	qreal opacity();

描述	获取透明度	
参数	参数名称	参数说明
	无	无
返回值	qreal	透明度值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setHighLightBackground(bool flag)	
描述	设置是否启用高亮背景色（自2.4启用）	
参数	参数名称	参数说明
	布尔值	true 启用高亮背景色，false 不启用高亮背景色
返回值	无	无
备注	无	

4.1.2 对话框模块

4.1.2.1 基础对话框

> 功能描述：KDialog，继承自 QDialog 支持响应主题背景切换，相应图标主题切换，窗口按钮样式符合 ukui3.1 的设计风格，标题颜色响应窗口激活状态。

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setWindowIcon(const QIcon &icon);	
描述	设置对话框图标	
参数	参数名称	参数说明
	QIcon	设置图标QIcon
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setWindowIcon(const QString& iconName);	
描述	直接根据图标名称设置窗口图标，例如：dialog->setWindowIcon("kylin-music");	
参数	参数名称	参数说明
	QString	需要设置的图标名称
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setWindowTitle(const QString &);	
描述	设置对话框标题名称	
参数	参数名称	参数说明

	QString	标题名称
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPushButton* maximumButton();	
描述	获取最大化按钮	
参数	参数名称	参数说明
	无	无
返回值	QPushButton*	最大化按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPushButton* minimumButton();	
描述	获取最小化按钮	
参数	参数名称	参数说明
	无	无
返回值	QPushButton*	最小化按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPushButton* closeButton();	
描述	获取关闭按钮	
参数	参数名称	参数说明
	无	无
返回值	QPushButton*	关闭按钮
备注	无	

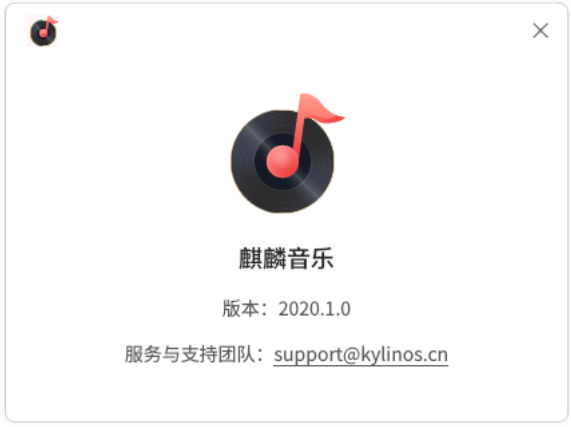
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KMenuButton* menuButton();	
描述	获取下拉菜单按钮，默认是隐藏的,不显示。	
参数	参数名称	参数说明
	无	无
返回值	KMenuButton*	菜单按钮
备注	无	

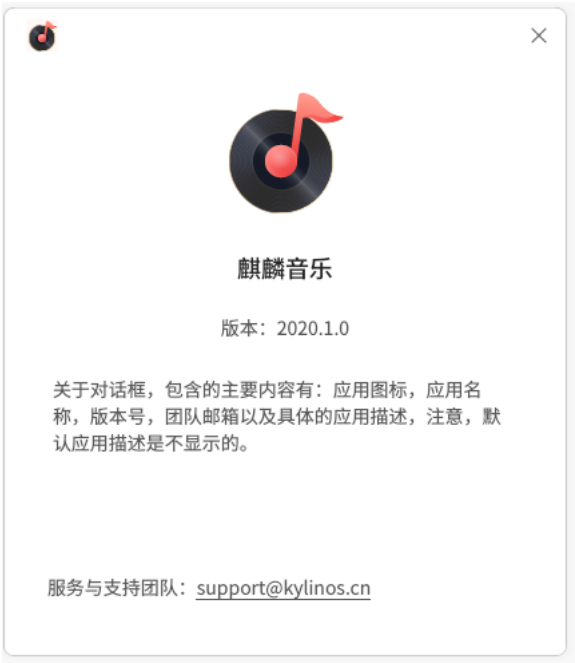
子模块	libkysdk-qtwidgets
接口类型	C++
原型	QWidget* mainWidget();
描述	获取主内容区，通过setLayout()添加内容。例如：dialog-

	>mainWidget()->setLayout(hLayout);	
参数	参数名称	参数说明
	无	无
返回值	QWidget*	获取主内容区widget
备注	无	

4.1.2.2 关于对话框

功能描述：KAboutDialog，包含的主要内容有：应用图标，应用名称，版本号，团队邮箱以及具体的应用描述，注意，默认应用描述是不显示的。可以通过 setBodyTextVisible(bool)控制其是否需要显示。





子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QWidget* sideBar();	
描述	获取左边栏widget	
参数	参数名称	参数说明
	无	无
返回值	QWidget*	返回侧边widget
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++

原型	void setAppIcon(const QIcon& icon);	
描述	设置应用程序图标	
参数	参数名称	参数说明
	QIcon	传入icon
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setAppName(const QString& appName);	
描述	获取应用程序名称	
参数	参数名称	参数说明
	QString	传入图片名称
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString appName();	
描述	获取应用程序名称	
参数	参数名称	参数说明
	无	无
返回值	QString	图标名称
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setAppVersion(const QString& appVersion);	
描述	设置应用程序版本号	
参数	参数名称	参数说明
	QString	版本号信息
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString appVersion();	
描述	获取应用程序版本号	
参数	参数名称	参数说明
	无	无
返回值	QString	版本号信息
备注	无	

子模块	libkysdk-qtwidgets

接口类型	C++	
原型	void setBodyText(const QString& bodyText);	
描述	设置应用程序具体的说明内容	
参数	参数名称	参数说明
	QString	应用程序的说明
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString bodyText();	
描述	获取应用程序具体的说明内容	
参数	参数名称	参数说明
	无	无
返回值	QString	应用程序的说明
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setAppSupport(const QString& appSupport);	
描述	设置服务与支持邮箱，有默认缺省	
参数	参数名称	参数说明
	QString	服务与支持邮箱
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString appSupport();	
描述	获取服务与支持邮箱	
参数	参数名称	参数说明
	无	无
返回值	QString	服务与支持邮箱
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBodyTextVisiable(bool flag);	
描述	设置是否显示说明内容	
参数	参数名称	参数说明
	bool	true 显示说明内容，false 隐藏说明内容
返回值	QWidget	返回侧边widget
备注	无	

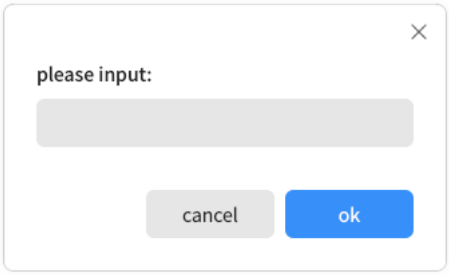
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setAppPrivacyLabelVisible(bool flag);	
描述	设置隐私按钮是否可见（自1.2.0.10版本启用）	
参数	参数名称	参数说明
	布尔值	true 隐私按钮可见，false 隐私按钮不可见
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool AppPrivacyLabelIsVisible();	
描述	获取隐私按钮是否可见（自1.2.0.10版本启用）	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 隐私按钮可见，false 隐私按钮不可见
备注	无	

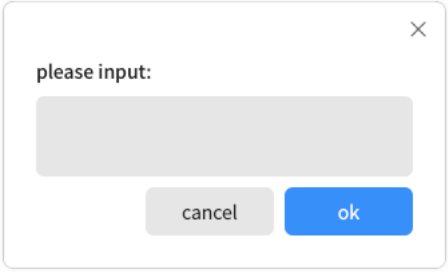
4.1.2.3 输入对话框

> 功能描述：输入对话框 KInputDialog，继承自 QDialog，参考 QInputDialog源码，对子控件布局以及样式进行了调整，功能同 QInputDialog。

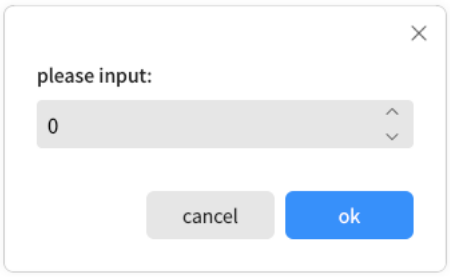
(1) QString KInputDialog::getText(QWidget *parent,const QString &label,QLineEdit::EchoMode mode, const QString &text,bool *ok,Qt::WindowFlags flags, Qt::InputMethodHints inputMethodHints)
文本输入对话框，同 QInputDialog



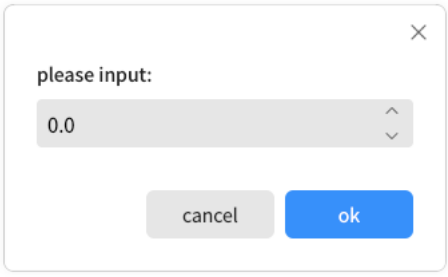
(2) QString KInputDialog::getMultiLineText(QWidget *parent,constQString&label, const QString &text, bool *ok, Qt::WindowFlags flags,Qt::InputMethodHints inputMethodHints)
多行文本输入框，同 QInputDialog



(3) int KInputDialog::getInt(QWidget *parent,const QString &label, int value,int min, int max, int step, bool *ok, Qt::WindowFlags flags)
整型数字输入对话框，同 QInputDialog



(4) double KInputDialog::getDouble(QWidget *parent,const QString &label,double value, double minValue, double maxValue, int decimals, bool *ok, Qt::WindowFlags flags)
浮点型数字输入对话框，同 QInputDialog



枚举类型

枚举类型	enum InputDialogOption { NoButtons, UseListViewForComboBoxItems, UsePlainTextEditForTextInput };
	enum InputMode { TextInput, IntInput, DoubleInput };

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setInputMode(InputMode mode);	
描述	设置输入模式	
参数	参数名称	参数说明
	InputMode	输入模式
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setLabelText(const QString &text);	
描述	设置标签文本	
参数	参数名称	参数说明
	QString	设置文本
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++
原型	QString labelText() const;

描述	获取标签文本	
参数	参数名称	参数说明
	无	无
返回值	QString	返回标签文本
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setOption(InputDialogOption option, bool on = true);	
描述	设置选项	
参数	参数名称	参数说明
	InputDialogOption	输入类型
	布尔值	true 选强启用，false 选项禁用
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool testOption(InputDialogOption option) const;	
描述	测试选项	
参数	参数名称	参数说明
	布尔值	true 选强启用，false 选项未启用
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setOptions(InputDialogOptions options);	
描述	设置选项	
参数	参数名称	参数说明
	InputDialogOptions	可传入多个选项
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	InputDialogOptions options() const;	
描述	获取输入框选项	
参数	参数名称	参数说明
	InputDialogOptions	获取选项模式
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTextValue(const QString &text);	
描述	设置文本值	
参数	参数名称	参数说明
	QString	设置文本值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString textValue() const;	
描述	获取文本值	
参数	参数名称	参数说明
	无	无
返回值	QString	文本值内容
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTextEchoMode(QLineEdit::EchoMode mode);	
描述	设置文本模式	
参数	参数名称	参数说明
	QLineEdit::EchoMode	文本模式
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QLineEdit::EchoMode textEchoMode() const;	
描述	获取文本模式	
参数	参数名称	参数说明
	无	无
返回值	QLineEdit::EchoMode	文本模式
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setComboBoxEditable(bool editable);	
描述	设置组合框可编辑	
参数	参数名称	参数说明
	布尔值	true 可编辑，false 不可编辑
返回值	无	无

备注	无

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isComboBoxEditable() const;	
描述	返回组合框是否可以编辑	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 可编辑，false 不可编辑
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setComboBoxItems(const QStringList &items);	
描述	设置组合框项目	
参数	参数名称	参数说明
	QStringList	组合框项目列表
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QStringList comboBoxItems() const;	
描述	返回组合框列表	
参数	参数名称	参数说明
	无	无
返回值	QStringList	组合框项目列表
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIntValue(int value);	
描述	设置int类型的值	
参数	参数名称	参数说明
	int	值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int intValue() const;	
描述	获取int类型的值	
参数	参数名称	参数说明
	无	无

返回值	int	int类型的值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIntMinimum(int min);	
描述	设置int类型的最低值	
参数	参数名称	参数说明
	min	int类型最小的值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int intMinimum() const;	
描述	获取int最低值	
参数	参数名称	参数说明
	无	无
返回值	int	获取int类型的最小值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIntMaximum(int max);	
描述	设置int类型最大值	
参数	参数名称	参数说明
	max	int类型最大值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int intMaximum() const;	
描述	获取int最大值	
参数	参数名称	参数说明
	无	无
返回值	int	int类型最大值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIntRange(int min, int max);	
描述	设置int范围	
参数	参数名称	参数说明

	min	int类型最小值
	max	int类型最大值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIntStep(int step);	
描述	设置int步数	
参数	参数名称	参数说明
	step	int类型步长
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int intStep() const;	
描述	获取int步数	
参数	参数名称	参数说明
	无	无
返回值	int	获取int类型步长
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setDoubleValue(double value);	
描述	设置double值	
参数	参数名称	参数说明
	value	设置double类型值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	double doubleValue() const;	
描述	获取double值	
参数	参数名称	参数说明
	无	无
返回值	double	获取double类型值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setDoubleMinimum(double min);	

描述	设置double最小值	
参数	参数名称	参数说明
	min	设置double类型最小值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	double doubleMinimum() const;	
描述	获取double最小值	
参数	参数名称	参数说明
	无	无
返回值	double	获取double类型最小值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setDoubleMaximum(double max);	
描述	设置double最大值	
参数	参数名称	参数说明
	max	设置double类型最大值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	double doubleMaximum() const;	
描述	获取double最大值	
参数	参数名称	参数说明
	无	无
返回值	double	获取double类型最大值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setDoubleRange(double min, double max);	
描述	设置double范围	
参数	参数名称	参数说明
	min	double范围最小值
	max	double范围最大值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	

接口类型	C++	
原型	void setDoubleDecimals(int decimals);	
描述	设置两个小数	
参数	参数名称	参数说明
	decimal	小数位数
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int doubleDecimals() const;	
描述	返回小数位数	
参数	参数名称	参数说明
	无	无
返回值	int	小数位数
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setOkButtonText(const QString &text);	
描述	设置确认按钮文本	
参数	参数名称	参数说明
	text	确认按钮的文本内容
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString okButtonText() const;	
描述	获取确认按钮文本	
参数	参数名称	参数说明
	无	无
返回值	QString	确认按钮的文本
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setCancelButtonText(const QString &text);	
描述	设置取消按钮文本	
参数	参数名称	参数说明
	text	取消按钮的文本内容
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString cancelButtonText() const;	
描述	获取取消按钮文本	
参数	参数名称	参数说明
	无	无
返回值	QString	取消按钮的文本内容
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void open(QObject *receiver, const char *member);	
描述	此函数将其信号之一连接到接收器和成员指定的插槽。 特定信号取决于在成员中指定的参数。	
参数	参数名称	参数说明
	receiver	信号
	member	获取信号执行操作
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setVisible(bool visible) override;	
描述	设置是否可见	
参数	参数名称	参数说明
	布尔值	true 可见，false 不可见
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QSize minimumSizeHint() const override;	
描述	最小尺寸提示	
参数	参数名称	参数说明
	无	无
返回值	QSize	最小尺寸
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QSize sizeHint() const override;	
描述	尺寸提示	
参数	参数名称	参数说明
	无	无

返回值	QSize	尺寸大小
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	static QString getText(QWidget *parent,const QString &label,QLineEdit::EchoMode echo = QLineEdit::Normal,const QString &text = String(), bool *ok = nullptr,Qt::WindowFlags flags = Qt::WindowFlags(),Qt::InputMethodHints inputMethodHints = Qt::ImhNone);	
描述	获取文本	
参数	参数名称	参数说明
	parent	父类
	label	显示给用户的文本
	echo	行编辑器将使用的回声模式
	text	行编辑中的默认文本
	ok	ok为非空*如果用户按下ok ok将被设置为true， 如果用户按下Cancel ok将被设置为false
	flags	Qt::WindowFlags选项
	inputMethodHints	输入法提示
返回值		文本内容
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	static QString getMultiLineText(QWidget *parent,const QString &label,const QString &text = QString(), bool *ok = nullptr,Qt::WindowFlags flags = Qt::WindowFlags(),Qt::InputMethodHints inputMethodHints = Qt::ImhNone);	
描述	获取多行文本	
参数	参数名称	参数说明
	parent	父类
	label	显示给用户的文本
	text	行编辑中的默认文本
	ok	ok为非空*如果用户按下ok ok将被设置为true， 如果用户按下Cancel ok将被设置为false
	flags	Qt::WindowFlags选项
	inputMethodHints	输入法提示
返回值	QString	文本内容
备注	无	

子模块	libkysdk-qtwidgets

接口类型	C++	
原型	static QString getItem(QWidget *parent,const QString &label,const QStringList &items, int current = 0, bool editable = true,bool *ok = nullptr, Qt::WindowFlags flags = Qt::WindowFlags(),Qt::InputMethodHints inputMethodHints = Qt::ImhNone);	
描述	获取项目	
参数	参数名称	参数说明
	parent	父类
	label	显示给用户的文本
	items	插入到组合框中的字符串列表
	current	当前项的编号
	editable	edit为true， 用户可以输入自己的文本; 否则， 用户可能只选择现有项目中的一个
	ok	ok为非空*如果用户按下ok ok将被设置为true， 如果用户按下Cancel ok将被设置为false
	flags	Qt::WindowFlags选项
	inputMethodHints	输入法提示
返回值	QString	返回选中的项目文本
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	static int getInt(QWidget *parent,const QString &label, int value = 0,int minValue = -2147483647, int maxValue = 2147483647,int step = 1, bool *ok = nullptr, Qt::WindowFlags flags = Qt::WindowFlags()	
描述	获取int型文本	
参数	参数名称	参数说明
	parent	父类
	label	显示给用户的文本
	value	spinbox将被设置为的默认整数
	minValue	选择的最小值
	maxValue	选择的最大值
	step	按箭头按钮增加或减少值时值的变化量
	ok	ok为非空*如果用户按下ok ok将被设置为true， 如果用户按下Cancel ok将被设置为false
	flags	Qt::WindowFlags选项
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets

接口类型	C++	
原型	static double getDouble(QWidget *parent,const QString &label,double value = 0, double minValue = -2147483647,double maxValue = 2147483647, int decimals = 1, bool *ok = nullptr,Qt::WindowFlags flags = Qt::WindowFlags());	
描述	获取double类型文本	
参数	参数名称	参数说明
	parent	父类
	label	显示给用户的文本
	value	编辑将设置为的默认浮点数
	minValue	选择的最小值
	maxValue	选择的最大值
	decimals	数字可能具有的最大小数位数
	ok	ok为非空*如果用户按下ok ok将被设置为true， 如果用户按下Cancel ok将被设置为false
	flags	Qt::WindowFlags选项
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setDoubleStep(double step);	
描述	设置double步数	
参数	参数名称	参数说明
	double	double类型步数
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	double doubleStep() const;	
描述	获取double步数	
参数	参数名称	参数说明
	无	无
返回值	double	double值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setPlaceholderText(const QString &);	
描述	设置PlaceholderText的文本内容（自1.2.0.12启用）	
参数	参数名称	参数说明

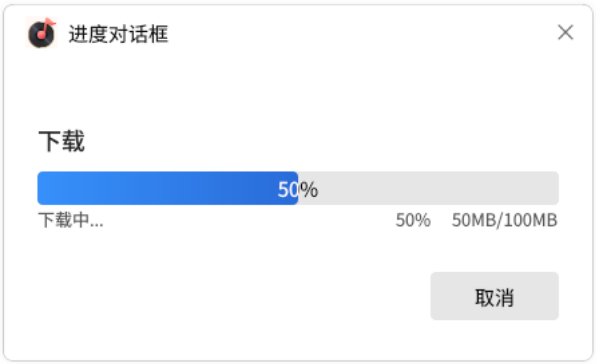
	QString	需要设置的文本
返回值	无	无
备注	无	

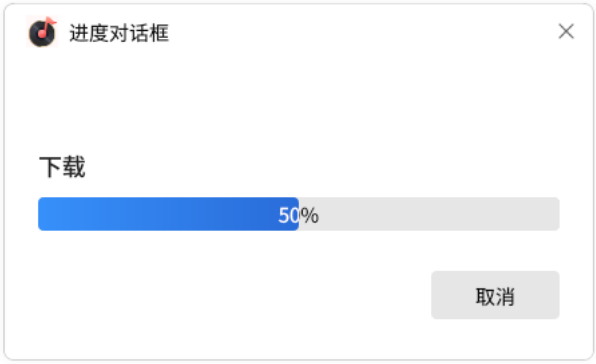
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString placeholderText() const;	
描述	获取PlaceholderText的文本内容（自1.2.0.12启用）	
参数	参数名称	参数说明
	无	无
返回值	QString	获取文本内容
备注	无	

4.1.2.4 进度对话框

> 功能描述：KProgressDialog，继承自 QDialog，参考 QProgressDialog 源码，对子控件 ProgressBar 的样式进行了调整。可以设置具体下载信息是否需要显示，设置进度值的后缀等。

```
KProgressDialog *progress2 = new KProgressDialog(tr("下载"),tr("取消"),0,100,this);
progress2->setSubContent("下载中...");
progress2->setSuffix("MB");
progress2->setWindowTitle("进度对话框");
progress2->setWindowIcon("kylin-music");
progress2->setValue(50);
progress2->setShowDetail(false);
```





子模块	libkysdk-qtwidgets
接口类型	C++
原型	void setLabel(QLabel *label);
描述	将标签设置给label。进度对话框会调整大小以适应。 标签成为进度对话框的所有权，必要时将被删除， 所以不要在堆栈中传递对象的地址。

参数	参数名称	参数说明
	QLabel*	标签设置给label
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setCancelButton(QPushButton *button);	
描述	将取消按钮设置给按钮，cancelButton。 进度对话框拥有这个按钮的所有权， 这个按钮在必要时将被删除， 所以不要传递堆栈中的对象的地址，用new()来创建按钮。 如果传递nullptr，将不会显示取消按钮。	
参数	参数名称	参数说明
	QPushButton*	将取消按钮设置给按钮
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBar(QProgressBar *bar);	
描述	将进度条部件设置给bar。进度对话框会调整大小以适应。 进度条对话框拥有进度条的所有权， 该进度条将在必要时被删除， 所以不要使用分配在堆栈中的进度条。	
参数	参数名称	参数说明
	QProgressBar*	进度条部件设置给bar
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setSuffix(const QString& suffix);	
描述	设置detail的后缀	
参数	参数名称	参数说明
	suffix	detail的后缀
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setShowDetail(bool flag);	
描述	设置是否显示详细信息	
参数	参数名称	参数说明
	flag	true 显示信息，false 隐藏信息

返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int minimum() const;	
描述	返回最小值	
参数	参数名称	参数说明
	无	无
返回值	int	最小值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int maximum() const;	
描述	返回最大值	
参数	参数名称	参数说明
	无	无
返回值	int	最大值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int value() const	
描述	返回当前值	
参数	参数名称	参数说明
	无	无
返回值	int	当前值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString labelText() const;	
描述	返回提示内容	
参数	参数名称	参数说明
	无	无
返回值	QString	返回提示内容
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setAutoReset(bool reset);	
描述	设置进度对话框是否在value()等于maximum() 时立即调用reset()，默认为true。	

参数	参数名称	参数说明
	布尔值	true 调用reset，false 不调用reset
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool autoReset() const;	
描述	获取是否自动重置	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 自动重置，false 不自动重置
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setAutoClose(bool close);	
描述	设置是否自动关闭对话框	
参数	参数名称	参数说明
	布尔值	true 自动关闭对话框，false 不自动关闭对话框
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool autoClose() const;	
描述	获取是否自动关闭对话框	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 自动关闭对话框，false 不自动关闭对话框
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QProgressBar* progressBar();	
描述	获取进度条	
参数	参数名称	参数说明
	无	无
返回值	QProgressBar*	进度条控件
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void cancel()	
描述	取消进度条。	
参数	参数名称	参数说明
	无	无
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void reset()	
描述	重置进度条。	
参数	参数名称	参数说明
	无	无
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setMaximum(int maximum)	
描述	设置进度条所代表的最高值，默认值是100。	
参数	参数名称	参数说明
	maximum	进度条最大值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setMinimum(int minimum)	
描述	设置进度条所代表的最小值，默认值是0。	
参数	参数名称	参数说明
	minimum	进度条最小值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setRange(int minimum, int maximum)	
描述	设置进度条范围，如果最大值小于最小值， 则最小值成为唯一的合法值。如果当前值超出了新的范围， 则用reset()重置进度框。	
参数	参数名称	参数说明
	minimum	进度条最小值

	maximum	进度条最大值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setValue(int progress)	
描述	设置当前进度值。	
参数	参数名称	参数说明
	progress	当前进度条的值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setLabelText(const QString &text)	
描述	设置提示标签文本。	
参数	参数名称	参数说明
	text	提示文本
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setCancelButtonText(const QString &text)	
描述	设置取消按钮文本。	
参数	参数名称	参数说明
	text	取消按钮文本
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setSubContent(const QString &text)	
描述	设置次级内容。	
参数	参数名称	参数说明
	text	取消按钮文本
返回值	无	无
备注	无	

4.1.2.5 程序卸载对话框

> 功能描述：KUninstallDialog，代码整合自麒麟安装器，支持显示应用图标，应用名称，包名，版本号等信息，不包括具体的卸载行为。使用时只需传入包名以及版本后两个参数即可。

```
KUninstallDialog *uninstallDialog = new KUninstallDialog("browser360-cn-stable","104",this);
```

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QLabel* debAppNameLabel();	
描述	获取应用名称的label	
参数	参数名称	参数说明
	无	无
返回值	QLabel*	应用名称的label
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QLabel* debNameLabel();	
描述	获取包名的label	
参数	参数名称	参数说明
	无	无
返回值	QLabel*	包名的label
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QLabel* debIconLabel();	
描述	获取应用图标的label	
参数	参数名称	参数说明
	无	无
返回值	QLabel*	应用图标的label
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++
原型	QLabel* debVersionLabel();

描述	获取包版本的label	
参数	参数名称	参数说明
	无	无
返回值	QLabel*	包版本的label
备注	无	

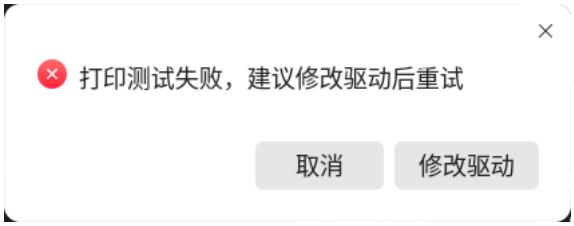
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPushButton* uninstallButttton();	
描述	获取卸载按钮pushbutton	
参数	参数名称	参数说明
	无	无
返回值	QPushButton*	卸载按钮pushbutton
备注	无	

| QPushButton* uninstallButttton(); | 获取卸载按钮pushbutton |

4.1.2.6 消息框

功能描述：KMessageBox，继承自KDialog，提供一个可自定义设置按钮和图标的对话框，类似QMessageBox

```
KMessageBox* messageBox = new KMessageBox(this);
```



枚举类型

枚举类型	enum Icon{ TNoIcon ,Information,Warning ,Critical ,Question };
	enum ButtonRole{ InvalidRole，AcceptRole，RejectRole,DestructiveRole,ActionRole,HelpRole,YesRole,NoRole,ResetRole,ApplyRole,N
	enum StandardButton{ NoButton,Ok,Save,SaveAll,Open,Yes,YesToAll,No,NoToAll,Abort,Retry,Ignore,Close,Cancel,Discard,Help,Apply,Reset,RestoreDefaults,Fi = Ok,LastButton = RestoreDefaults,YesAll = YesToAll, NoAll = NoToAll, Default = 0x00000100,Escape = 0x00000200,FlagMask = 0x0000 ButtonMask = ~FlagMask};

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setCustomIcon(const QIcon&icon)	
描述	自定义KMessageBox的提示图标	
参数	参数名称	参数说明
	QIcon	设置的图标
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++
原型	void addButton(QAbstractButton *button,ButtonRole role)

描述	添加一个自定义按钮	
参数	参数名称	参数说明
	button	自定义按钮
	role	按钮角色
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPushButton* addButton(const QString &text,ButtonRole role)	
描述	添加设置好文本的按钮	
参数	参数名称	参数说明
	text	按钮文本
	role	按钮角色
返回值	QPushButton*	给定文本和角色的按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPushButton* addButton(StandardButton button)	
描述	添加一个标准按钮并且返回这个按钮	
参数	参数名称	参数说明
	button	添加的标准按钮
返回值	QPushButton*	需要添加的标准按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void removeButton(QAbstractButton *button)	
描述	移除一个按钮	
参数	参数名称	参数说明
	button	需要移除的按钮
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QAbstractButton *button(StandardButton which) const	
描述	返回与标准按钮对应的指针， 如果此消息框中不存在标准按钮，则返回0	
参数	参数名称	参数说明
	which	标准按钮
返回值	QAbstractButton*	标准按钮的指针，

		不存在返回0
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QList buttons() const	
描述	返回已添加到消息框中的所有按钮的列表	
参数	参数名称	参数说明
	无	无
返回值	QList	所有按钮的列表
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KMessageBox::ButtonRole buttonRole(QAbstractButton *button) const	
描述	返回指定按钮的按钮角色， 如果按钮为0或尚未添加到消息框中， 此函数将返回InvalidRole	
参数	参数名称	参数说明
	button	指定按钮
返回值	ButtonRole	指定按钮的角色属性
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QCheckBox checkBox() const	
描述	返回KMessageBox中显示的复选框	
参数	参数名称	参数说明
	无	无
返回值	QCheckBox	MessageBox中显示的复选框
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setCheckBox(QCheckBox *cb)	
描述	设置KMessageBox显示的复选框，未设置则为0	
参数	参数名称	参数说明
	cb	设置KMessageBox显示的复选框，未设置则为0
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	

原型	QString text() const	
描述	获取KMessageBox的文本	
参数	参数名称	参数说明
	无	无
返回值	QString	messagebox文本
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setText（const QString& text）	
描述	设置KMessageBox的文本	
参数	参数名称	参数说明
	text	需要设置的文本
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString informativeText() const	
描述	获取KMessageBox信息性文本的描述	
参数	参数名称	参数说明
	无	无
返回值	QString	获取KMessageBox信息性文本的描述
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setInformativeText(const QString &text)	
描述	设置KMessageBox信息性文本的描述	
参数	参数名称	参数说明
	text	设置KMessageBox信息性文本的描述
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	ICon icon() cosnt	
描述	获取KMessageBox的图标	
参数	参数名称	参数说明
	无	无
返回值	ICon	获取KMessageBox的图标
备注	无	

子模块	libkysdk-qtwidgets

接口类型	C++	
原型	void setIcon(Icon icon)	
描述	设置KMessageBox的图标	
参数	参数名称	参数说明
	icon	设置KMessageBox的图标
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPixmap iconPixmap() const	
描述	返回当前KMessageBox的icon	
参数	参数名称	参数说明
	无	无
返回值	QPixmap	返回当前KMessageBox的icon
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIconPixmap(const QPixmap &pixmap)	
描述	设置当前KMessageBox的icon	
参数	参数名称	参数说明
	pixmap	设置当前KMessageBox的icon
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KMessageBox::StandardButtons standardButtons() const	
描述	KMessageBox中标准按钮的集合	
参数	参数名称	参数说明
	无	无
返回值	KMessageBox::StandardButtons	KMessageBox中标准按钮的集合
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setStandardButtons(KMessageBox::StandardButtons buttons)	
描述	设置多个标准按钮	
参数	参数名称	参数说明
	buttons	设置多个标准按钮
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KMessageBox::StandardButton standardButton(QAbstractButton *button) const	
描述	返回与给定按钮对应的标准按钮枚举值，如果给定按钮不是标准按钮，则返回NoButton	
参数	参数名称	参数说明
	button	传入按钮
返回值	KMessageBox::StandardButton	返回与给定按钮对应的标准按钮枚举值，如果给定按钮不是标准按钮，则返回NoButton
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPushButton* defaultButton() const	
描述	返回KMessageBox的默认按钮	
参数	参数名称	参数说明
	无	无
返回值	QPushButton*	返回KMessageBox的默认按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setDefaultButton(QPushButton *button)	
描述	设置KMessageBox的默认按钮	
参数	参数名称	参数说明
	button	设置KMessageBox的默认按钮
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setDefaultButton(KMessageBox::StandardButton button)	
描述	设置KMessageBox的默认按钮	
参数	参数名称	参数说明
	button	设置KMessageBox的默认按钮
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QAbstractButton* clickedButton() const	
描述	返回被点击的按钮	
参数	参数名称	参数说明

	无	无
返回值	QAbstractButton*	被点击的按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	static QPixmap standardIcon(Icon icon)	
描述	返回用于标准图标的pixmap	
参数	参数名称	参数说明
	icon	传入的图标
返回值	QPixmap	返回用于标准图标的pixmap
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	static StandardButton information(QWidget *parent,const QString &title,const QString &text,StandardButton buttons =OK ,StandardButton defaultButton = NoButton)	
描述	打开带有给定标题和文本的信息消息框， 对话框最多可以有三个按钮	
参数	参数名称	参数说明
	parent	父类
	title	窗口标题
	text	提示文本
	buttons	添加按钮
	defaultButton	NoButton
返回值	StandardButton	返回被单击的标准按钮的标识。 如果按了Esc键， 则返回escape键
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	static StandardButton qusetion(QWidget *parent,const QString &title,const QString *text,StandardButtons buttons =StandardButton(Yes | No),StandardButton defaultButton = NoButton)	
描述	打开一个带有给定标题和文本的问题消息框	
参数	参数名称	参数说明
	parent	父类
	title	窗口标题
	text	提示文本
	buttons	添加按钮
	defaultButton	NoButton
返回值	StandardButton	返回被单击的标准按钮的标识。 如果按了Esc键，

		则返回escape键
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	static StandardButton warning(QWidget *parent,const QString &title,const QString &text,StandardButtons buttons =Ok,StandardButton defaultButton = NoButton)	
描述	打开一个带有给定标题和文本的警告消息框	
参数	参数名称	参数说明
	parent	父类
	title	窗口标题
	text	提示文本
	buttons	添加按钮
	defaultButton	NoButton
返回值	StandardButton	返回被单击的标准按钮的标识。 如果按了Esc键， 则返回escape键
备注	无	

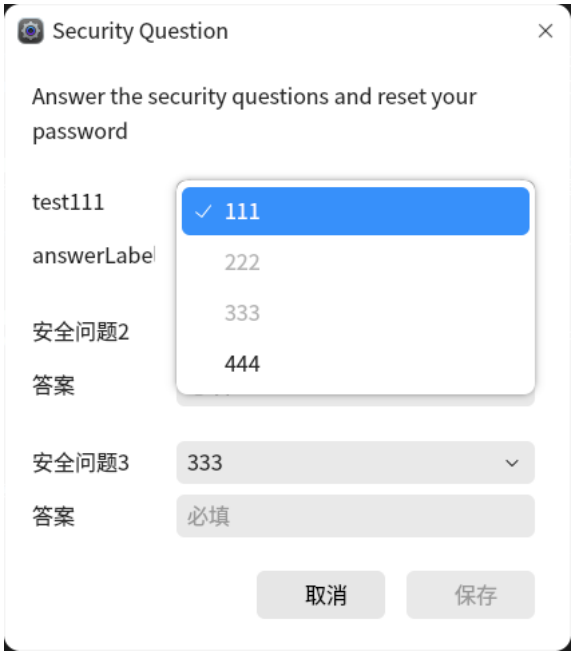
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	static StandardButton critical(QWidget *parent,const QString *title,const QString &text,StandardButtons buttons = Ok,StandardButton defaultButton = NoButton)	
描述	用给定的标题和文本打开一个关键信息框	
参数	参数名称	参数说明
	parent	父类
	title	窗口标题
	text	提示文本
	buttons	添加按钮
	defaultButton	NoButton
返回值	StandardButton	返回被单击的标准按钮的标识。 如果按了Esc键， 则返回escape键
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	static StandardButton success(QWidget *parent, const QString &title,const QString &text, StandardButtons buttons = Ok,StandardButton defaultButton = NoButton);	
描述	用给定的标题和文本打开一个成功消息框	
参数	参数名称	参数说明
	parent	父类
	title	窗口标题

	text	提示文本
	buttons	添加按钮
	defaultButton	NoButton
返回值	StandardButton	返回被单击的标准按钮的标识。 如果按了Esc键， 则返回escape键
备注	无	

4.1.2.7 KSecurityQuestionDialog

功能说明：安全问题对话框，类似qq密保问题验证（自2.4启用）



子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTitleText(const QString &text);	
描述	设置对话框标题	
参数	参数名称	参数说明
	text	对话框标题
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void addSecurityQuestionItem(const int count, bool mutex);	
描述	添加安全问题控件	
参数	参数名称	参数说明
	count	控件个数
	布尔值	是否开启下拉框互斥校验， 若开启后，已被选择项， 在其他下拉框中被置灰
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void initQustionCombox(const QStringList &questionlist);	
描述	添加下拉框选项列表	
参数	参数名称	参数说明
	questionlist	下拉框列表
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QLabel *questionLabel(const int questionIndex);	
描述	获取下拉框左侧Label	
参数	参数名称	参数说明
	questionIndex	item索引
返回值	QLabel*	下拉框左侧Label， 若未获取到，返回nullptr
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QComboBox *questionCombox(const int questionIndex);	
描述	获取combox	
参数	参数名称	参数说明
	questionIndex	item索引
返回值	QComboBox*	下拉框，若未获取到， 返回nullptr
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QLabel *answerLabel(const int answerIndex);	
描述	获取lineedit左侧label	
参数	参数名称	参数说明
	answerIndex	item索引
返回值	QLabel*	lineedit左侧label， 若未获取到，返回nullptr
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QLineEdit *answerLineedit(const int answerIndex);	
描述	获取 lineedit	
参数	参数名称	参数说明

	answerIndex	item索引
返回值	QLineEdit*	lineedit，若未获取到，返回nullptr
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QLabel *tipsLabel(const int index);	
描述	获取提示 label	
参数	参数名称	参数说明
	index	item 索引
返回值	QLabel*	提示label，若未获取到，返回nullptr
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KPushButton *cancelButton();	
描述	获取取消按钮	
参数	参数名称	参数说明
	无	无
返回值	KPushButton*	获取取消按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KPushButton *confirmButton();	
描述	获取确认按钮	
参数	参数名称	参数说明
	无	无
返回值	KPushButton*	获取确认按钮
备注	无	

4.1.3 输入框模块

4.1.3.1 密码输入框

功能描述：KPasswordEdit，支持切换输入内容明文/暗文的切换，支持设置密码校验结果（正常输入为蓝色边框，设置密码正确为绿色边框，密码错误为红色边框），支持 loading 状态，支持设置是否启用 clearButton。

枚举类型

枚举类型	enum LoginState{Ordinary,LoginSuccess,LoginFailed};

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setState(LoginState state);	
描述	设置登录状态	
参数	参数名称	参数说明
	state	LoginState类枚举，显示登录状态
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	LoginState state();	
描述	返回登录状态	
参数	参数名称	参数说明
	无	无
返回值	LoginState	获取搜索框的登录状态
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setLoading(bool flag);	
描述	设置是否启用加载状态。	
参数	参数名称	参数说明
	布尔值	true 启用加载状态，false 不启用加载状态
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isLoading();	
描述	获取是否处于加载状态	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 启用加载状态，false 不启用加载状态
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString placeholderText();	

描述	返回placeholderText的文本内容	
参数	参数名称	参数说明
	无	无
返回值	QString	placeholder的文本内容
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setPlaceholderText(QString&text);	
描述	设置PlaceholderText的文本内容	
参数	参数名称	参数说明
	text	设置placeholder的文本内容
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setClearButtonEnabled(bool enable);	
描述	设置是否启用ClearButton	
参数	参数名称	参数说明
	bool	true 启用清除按钮，false 禁用清除按钮
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isClearButtonEnabled() const;	
描述	获取是否启用了ClearButton	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 启用清除按钮，false 禁用清除按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setEchoModeBtnVisible(bool enable);	
描述	设置EchoModeBtn是否可见,1.2版本添加	
参数	参数名称	参数说明
	布尔值	true 文本样式按钮可见，false 文本样式按钮不可见
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool echoModeBtnVisible();	
描述	获取EchoModeBtn是否可见,1.2版本添加	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 文本样式按钮可见， false 文本样式按钮不可见
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setClearBtnVisible(bool enable);	
描述	设置ClearBtn是否可见,1.2版本添加	
参数	参数名称	参数说明
	布尔值	true 清除按钮可见，false 清楚按钮不可见
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool clearBtnVisible();	
描述	获取ClearBtn是否可见,1.2版本添加	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 清除按钮可见，false 清楚按钮不可见
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setEnabled(bool);	
描述	设置KPasswordEdit是否可用	
参数	参数名称	参数说明
	布尔值	true 控件可用，false 控件不可用
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setEchoMode(EchoMode mode)	
描述	设置文本的显示模式，1.2版本添加	
参数	参数名称	参数说明

	mode	决定文本显示样式
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setUseCustomPalette(bool flag)	
描述	设置是否走默认palette,1.2版本添加	
参数	参数名称	参数说明
	布尔值	true 跟随默认palette，false 跟随自定义palette
返回值	无	无
备注	无	

4.1.3.2 搜索输入框

功能描述：可以设置 placeHolder 的文字内容及对齐方式，输入文字的对齐方式，是否启用清除按钮等。

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setEnabled(bool);	
描述	设置是否可用	
参数	参数名称	参数说明
	bool	true 控件可用，false 控件不可用
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isEnabled();	
描述	获取是否可用	
参数	参数名称	参数说明
	无	无
返回值	bool	true 控件可用，false 控件不可用
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setClearButtonEnabled(bool enable);	
描述	设置是否显示清除按钮	

参数	参数名称	参数说明
	bool	true 显示清除按钮，false 不显示清除按钮
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isClearButtonEnabled() const;	
描述	获取是否显示清除按钮	
参数	参数名称	参数说明
	无	无
返回值	bool	true 显示清除按钮，false 不显示清除按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString placeholderText() const;	
描述	返回placeholder	
参数	参数名称	参数说明
	无	无
返回值	QString	返回placeholder文本
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setPlaceholderText(const QString &);	
描述	设置placeholder的文本	
参数	参数名称	参数说明
	QString	设置placeholder的文本
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	Qt::Alignment placeholderAlignment() const;	
描述	返回placeholder的对齐方式	
参数	参数名称	参数说明
	无	无
返回值	Qt::Alignment	返回placeholder的对齐方式
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++

原型	void setPlaceholderAlignment(Qt::Alignment flag);	
描述	设置placeholder的对齐方式	
参数	参数名称	参数说明
	Qt::Alignment	设置placeholder的对齐方式
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	Qt::Alignment alignment() const;	
描述	返回输入文本的对齐方式	
参数	参数名称	参数说明
	无	无
返回值	Qt::Alignment	返回输入文本的对齐方式
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setAlignment(Qt::Alignment flag)；	
描述	设置输入文本的对齐方式。	
参数	参数名称	参数说明
	Qt::Alignment	设置输入文本的对齐方式。
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void clear()；	
描述	清空搜索框内容。	
参数	参数名称	参数说明
	无	无
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTranslucent(bool flag);	
描述	设置是否启用半透明效果（自1.2.0.10启用）	
参数	参数名称	参数说明
	bool	true 启用半透明效果，false 禁用半透明效果
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isTranslucent();	
描述	获取是否启用半透明效果（自1.2.0.10启用）	
参数	参数名称	参数说明
	无	无
返回值	bool	true 启用半透明效果，false 禁用半透明效果
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void reloadStyle()	
描述	重新加载style,（自2.0启用）	
参数	参数名称	参数说明
	无	无
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KToolButton *customButton()	
描述	获取自定义按钮 （自2.4启用）	
参数	参数名称	参数说明
	无	无
返回值	KToolButton*	获取到自定义按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setGradation(bool flag)	
描述	设置是否开启渐变色（自2.4启用）	
参数	参数名称	参数说明
	bool	true 开启渐变色，false 关闭渐变色
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setCustomButtonVisible(bool flag)	
描述	设置自定义按钮是否可见（自2.4启用）	
参数	参数名称	参数说明
	布尔值	true 自定义按钮可见，false

		自定义按钮不可见
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isCustomButtonVisible()	
描述	获取自定义按钮是否可见（自2.4启用）	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 自定义按钮可见，false 自定义按钮不可见
备注	无	

4.1.3.3 文本框

功能描述：KTextEdit继承自QTextEdit的文本框，自2.4启用

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setLabelVisible(bool flag)	
描述	设置label是否可见	
参数	参数名称	参数说明
	布尔值	true label可见，false label不可见

返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setPixmap(const QPixmap &pixmap)	
描述	设置图标	
参数	参数名称	参数说明
	pixmap	将pixmap设置给label
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	const QPixmap *pixmap()const	
描述	获取图标	
参数	参数名称	参数说明
	无	无
返回值	QPixmap*	返回label的QPixmap
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setLabelText(const QString &text)	
描述	设置label文本	
参数	参数名称	参数说明
	text	设置的文本内容
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setReadOnly（bool ro）	
描述	设置是否可编辑状态	
参数	参数名称	参数说明
	布尔值	true textedit可编辑，false textedit不可编辑
返回值	无	无
备注	无	

4.1.4 按钮模块

4.1.4.1 带边框按钮

> 功能描述：KBoderButton,继承自 QPushButton,样式上进行了封装。可以通过setPalette()进一步调整样式。如：

```
QPalette palette = m_pBtn1->palette();
palette.setColor(QPalette::ButtonText,QColor(255,0,0));
m_pBtn1->setPalette(palette);
```

> QPushButton 中的各接口均适用，支持四种构造方法。

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KBorderButton(QWidget* parent = nullptr);	
描述	仅一个button	
参数	参数名称	参数说明
	parent	父类
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KBorderButton(const QString &text, QWidget *parent = nullptr);	
描述	构造一个带文本button	
参数	参数名称	参数说明
	text	给button设置文本text
	parent	父类
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KBorderButton(const QIcon &icon, const QString &text, QWidget *parent = nullptr);	
描述	构造一个带文本和图标的button	
参数	参数名称	参数说明
	icon	button设置的图标icon
	text	给button设置的文本text
	parent	父类
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KBorderButton(const QIcon &icon, QWidget *parent = nullptr);	
描述	构造一个带图标的button	
参数	参数名称	参数说明
	icon	button设置的图标
	parent	父类

返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIcon(const QIcon &icon);	
描述	设置按钮图标	
参数	参数名称	参数说明
	icon	button设置的图标icon
返回值	无	无
备注	无	

4.1.4.2 无边框按钮

功能描述：KboderlessButton,继承自 QPushButton,样式上进行了封装。同样可以可以通过 setPalette()进一步调整样式。

QPushButton 中的各接口均适用，支持四种构造方法。

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KBorderlessButton(QWidget* parent = nullptr);	
描述	仅一个button	
参数	参数名称	参数说明
	parent	父类
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KBorderlessButton(const QString &text, QWidget *parent = nullptr);	
描述	构造一个带文本button	
参数	参数名称	参数说明
	text	给button设置的文本
	parent	父类
返回值	无	无

备注	无

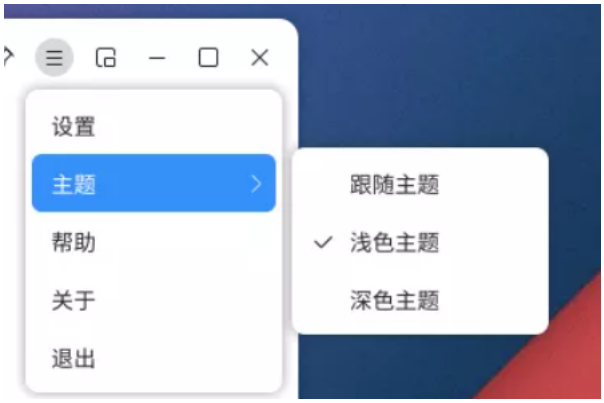
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KBorderlessButton(const QIcon &icon, const QString &text, QWidget *parent = nullptr);	
描述	构造一个带文本和图标的button	
参数	参数名称	参数说明
	icon	给button设置的图标icon
text	给button设置的文本	
parent	父类	
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KBorderlessButton(const QIcon &icon, QWidget *parent = nullptr);	
描述	构造一个带图标的button	
参数	参数名称	参数说明
	icon	给button设置图标icon
	parent	父类
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIcon(const QIcon &icon)	
描述	设置无边框按钮图标。	
参数	参数名称	参数说明
	icon	给button设置的图标
返回值	无	无
备注	无	

4.1.4.3 下拉菜单按钮

功能描述：KMenuButton，继承自 QPushButon，默认 ICon 为"open-menu-symbolic"，一级菜单中包含 5 个选项，分别是："设置"，"主题"，"帮助"，"关于"，"退出"，主题中包括 3 个二级选项，分别是："跟随主题"，"浅色主题"，"深色主题"。



QPushButton 的各个接口均适用。

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QMenu* menu();	
描述	获取主菜单	
参数	参数名称	参数说明
	无	无
返回值	QMenu*	返回主菜单menu
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QMenu* themeMenu();	
描述	获取主题菜单	
参数	参数名称	参数说明
	无	无
返回值	QMenu*	主题菜单menu
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QAction* settingAction();	
描述	获取设置action	
参数	参数名称	参数说明
	无	无
返回值	QAction*	获取设置action
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QAction* themeAction();	
描述	获取主题Action	
参数	参数名称	参数说明
	无	无
返回值	QAction*	获取主题Action
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QAction* assistAction();	
描述	获取帮助Action	
参数	参数名称	参数说明
	无	无

返回值	QAction*	获取帮助Action
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QAction* aboutAction()	
描述	获取关于Action	
参数	参数名称	参数说明
	无	无
返回值	QAction*	获取关于Action
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QAction* quitAction();	
描述	获取离开Action	
参数	参数名称	参数说明
	无	无
返回值	QAction*	获取离开Action
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QAction* autoAction();	
描述	获取跟随主题Action	
参数	参数名称	参数说明
	无	无
返回值	QAction*	获取跟随主题Action
备注	无	

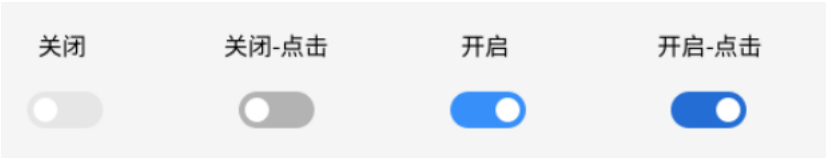
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QAction* lightAction();	
描述	获取浅色主题Action	
参数	参数名称	参数说明
	无	无
返回值	QAction*	获取浅色主题Action
备注	无	

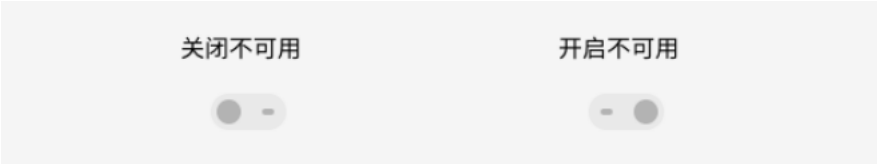
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QAction* darkAction();	
描述	获取深色主题Action	
参数	参数名称	参数说明

	无	无
返回值	QAction*	获取深色主题Action
备注	无	

4.1.4.4 开关按钮

功能描述：KSwitchButton，继承自 QPushButton，对按钮进行了重绘，用于指示开/关状态。





子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setCheckable(bool);	
描述	设置是否可选中	
参数	参数名称	参数说明
	布尔值	true 可选中，false 不可选中
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isCheckable() const;	
描述	获取是否可选中	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 可选中，false 不可选中
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isChecked() const;	
描述	获取是否选中	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 选中，false 未选中
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++
原型	void setChecked(bool);

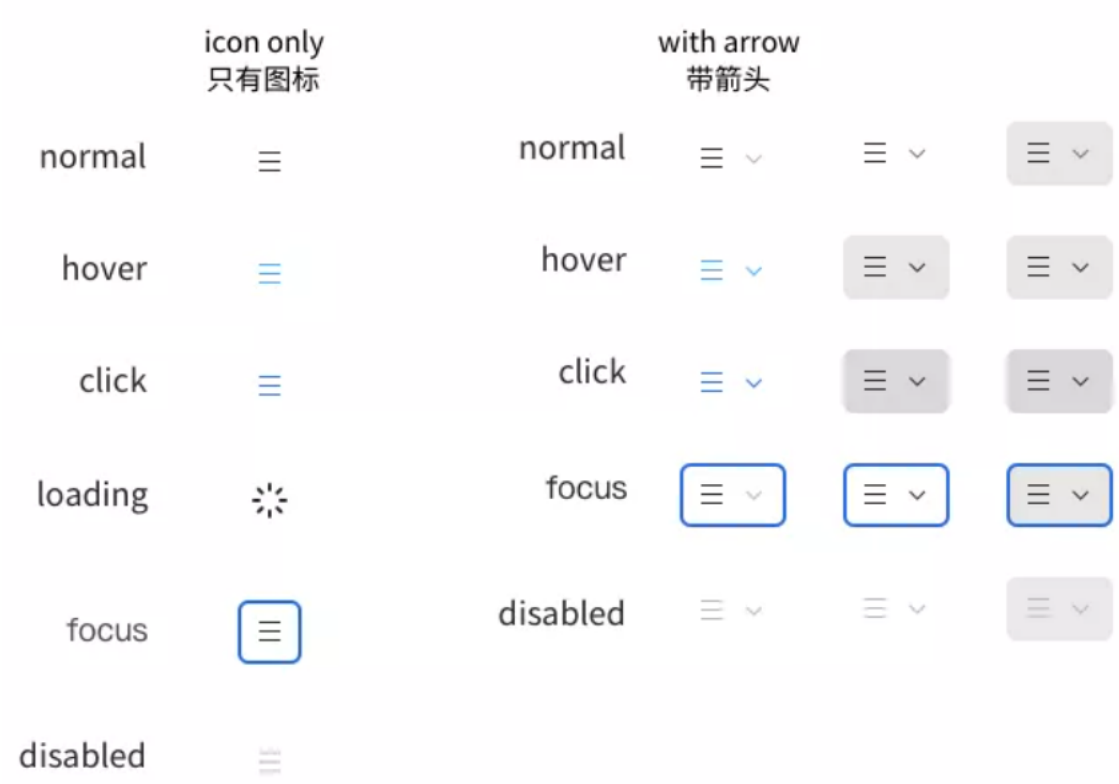
描述	设置是否选中	
参数	参数名称	参数说明
	布尔值	true 选中，false 未选中
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTranslucent(bool flag);	
描述	设置是否启用半透明效果（自1.2.0.10启用）	
参数	参数名称	参数说明
	布尔值	true 启用半透明效果 ，false 不启用半透明效果
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isTranslucent()	
描述	获取是否启用半透明效果（自1.2.0.10启用）	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 启用半透明效果 ，false 不启用半透明效果
备注	无	

4.1.4.5 工具按钮

> 功能描述：KToolButton，继承自 QToolButton，支持三种样式，支持 loading状态，支持是否显示下拉按钮。



枚举类型

枚举类型	enum KToolButtonType{Flat,SemiFlat,Background};

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KToolButtonType type();	
描述	返回类型	
参数	参数名称	参数说明
	无	无
返回值	KToolButtonType	toolbutton类型
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setType(KToolButtonType type);	
描述	设置类型	
参数	参数名称	参数说明
	type	设置toolbutton的类型
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIcon(const QIcon& icon);	
描述	设置Icon	
参数	参数名称	参数说明

	icon	为toolbutton设置的图标
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setLoading(bool flag);	
描述	设置正在加载状态,仅不带箭头的toolbuttuon支持该状态	
参数	参数名称	参数说明
	布尔值	true 加载状态，false 正常状态
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isLoading();	
描述	获取是否正在加载	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 加载状态，false 正常状态
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QIcon icon();	
描述	获取Icon	
参数	参数名称	参数说明
	无	无
返回值	QIcon	获取button的图标
备注	无	

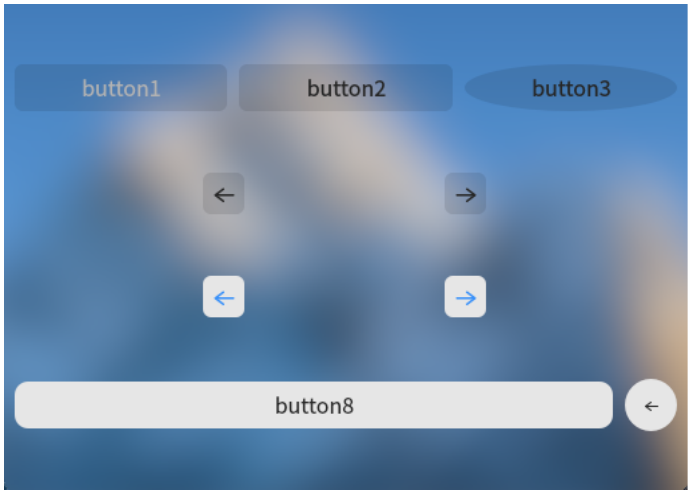
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setArrow(bool flag);	
描述	设置是否显示向下箭头，默认不显示	
参数	参数名称	参数说明
	布尔值	true 显示箭头，false 不显示箭头
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	

原型	bool hasArrow() const;	
描述	获取是否显示箭头	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 显示箭头，false 不显示箭头
备注	无	

4.1.4.6 KPushButton

功能描述：KPushButton，继承自 QPushButton，在 QPushButton 的基础上新增提供半透明效果，可以设置按钮圆角、背景色、图标是否跟随系统高亮，还可以设置按钮类型，分为正常类型和圆形（需要设置大小实现正圆）。自 1.2.0.10 版本启用。



枚举类型

枚举类型	enum ButtonType { NormalType, CircleType ,ShadowType(2.4启用)};
	enum ArrowDirection { ArrowTop,ArrowBottom,ArrowLeft,ArrowRight}（2.4启用）

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius(int radius);	
描述	通过圆角半径设置按钮圆角（每个圆角相同）	
参数	参数名称	参数说明
	radius	四个角的圆角半径
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius(int bottomLeft,int topLeft,int topRight,int bottomRight);	
描述	通过四个点来设置圆角	
参数	参数名称	参数说明
	bottomLeft	左下角圆角半径
	topLeft	左上角圆角半径

	topRight	右上角圆角半径
	bottomRight	右下角圆角半径
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int borderRadius();	
描述	获取按钮圆角	
参数	参数名称	参数说明
	无	无
返回值	int	返回四个角的圆角半径（四个圆角半径相同时可用）
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBackgroundColor(QColor color);	
描述	设置按钮背景色	
参数	参数名称	参数说明
	color	button的背景色
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QColor backgroundColor();	
描述	获取按钮背景色	
参数	参数名称	参数说明
	无	无
返回值	QColor	button的背景色
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setButtonType(ButtonType type);	
描述	设置KPushButton的类型	
参数	参数名称	参数说明
	type	设置button的类型
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	

接口类型	C++	
原型	ButtonType buttonType();	
描述	获取KPushButton的类型	
参数	参数名称	参数说明
	无	无
返回值	ButtonType	button的类型
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTranslucent(bool flag);	
描述	设置KPushButton是否为半透明	
参数	参数名称	参数说明
	布尔值	true 启用半透明，fasle 禁用半透明
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isTranslucent();	
描述	获取KPushButton是否为半透明	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 启用半透明，fasle 禁用半透明
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIconHighlight(bool flag);	
描述	设置图标是否跟随系统高亮色，默认不跟随	
参数	参数名称	参数说明
	布尔值	true 跟随系统高亮色，false 不跟随系统高亮色
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isIconHighlight();	
描述	获取图标是否跟随系统高亮色	
参数	参数名称	参数说明
	无	无

返回值	布尔值	true 跟随系统高亮色，false 不跟随系统高亮色
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIconColor(QColor color);	
描述	设置按钮添加图标的颜色（自1.2.0.13启用）	
参数	参数名称	参数说明
	color	设置图标的颜色
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QColor IconColor();	
描述	获取按钮添加图标的颜色（自1.2.0.13启用）	
参数	参数名称	参数说明
	无	无
返回值	QColor	获取自定义图标颜色
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBackgroundColorHighlight(bool flag);	
描述	设置按钮背景色是否跟随系统高亮色，默认不跟随（自1.2.0.13启用）	
参数	参数名称	参数说明
	布尔值	true button背景色跟随系统高亮色， false button背景色不跟随系统高亮色
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isBackgroundColorHighlight();	
描述	获取按钮背景色是否跟随系统高亮色（自1.2.0.13启用）	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true button背景色跟随系统高亮色， false button背景色不跟随系统高亮色
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setArrow(bool flag,ArrowDirection arrowDirection = ArrowBottom);	
描述	设置是否显示箭头，设值箭头方向（自2.4启用）	
参数	参数名称	参数说明
	布尔值	true 显示箭头，fasle 不显示箭头
	arrowDirection	箭头方向
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool hasArrow() const ;	
描述	返回是否显示箭头 （自2.4启用）	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 显示箭头，fasle 不显示箭头
备注	无	

4.1.4.7 KPressButton

> 功能描述：KPressButton,继承自 QPushButton，在 QPushButton 的基础上新增提供半透明效果，可以设置圆角，是否可选中，是否选中，还可以设置按钮的类型以及是否启用 loading 的状态。自 1.2.0.10 版本启用。

> 枚举类型

枚举类型	enum ButtonType { NormalType, CircleType };

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius(int radius);	
描述	通过圆角半径设置按钮圆角（每个圆角相同）	
参数	参数名称	参数说明
	radius	圆角半径
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius(int bottomLeft,int topLeft,int topRight,int bottomRight);	
描述	通过四个点来设置圆角	
参数	参数名称	参数说明
	bottomLeft	左下角圆角半径
	topLeft	左上角圆角半径
	topRight	右上角圆角半径
	bottomRight	有下角圆角半径
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setCheckable(bool);	
描述	设置是否可选中	
参数	参数名称	参数说明
	布尔值	true 可选中，false 不可选中
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isCheckable() const;	
描述	获取是否可选中	
参数	参数名称	参数说明
	布尔值	true 可选中，false 不可选中
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setChecked(bool);	
描述	设置是否选中	
参数	参数名称	参数说明
	布尔值	true 选中，false 未选中
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++
原型	bool isChecked() const;

描述	获取是否选中	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 选中，false 未选中
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setButtonType(ButtonType type);	
描述	设置button类型	
参数	参数名称	参数说明
	type	button的类型
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	ButtonType buttonType();	
描述	获取KPressButton的类型	
参数	参数名称	参数说明
	无	无
返回值	ButtonType	获取button的类型
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTranslucent(bool flag);	
描述	设置KPressButton是否为半透明	
参数	参数名称	参数说明
	布尔值	true 启用半透明，false 禁用半透明
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isTranslucent();	
描述	bool isTranslucent();	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 启用半透明，false 禁用半透明
备注	无	

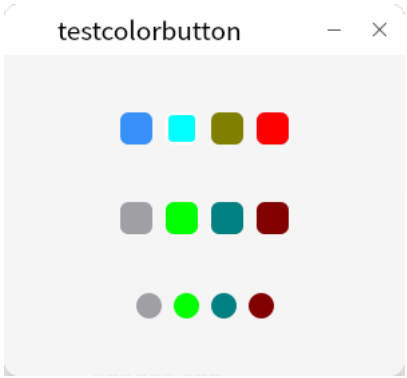
子模块	libkysdk-qtwidgets

接口类型	C++	
原型	void setLoaingStatus(bool flag);	
描述	设置是否启用loading状态	
参数	参数名称	参数说明
	布尔值	true loading状态，false 正常状态
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isLoading();	
描述	获取是否启用loading状态	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true loading状态，false 正常状态
备注	无	

4.1.4.8 KColorButton

功能描述:提供用于设置颜色和样式的按钮，以适用于不同的场景



枚举类型	enum Circle,RoundedRect,CheckedRect
	enum TailLocation{ LeftLocation, MiddleLocation, RightLocation };

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBackgoundColor(QColor color)	
描述	设置colorButton背景色	
参数	参数名称	参数说明
	color	设置button背景色
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++

原型	QColor backgroundColor()	
描述	获取colorButton背景色	
参数	参数名称	参数说明
	无	无
返回值	QColor	button的背景色
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius(int radious)	
描述	设置圆角（RoundedRect，CheckedRect类型生效）	
参数	参数名称	参数说明
	radious	button的四个角的圆角半径
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int borderRadius()	
描述	返回圆角	
参数	参数名称	参数说明
	无	无
返回值	int	获取button的四个角的圆角半径
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setButtonType(KColorButton::ButtonType type)	
描述	设置colorbutton类型	
参数	参数名称	参数说明
	type	设置button的类型
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KColorButton::ButtonType buttonType()	
描述	返回colorButton类型	
参数	参数名称	参数说明
	无	无
返回值	KColorButton::ButtonType	获取button的类型
备注	无	

4.1.4.9 KAddFileButton

功能描述：提供用于选择文件/文件夹的按钮

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setText(QString str)	
描述	设置文本	
参数	参数名称	参数说明
	str	设置的文本内容
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QFileDialog *fileDialog()	
描述	获取filedialog	
参数	参数名称	参数说明
	无	无
返回值	QFileDialog*	获取filedialog
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setNameFilter(const QString &filter)	
描述	将文件对话框中使用的过滤器设置为给定的过滤器	
参数	参数名称	参数说明
	filter	设置需要过滤的内容
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setNameFilters(const QStringList &filters)	
描述	设置文件对话框中使用的过滤器	
参数	参数名称	参数说明
	filters	设置需要过滤的内容
返回值	无	无
备注	无	

4.1.5 Bar 模块

4.1.5.1 KIconBar

功能描述：KWidget 和 KDialog 的一个组成部分，用于显示图标和窗口名称，2.4新增。

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIcon(const QString& iconName);	
描述	设置图标名称	
参数	参数名称	参数说明
	iconName	需要设置的图标名称
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIcon(const QIcon& icon);	
描述	设置图标	
参数	参数名称	参数说明
	icon	button设置的图标
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setWidgetName(const QString& widgetName)	
描述	设置标题	
参数	参数名称	参数说明
	widgetName	设置窗体标题
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QLabel* nameLabel();	
描述	获取标题label	
参数	参数名称	参数说明
	无	无
返回值	QLabel*	获取标题label
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++
原型	QLabel* iconLabel();

描述	获取图标label	
参数	参数名称	参数说明
	无	无
返回值	QLabel*	获取图标label
备注	无	

4.1.5.2 KWindowButtonBar

功能描述：KWidget 和 KDialog 的一个组成部分，用于下拉菜单、最小化、最大化、关闭按钮。

枚举类型

枚举类型	enum MaximumButtonState{Maximum,Restore};

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPushButton* minimumButton();	
描述	获取最小化按钮	
参数	参数名称	参数说明
	无	无
返回值	QPushButton*	获取最小化按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPushButton* maximumButton();	
描述	获取最大化按钮	
参数	参数名称	参数说明
	无	无
返回值	QPushButton*	获取最大化按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPushButton* closeButton();	
描述	获取关闭按钮	
参数	参数名称	参数说明
	无	无
返回值	QPushButton*无	获取关闭按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KMenuButton* menuButton();	

描述	获取菜单按钮	
参数	参数名称	参数说明
	无	无
返回值	KMenuButton*	获取菜单按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	MaximumButtonState maximumButtonState();	
描述	获取最大化按钮的状态(最大化/恢复)	
参数	参数名称	参数说明
	无	无
返回值	MaximumButtonState	获取最大化按钮的状态 (最大化/恢复)
备注	无	

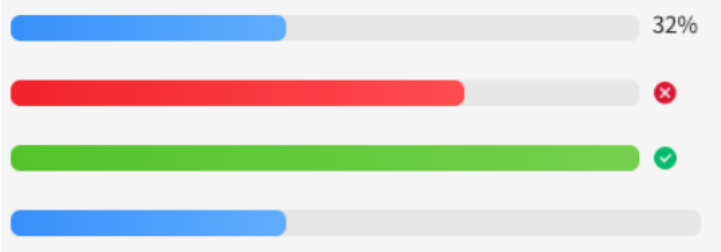
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setMaximumButtonState(MaximumButtonState state);	
描述	设置最大化按钮图标状态（最大化/恢复）	
参数	参数名称	参数说明
	MaximumButtonState	设置最大化按钮图标状态 （最大化/恢复）
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setFollowMode(bool flag);	
描述	设置是否遵循切换模式(自2.0启用)	
参数	参数名称	参数说明
	布尔值	true 遵循切换模式，false 不遵循切换模式
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool followMode();	
描述	获取是否遵循模式(自2.0启用)	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 遵循切换模式，false 不遵循切换模式
备注	无	

4.1.5.3 进度条

功能描述：KProgressBar，继承自 QProgressBar，支持三种状态：正常、完成和失败，支持是否显示进度值，支持水平和竖直。



枚举类型

枚举类型	enum ProgressBarState{NormalProgress,FailedProgress,SuccessProgress};

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	ProgressBarState state();	
描述	获取状态	
参数	参数名称	参数说明
	无	无
返回值	ProgressBarState	获取状态
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setState(ProgressBarState state);	
描述	设置状态	
参数	参数名称	参数说明
	state	设置状态
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString text() const override;	
描述	获取文本	
参数	参数名称	参数说明
	QString	获取文本
返回值	无	无
备注	无	

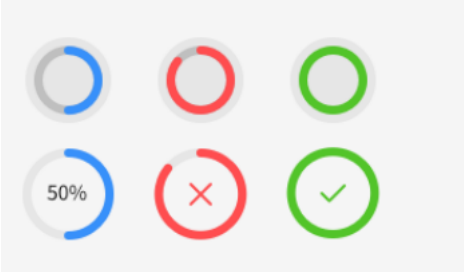
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setOrientation(Qt::Orientation);	
描述	设置方向	
参数	参数名称	参数说明

	Qt::Orientation	设置方向
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBodyWidth(int width)	
描述	设置进度条宽度(自1.2启用)	
参数	参数名称	参数说明
	width	设置进度条宽度
返回值	无	无
备注	无	

4.1.5.4 KProgressCircle

> 环形进度条，支持三种状态：正常、完成和失败，支持是否显示进度值。



子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int minimum() const	
描述	获取环形进度条的最小值。	
参数	参数名称	参数说明
	无	无
返回值	int	获取环形进度条的最小值。
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int maximum() const	
描述	获取环形进度条的最大值。	
参数	参数名称	参数说明
	无	无
返回值	int	获取环形进度条的最大值。
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int value() const	
描述	获取环形进度条的值。	

参数	参数名称	参数说明
	无	无
返回值	int	获取环形进度条的值。
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString text() const	
描述	获取环形进度条的文本。	
参数	参数名称	参数说明
	无	无
返回值	QString	获取环形进度条的文本。
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTextVisible(bool visible)	
描述	设置环形进度条的文本是否可见。	
参数	参数名称	参数说明
	布尔值	true 文本可见，false 文本不可见
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isTextVisible() const	
描述	获取环形进度条的文本是否可见。	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 文本可见，false 文本不可见
备注	无	

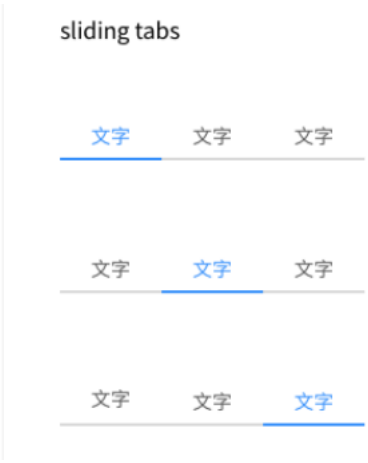
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	ProgressBarState state()	
描述	获取环形进度条的状态。	
参数	参数名称	参数说明
	无	无
返回值	ProgressBarState	环形进度条的状态
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++

原型	void setState(ProgressBarState state)	
描述	设置环形进度条的状态。	
参数	参数名称	参数说明
	state	设置环形进度条的状态。
返回值	无	无
备注	无	

4.1.5.5 KTabBar

功能描述：继承自 QTabBar



枚举类型

枚举类型	enum KTabBarStyle{SegmentDark,SegmentLight,Sliding};

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTabBarStyle(KTabBarStyle barStyle);	
描述	设置TabBar样式	
参数	参数名称	参数说明
	barStyle	设置tabbar的样式
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KTabBarStyle barStyle();	
描述	返回TabBar样式	
参数	参数名称	参数说明
	无	无
返回值	KTabBarStyle	返回tabbar的样式
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++

原型	void setBorderRadius(int radius);	
描述	设置圆角半径，只对SegmentDark，SegmentLight样式生效	
参数	参数名称	参数说明
	radius	设置控件圆角半径
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int borderRadius();	
描述	获取圆角半径	
参数	参数名称	参数说明
	无	无
返回值	int	返回控件的圆角半径
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBackgroundColor(const QColor &color)	
描述	设置KTabBar背景色。(自1.2启用)	
参数	参数名称	参数说明
	color	设置控件的背景色
返回值	无	无
备注	无	

2.4新增信号

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void rightlicked(QPoint point)	
描述	鼠标右键点击触发	
参数	参数名称	参数说明
	point	鼠标点击的位置
返回值	无	无
备注	无	

4.1.5.6 导航栏

功能描述：KNavigationBar，支持显示三种样式的 item，带图标的表示一级导航的 item，不带图标的表示二级的导航 item，还有表示组别的灰色字体的item。

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void addItem(QStandardItem*item);	
描述	增加常规Item	
参数	参数名称	参数说明

	item	需要增加的常规Item
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void addSubItem(QStandardItem*subItem);	
描述	增加次级Item	
参数	参数名称	参数说明
	subItem	需要增加的次级Item
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void addGroupItems(QListitems,const QString& tag);	
描述	成组增加Item,在导航栏中会显示tag	
参数	参数名称	参数说明
	items	需要增加item的列表
	tag	添加组的名称
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void addTag(const QString& tag);	
描述	添加tag	
参数	参数名称	参数说明
	tag	添加tag
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QStandardItemModel* model();	
描述	获取model	
参数	参数名称	参数说明
	无	无
返回值	QStandardItemModel*	获取model
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QListView* listview();	

描述	获取listview	
参数	参数名称	参数说明
	无	无
返回值	QListView*	获取listview
备注	无	

4.1.5.7 KPixmapContainer

功能描述：用于表示头像右上方消息提示信息，可以设置提示信息数值，字体大小，背景色，图片大小。

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int value() const;	
描述	获取值	
参数	参数名称	参数说明
	无	无
返回值	int	获取值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setValue(int value);	
描述	设置值	
参数	参数名称	参数说明
	value	设置值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setValueVisiable(bool flag);	
描述	设置值是否可见	
参数	参数名称	参数说明
	布尔值	true 值可见，fasle 值不可见
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isValueVisiable() const;	
描述	获取值是否可见	
参数	参数名称	参数说明
	无	无

返回值	布尔值	true 值可见，fasle 值不可见
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setPixmap(const QPixmap& pixmap);	
描述	设置pixmap	
参数	参数名称	参数说明
	pixmap	设置pixmap
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPixmap pixmap()const;	
描述	获取pixmap	
参数	参数名称	参数说明
	无	无
返回值	QPixmap	获取pixmap
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void clearValue();	
描述	清除值	
参数	参数名称	参数说明
	无	无
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QColor color();	
描述	返回背景色	
参数	参数名称	参数说明
	无	无
返回值	QColor	返回背景色
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setColor(const QColor& color);	
描述	设置背景色	
参数	参数名称	参数说明

	color	设置背景色
返回值	无	无
备注	无	

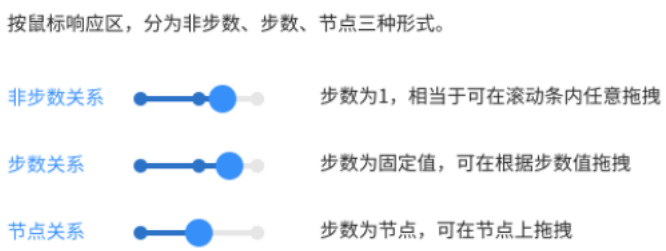
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int fontSize();	
描述	返回字体大小	
参数	参数名称	参数说明
	无	无
返回值	int	返回字体大小
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setFontSize(int size);	
描述	设置字体大小	
参数	参数名称	参数说明
	size	设置字体大小
返回值	无	无
备注	无	

4.1.6 滑动条模块

4.1.6.1 滑动条

> 功能描述：
>
> 非步数关系：步数为 1，可以在任意位置点击和拖拽；
>
> 步数关系：步数为固定值，可根据步数值点击和拖拽；
>
> 节点关系：步数为节点间隔，可根据节点间隔点击和拖拽



> 枚举类型

枚举类型	enum KSliderType{SmoothSlider,StepSlider,NodeSlider,SingleSelectSlider};

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTickInterval(int interval);	
描述	设置节点间隔	
参数	参数名称	参数说明

	interval	节点间隔
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setSliderType(KSliderType type);	
描述	设置滑动条类型	
参数	参数名称	参数说明
	type	设置滑动条类型
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KSliderType sliderType();	
描述	获取滑动条类型	
参数	参数名称	参数说明
	无	无
返回值	KSliderType	获取滑动条类型
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int tickInterval() const;	
描述	获取节点间隔	
参数	参数名称	参数说明
	无	无
返回值	int	获取节点间隔
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setValue(int);	
描述	设置值	
参数	参数名称	参数说明
	int	设置值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setNodeVisible(bool flag);	
描述	设置是否显示节点	

参数	参数名称	参数说明
	布尔值	true 显示节点，fasle 隐藏节点
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool nodeVisible();	
描述	获取是否显示节点	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 显示节点，fasle 隐藏节点
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setToolTip(const QString&);	
描述	设置tooltip（自1.2.0.7启用）	
参数	参数名称	参数说明
	QString	tooltip提示信息
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString toolTip() const;	
描述	获取toolTip（自1.2.0.7启用）	
参数	参数名称	参数说明
	无	无
返回值	QString	获取tooltip提示信息
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTranslucent(bool flag);	
描述	设置是否启用半透明效果（自1.2.0.10启用）	
参数	参数名称	参数说明
	布尔值	true 启用半透明效果，false 禁用半透明效果
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets

接口类型	C++	
原型	bool isTranslucent();	
描述	获取是否启用半透明效果（自1.2.0.10启用）	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 启用半透明效果，false 禁用半透明效果
备注	无	

4.1.7 消息提示模块

4.1.7.1 KBadge

> 功能描述：消息提醒气泡。可以设置提示信息数值，字体大小，背景色。

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int value();	
描述	返回当前值	
参数	参数名称	参数说明
	无	无
返回值	int	返回当前值
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setValue(int value);	
描述	设置值，最大显示数值为999，大于999显示"..."	
参数	参数名称	参数说明
	value	设置当前值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setValueVisiable(bool flag);	
描述	设置值是否可见	
参数	参数名称	参数说明
	布尔值	true 值可见，false 值不可见
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++
原型	bool isValueVisiable() const;

描述	获取值是否可见	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 值可见，false 值不可见
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QColor color();	
描述	获取背景色	
参数	参数名称	参数说明
	无	无
返回值	QColor	返回背景色
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setColor(const QColor& color);	
描述	设置背景色	
参数	参数名称	参数说明
	color	设置背景色
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int fontSize();	
描述	获取字体大小	
参数	参数名称	参数说明
	无	无
返回值	int	返回字体大小
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setFontSize(int size);	
描述	设置字体大小	
参数	参数名称	参数说明
	size	设置字体大小
返回值	无	无
备注	无	

4.1.7.2 KBallonTip

> 功能描述：消息提示框，支持四种背景色以及对应的提示图标。

枚举类型

枚举类型	enum TipType{Nothing,Normal,Info,Warning,Error};

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void showInfo();	
描述	气泡在显示之后一定时间后自动消失	
参数	参数名称	参数说明
	无	无
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTipType(const TipType& type);	
描述	设置类型	
参数	参数名称	参数说明
	type	设置控件类型
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	TipType tipType();	
描述	返回类型	
参数	参数名称	参数说明
	无	无

返回值	TipType	返回类型
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setText(const QString& text);	
描述	设置文本内容	
参数	参数名称	参数说明
	text	设置文本内容
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString text();	
描述	返回文本内容	
参数	参数名称	参数说明
	无	无
返回值	QString	返回文本内容
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setContentsMargins(int left, int top, int right, int bottom)	
描述	通过上下左右数值设置内容边距。	
参数	参数名称	参数说明
	left	左侧边距
	top	上方边距
	right	右侧边距
	bottom	下方边距
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setContentsMargins(const QMargins &margins);	
描述	通过QMargins对象设置内容边距。	
参数	参数名称	参数说明
	margins	设置四个方向的边距
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++

原型	void setTipTime(int my_time);	
描述	设置持续时间	
参数	参数名称	参数说明
	my_time	设置持续显示时间
返回值	无	无
备注	无	

4.1.7.3 KSecurityLevelBar

功能描述：密码强度提示条，用于指示密码强度等级。分为低，中，高三个等级，等级具体划分策略由应用程序指定。

枚举类型

枚举类型	enum SecurityLevel{Low,Medium,High};

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setSecurityLevel(SecurityLevel level);	
描述	设置安全等级	
参数	参数名称	参数说明
	level	枚举 设置安全等级
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	SecurityLevel securityLevel();	
描述	获取安全等级	
参数	参数名称	参数说明
	无	无
返回值	SecurityLevel	获取安全等级
备注	无	

4.1.8 容器模块

4.1.8.1 KBackgroundGroup

功能描述：:提供了一个用于存储部件的容器

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void addWidget(QWidget* widget)	
描述	添加一个widget控件	
参数	参数名称	参数说明
	widget	将widget添加到容器中
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void addWidgetList(QList list)	
描述	添加一个widget list	
参数	参数名称	参数说明
	list	将list中的wigdet逐个添加到容器中
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void removeWidget(int index)	
描述	删除一个指定index的widget	
参数	参数名称	参数说明
	index	删除一个指定index的widget
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void removeWidget（QWidget* widget）	
描述	删除一个widget	
参数	参数名称	参数说明
	widget）	将widget从容器中删除

返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void removeWidgetList(QList list)	
描述	删除一个widget list	
参数	参数名称	参数说明
	list	将list中的widget逐个删除
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void insertWidgetAt(int index,QWidget *widget)	
描述	指定位置插入一个widget	
参数	参数名称	参数说明
	index	需要插入的index
	widget	在index位置需要插入的widget
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void insertWidgetList(int index, QList list)	
描述	指定位置插入一个widget list	
参数	参数名称	参数说明
	index	从index位置开始插入
	list	从index位置开始， 逐个添加list中的widget
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius(int radius)	
描述	设置KBackgroundGroup的圆角	
参数	参数名称	参数说明
	radius	设置四个圆角半径
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets
接口类型	C++

原型	int borderRadius()	
描述	获取KBackgroundGroup的圆角	
参数	参数名称	参数说明
	无	无
返回值	int	获取四个角的圆角半径
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBackgroundRole(QPalette::ColorRole role)	
描述	设置背景颜色	
参数	参数名称	参数说明
	QPalette::ColorRole	设置背景色的role
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QPalette::ColorRole backgroundRole() cosnt	
描述	返回背景颜色	
参数	参数名称	参数说明
	无	无
返回值	QPalette::ColorRole	返回背景色的role
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setStateEnable(QWidget* widget,bool flag)	
描述	设置窗体是否可以响应三态	
参数	参数名称	参数说明
	widget	指定widget
	布尔值	true widget响应三态，false widget不响应三态
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QList widgetList()	
描述	返回widget列表	
参数	参数名称	参数说明
	无	无
返回值	QList	以列表的形式获取容器中所有的widget
备注	无	

4.1.8.2 KButtonBox

> 功能描述：提供了一个按钮类容器

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	Qt::Orientation orientation()	
描述	获取KButtonBox的布局类型	
参数	参数名称	参数说明
	无	无
返回值	Qt::Orientation	获取KButtonBox的布局类型
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setOrientation(Qt::Orientation orientation)	
描述	设置KButtonBox的布局类型，包括水平类型和垂直类型	
参数	参数名称	参数说明
	orientation	设置KButtonBox的布局类型， 包括水平类型和垂直类型
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void addButton(KPushButton *button,int i =-1)	
描述	添加按钮	
参数	参数名称	参数说明
	button	需要添加到buttonbox的button
	i	添加button的位置， 默认按顺序添加
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void removeButton(KPushButton *button)	
描述	删除按钮	
参数	参数名称	参数说明
	button	从buttonbox中删除按钮
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void removeButton(int i)	
描述	按id删除指定按钮	
参数	参数名称	参数说明
	i	删除指定id的button
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setButtonList(const QList &list)	
描述	以列表形式向KButtonBox中添加按钮	
参数	参数名称	参数说明
	list	以列表形式向KButtonBox中添加按钮
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QList buttonList()	
描述	获取KButtonBox中的按钮列表	
参数	参数名称	参数说明
	无	无
返回值	QList	获取KButtonBox中的按钮列表
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius(int radius)	
描述	设置KButtonBox首尾部分按钮的圆角	
参数	参数名称	参数说明
	radius	设置buttonbox的四角圆角半径
返回值	无	无

备注	无

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int borderRadius()	
描述	获取KButtonBox首尾部分按钮的圆角	
参数	参数名称	参数说明
	无	无
返回值	int	获取KButtonBox四个圆角半径
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setId(KPushButton *button,int id)	
描述	设置按钮id	
参数	参数名称	参数说明
	button	指定要设置id的button
	id	将指定button设置为指定id
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets		
接口类型	C++		
原型	int id(KPushButton *button)		
描述	获取按钮id		
参数	参数名称	参数说明	
	button	指定按钮button	
	返回值	int	获取指定按钮的id
备注	无		

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KPushButton* checkedButton()	
描述	返回已选中的按钮	
参数	参数名称	参数说明
	无	无
返回值	KPushButton*	获取当前选中的按钮
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KPushButton* button(int id)	
描述	通过按钮id获取按钮	
参数	参数名称	参数说明

	id	指定id
返回值	KPushButton*	获取指定id的button
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int checkedId()	
描述	返回已选中按钮的id	
参数	参数名称	参数说明
	无	无
返回值	int	获取当前选中按钮的id
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setExclusive(bool)	
描述	设置KButtonBox按钮间是否互斥	
参数	参数名称	参数说明
	布尔值	true 互斥，false 不互斥
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool exclusiv()	
描述	获取KButtonBox按钮间是否互斥	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 互斥，false 不互斥
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setCheckable(bool flag)	
描述	设置KButtonBox中的按钮是否可以选中	
参数	参数名称	参数说明
	布尔值	true 可选中，false 不可选中
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isCheckable()	
描述	获取KButtonBox中的按钮是否可以选中	

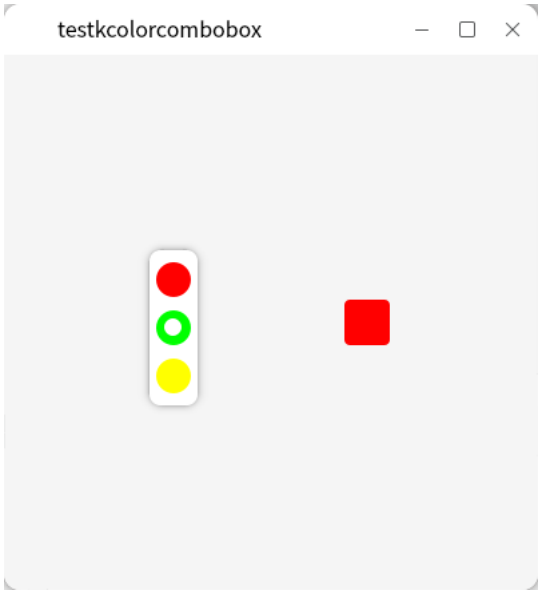
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 可选中，false 不可选中
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setShadow(bool flag)	
描述	设置是否启动阴影效果（自2.4启用）	
参数	参数名称	参数说明
	布尔值	true 启动阴影效果，false 未启动阴影效果
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool shadow()	
描述	获取是否启用阴影效果（自2.4启用）	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 启动阴影效果，false 未启动阴影效果
备注	无	

4.1.8.3 KColorComboBox

> 功能描述：可以选择容器中的任何颜色



枚举类型	enum ComboType{ Circle，RoundedRect};

子模块	libkysdk-qtwidgets
接口类型	C++

原型	void setColorList(const QList& list)	
描述	以列表形式添加颜色选项	
参数	参数名称	参数说明
	list	以列表形式添加颜色选项
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QList colorList()	
描述	获取颜色列表	
参数	参数名称	参数说明
	无	无
返回值	QList	获取颜色列表
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void addColor(const QColor& color)	
描述	添加颜色列表	
参数	参数名称	参数说明
	color	添加颜色列表
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setComboType(const ComboType& type)	
描述	设置显示样式	
参数	参数名称	参数说明
	type	设置显示样式
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	ComboTyle comboType()	
描述	获取显示样式	
参数	参数名称	参数说明
	无	无
返回值	ComboTyle	获取显示样式
备注	无	

子模块	libkysdk-qtwidgets

接口类型	C++	
原型	void setPopupItemSize(const QSize &size)	
描述	设置item尺寸	
参数	参数名称	参数说明
	size	设置item尺寸
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QSize popupItemSzie()	
描述	返回item尺寸	
参数	参数名称	参数说明
	无	无
返回值	QSize	返回item尺寸
备注	无	

4.1.9 面包屑 KBreadCrumb

> 无边框的标签栏，可以添加文字和图标，支持点击和 hover 高亮。
>
> 功能图：

> 枚举类型

枚举类型	enum KBreadCrumbType { FlatBreadCrumb, CubeBreadCrumb };

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIcon(const QIcon &icon);	
描述	设置图标	
参数	参数名称	参数说明
	icon	需要设置给部件的icon
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets

接口类型	C++	
原型	QIcon icon() const;	
描述	返回图标	
参数	参数名称	参数说明
	无	无
返回值	QIcon	获取控件的图标
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool isFlat() const;	
描述	获取KBreadCrumb是否为flat类型。	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true flat类型，false 不是flat类型
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setFlat(bool flat);	
描述	设置KBreadCrumb是否为flat类型。	
参数	参数名称	参数说明
	布尔值	true flag类型，false 不是flat类型
返回值	无	无
备注	无	

4.1.10 KCommentPanel

> 枚举类型

枚举类型	enum StarLevel {LevelZero =0, LevelOne, LevelTwo, LevelThree, LevelFour,LevelFive};

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIcon(const QIcon&);	
描述	设置图标	
参数	参数名称	参数说明
	QIcon	设置图标
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets

接口类型	C++	
原型	void setTime(const QString&);	
描述	设置评论时间	
参数	参数名称	参数说明
	QString	设置评论的时间
返回值	无	无
备注	无	

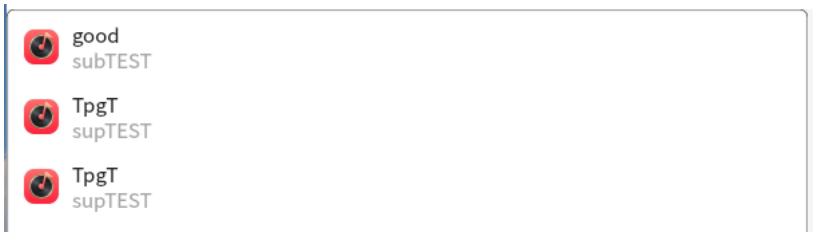
子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setName(const QString&);	
描述	设置评论名称	
参数	参数名称	参数说明
	QString	设置评论者的名称
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setText(const QString&);	
描述	设置评论内容	
参数	参数名称	参数说明
	QString	设置评论的内容
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setGrade(StarLevel level);	
描述	设置评论等级	
参数	参数名称	参数说明
	level	设置评论的等级
返回值	无	无
备注	无	

4.1.11 KListView

> 提供一个图片两行文字的显示效果，若只有一行文字，则对于 icon 居中显示 。



4.1.12 标签模块

4.1.12.1 KTag

标签，分为默认和可关闭的两种。

枚举类型

枚举类型	enum TagStyle { HighlightTag, BoderTag, BaseBoderTag, GrayTag, IconTag(since 2.4) };

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setClosable(bool flag)	
描述	设置标签是否可以关闭。	
参数	参数名称	参数说明
	布尔值	true 显示关闭按钮，可关闭，false 隐藏关闭按钮，不可关闭
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool closable()	
描述	获取标签是否可以关闭。	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 显示关闭按钮，可关闭，false 隐藏关闭按钮，不可关闭
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setText(const QString &text)	
描述	设置标签的文本。	
参数	参数名称	参数说明
	text	设置文本

返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTagStyle(TagStyle style)	
描述	设置标签的样式。	
参数	参数名称	参数说明
	style	设置tag的样式
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	TagStyle tagStyle()	
描述	获取标签的样式。	
参数	参数名称	参数说明
	无	无
返回值	TagStyle	获取标签的样式。
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QString text()	
描述	获取标签的文本。	
参数	参数名称	参数说明
	无	无
返回值	QString	获取标签的文本。
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setTranslucent(bool flag)	
描述	设置半透明 since 2.4	
参数	参数名称	参数说明
	布尔值	true 启用半透明效果。false 禁用半透明效果
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool translucent()	
描述	获取是否启用半透明 since 2.4	

参数	参数名称	参数说明
	无	无
返回值	布尔值	true 启用半透明，fasle 禁用半透明
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBackgroundColor(QColor color)	
描述	设置背景颜色	
参数	参数名称	参数说明
	color	设置控件的背景颜色
返回值	无	无
备注	无	

4.1.12.2 KLabel

功能描述：会显示省略文本的lable以及存在一些特定样式，自2.4启用

枚举类型

枚举类型	enum KLabelType {NormalType,DataType};

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setLabelType(KLabelType type = KLabelType::DataType)	
描述	设置label类型	
参数	参数名称	参数说明
	type	设置label的类型
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setDateHightColor（bool flag）	
描述	设置是否启用文本/图标高亮色	
参数	参数名称	参数说明
	布尔值	true 启用高亮色，false 禁用高亮色
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setPixmap（const QPixmap &pixmap）	
描述	设置图标pixmap	
参数	参数名称	参数说明
	pixmap）	label设置pixmap
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	const QPixmap *pixmap() const	
描述	获取图标	
参数	参数名称	参数说明
	无	无
返回值	QPixmap	获取label的图标
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBackgroundColor(bool flag,QColor color = Qt::white)	
描述	是否启用自定义背景并设置自定义背景色	
参数	参数名称	参数说明
	布尔值	true 启用自定义背景色， false 禁用自定义背景色
	color	自定义背景色的色值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius(int bottomLeft,int topLeft,int topRight,int bottomRight);	
描述	设置圆角	
参数	参数名称	参数说明
	bottomLeft	左下角圆角半径
	topLeft	左上角圆角半径
	topRight	右上角圆角半径
	bottomRight	右下角圆角半径
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets

接口类型	C++	
原型	void setBorderRadius(int radius);	
描述	设置圆角	
参数	参数名称	参数说明
	radius	设置圆角半径
返回值	无	无
备注	无	

4.1.13 KTranslucentFloor

> KTranslucentFloor，继承自 QFrame，提供一个毛玻璃底板，可以设置圆角以及是否添加阴影效果。自 1.2.0.12 版本启用。

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadious(int radious);	
描述	设置圆角半径	
参数	参数名称	参数说明
	radious	设置圆角半径
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	int borderRadius();	
描述	返回圆角半径	
参数	参数名称	参数说明
	无	无
返回值	int	获取圆角半径
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setShadow(bool flag);	
描述	设置是否显示阴影	
参数	参数名称	参数说明
	布尔值	true 启用阴影，false

		禁用阴影
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool shadow();	
描述	获取是否显示阴影	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 启用阴影，false 禁用阴影
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setEnableBlur(bool flag);	
描述	设置是否启用毛玻璃效果（自2.0.0.0启用）	
参数	参数名称	参数说明
	布尔值	true 启用毛玻璃效果，禁用毛玻璃效果
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	bool enableBlur();	
描述	获取是否已启用毛玻璃效果（自2.0.0.0启用）	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 启用毛玻璃效果，禁用毛玻璃效果
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setOpacity(qreal opacity);	
描述	设置透明度（自2.0.0.0启用）	
参数	参数名称	参数说明
	opacity	设置控件透明度的值
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	

原型	qreal opacity();	
描述	获取透明度（自2.0.0.0启用)	
参数	参数名称	参数说明
	无	无
返回值	qreal	获取当前透明度的值
备注	无	

4.1.14 KDragWidget

> KDragWidget,继承自QWidget，可以获取拖动进入控件或者选中的文件夹或文件的路径，提供获取设置图片的按钮，获取设置文本的label，获取filedialog等接口,自2.3.0.0启用

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	KPushbutton *iconButton()；	
描述	返回设置图标的button,自2.3.0.0启用	
参数	参数名称	参数说明
	无	无
返回值	KPushbutton*	获取设置图标的button
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	QLabel *textlabel();	
描述	返回设置文本的label,自2.3.0.0启用	
参数	参数名称	参数说明
	无	无
返回值	QLabel*	返回设置文本的label
备注	无	

子模块	libkysdk-qtwidgets

接口类型	C++	
原型	QFileDialog *fileDialog()	
描述	返回打开的文件对话框 ,自2.3.0.0启用	
参数	参数名称	参数说明
	无	无
返回值	QFileDialog*	返回打开的文件对话框
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setNameFilter(const QString &filter)	
描述	将文件对话框中使用的过滤器设置为给定的过滤器,自2.3.0.0启用	
参数	参数名称	参数说明
	filter	将文件对话框中使用的过滤器设置为给定的过滤器
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setNameFilters(const QString &filters)	
描述	设置文件对话框中使用的过滤器,自2.3.0.0启用	
参数	参数名称	参数说明
	filters	设置文件对话框中使用的过滤器
返回值	无	无
备注	无	

4.1.15 KFileWidget

图片+两行文本的组合（自2.4启用）

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setMainText(QString str)	
描述	设置主文本	
参数	参数名称	参数说明
	str	设置主文本
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setSubText（QString str）	
描述	设置次文本	
参数	参数名称	参数说明
	str	设置次级文本
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setIcon（QIcon icon）	
描述	设置图片	
参数	参数名称	参数说明
	icon	设置图片
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius（int radius）	
描述	获取左边栏widget	
参数	参数名称	参数说明
	无	无
返回值	无	无
备注	无	

子模块	libkysdk-qtwidgets	
接口类型	C++	
原型	void setBorderRadius(int bottomLeft,int topLeft,int topRight,int bottomRight);	
描述	设置圆角	
参数	参数名称	参数说明
	bottomLeft	左下角圆角半径
	topLeft	左上角圆角半径
	topRight	右上角圆角半径
	bottomRight	右下角圆角半径
返回值	无	无
备注	无	

4.2 Wayland-helper

该模块主要负责提供在 X 平台和 Wayland 平台下均能生效的兼容接口，使应用无需考虑显示平台的差异。该模块安装方式如下：

```
sudo apt install libkysdk-waylandhelper libkysdk-waylandhelper-dev
```

根据不同项目类型，可参考以下 demo 构建项目：
（1） .pro 文件构建项目：
qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-waylandhelper
```

（2） CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(Qt5 COMPONENTS Widgets REQUIRED)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKWAYLANDHELPER_PKG kysdk-waylandhelper)
target_include_directories(demo PRIVATE ${KYSDKWAYLANDHELPER_PKG_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKWAYLANDHELPER_PKG_LIBRARY_DIRS})
target_link_libraries(demo Qt5::Widgets ${KYSDKWAYLANDHELPER_PKG_LIBRARIES})
```

4.2.1 WindowManager

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static WindowManager* self();	
描述	获取单例对象	
参数	参数名称	参数说明
	无	无
返回值	WindowManager*	获取单例对象
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static WindowInfo getwindowInfo(const WindowId& windowId);	
描述	获取窗口信息	
参数	参数名称	参数说明
	windowId	传入window id
返回值	WindowInfo	通过winid返回windowinfo
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static WindowId currentActiveWindow();	
描述	获取当前活动窗口	
参数	参数名称	参数说明
	无	无
返回值	WindowId	获取当前窗口的windowid
备注	无	

子模块	libkysdk-waylandhelper

接口类型	C++	
原型	static void keepWindowAbove(const WindowId& windowId);	
描述	置顶窗口	
参数	参数名称	参数说明
	windowId	传入window id
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static QString getWindowTitle(const WindowId& windowId);	
描述	获取窗口标题	
参数	参数名称	参数说明
	windowId	传入window id
返回值	QString	通过window id 获取窗口标题
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static QIcon getWindowIcon(const WindowId& windowId);	
描述	获取窗口图标	
参数	参数名称	参数说明
	获取窗口图标	传入windowid
返回值	QIcon	获取传入windowid的窗口图标
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static QString getWindowGroup(const WindowId& windowId);	
描述	获取窗口所在组的组名	
参数	参数名称	参数说明
	windowId	传入的window id
返回值	QString无	获取传入window id所在组的组名
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static void closeWindow(const WindowId& windowId);	
描述	关闭窗口	
参数	参数名称	参数说明
	windowId	传入window id

返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static void activateWindow(const WindowId& windowId);	
描述	激活窗口	
参数	参数名称	参数说明
	windowId	传入window id
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static void maximizeWindow(const WindowId& windowId);	
描述	最大化窗口	
参数	参数名称	参数说明
	windowId	传入的window id
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static void minimizeWindow(const WindowId& windowId);	
描述	最小化窗口	
参数	参数名称	参数说明
	windowId	传入的window id
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static quint32 getPid(const WindowId& windowId);	
描述	获取窗口进程pid	
参数	参数名称	参数说明
	windowId	传入的window id
返回值	quint32	获取到对应窗口进程的pid
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static void showDesktop();	
描述	显示当前桌面	
参数	参数名称	参数说明

	无	无
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static void hideDesktop();	
描述	取消显示当前桌面	
参数	参数名称	参数说明
	无	无
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static QString currentDesktop();	
描述	获取当前桌面的名称	
参数	参数名称	参数说明
	无	无
返回值	QString	获取当前桌面的名称
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static QList windows();	
描述	获取当前窗口列表	
参数	参数名称	参数说明
	无	无
返回值	QList	获取当前窗口列表
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static NET::WindowType getWindowType(const WindowId& windowId);	
描述	获取窗口类型，仅适用于X环境下， wayland下统一返回normal	
参数	参数名称	参数说明
	windowId	传入的window id
返回值	NET::WindowType	获取窗口类型， 仅适用于X环境下， wayland下统一返回normal
备注	无	

子模块	libkysdk-waylandhelper	

接口类型	C++	
原型	static void setGeometry(QWindow *window,const QRect &rect);	
描述	设置窗口位置	
参数	参数名称	参数说明
	window	传入的window id
	rect	需要将窗体移动到的位置rect
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static void setSkipTaskBar(QWindow *window,bool skip);	
描述	设置是否跳过任务栏（自2.0.0.0启用）	
参数	参数名称	参数说明
	window	要跳过任务栏的窗口
	布尔值	true 跳过任务兰，false 不跳过任务栏
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static void setSkipSwitcher(QWindow *window,bool skip);	
描述	设置是否跳过窗口选择（自2.0.0.0启用）	
参数	参数名称	参数说明
	window	要跳过窗口选择的窗口
	布尔值	true 跳过窗口选择，fasle 不跳过窗口选择
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static bool skipTaskBar(const WindowId& windowId);	
描述	获取窗体是否跳过任务栏（自2.0.0.0启用）	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 跳过任务兰，false 不跳过任务栏
备注	无	

子模块	libkysdk-waylandhelper
接口类型	C++
原型	static bool skipSwitcher(const WindowId& windowId);

描述	获取窗体是否跳过窗口选择（自2.0.0.0启用）	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 跳过窗口选择，fasle 不跳过窗口选择无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static bool isShowingDesktop();	
描述	获取桌面是否处于显示状态（自2.0.0.0启用）	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 出于显示状态，false 处于隐藏状态
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static void setOnAllDesktops(const WindowId &windowId);	
描述	设置窗口在所有桌面中显示（自2.0.0.0启用）	
参数	参数名称	参数说明
	windowId	显示在所有桌面窗口的window id
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static bool isOnAllDesktops(const WindowId &windowId);	
描述	获取窗口是否在所有桌面中显示（自2.0.0.0启用）	
参数	参数名称	参数说明
	windowId	传入窗口的window id
返回值	布尔值	true 当前窗口在所有桌面中显示， false 当前窗口不在所有桌面中显示
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static bool isOnCurrentDesktop(const WindowId& id);	
描述	获取窗口是否在当前桌面(自2.1启用)	
参数	参数名称	参数说明
	id	传入窗口的window id

返回值	布尔值	true 窗口在当前桌面，false 窗口不在当前桌面
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static bool isOnDesktop(const WindowId &id , int desktop);	
描述	获取窗口是否在指定桌面(自2.1启用)	
参数	参数名称	参数说明
	id	窗口的window id
	desktop	指定桌面
返回值	布尔值	true 窗口在指定桌面，false 窗口不在指定桌面
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static void setPanelTakefocus(QWindow *window ,bool flag)	
描述	设置panel属性窗体是否获取焦点，仅在wayland环境下生效(自2.3启用)	
参数	参数名称	参数说明
	window	传入的窗口 window
布尔值	true 获取焦点，false 未获取焦点	
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static void demandAttention(const WindowId &wid)	
描述	发送demanAttention状态(自2.3启用)	
参数	参数名称	参数说明
	wid	传入的window id
返回值	无	无
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static QString getProcessName(const WindowId& windowId)	
描述	获取窗口所属的进程名称(自2.3启用)	
参数	参数名称	参数说明
	windowId	传入的window id
返回值	QString	获取窗口所属的进程名称
备注	无	

4.2.2 WindowInfo

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	inline bool isMaximized() const;	
描述	返回窗口是否是最大化状态	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 窗口在最大化状态， false 窗口不在最大化状态
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	inline bool isMinimized() const;	
描述	返回窗口是否是最小化状态	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 窗口在最小化状态， false 窗口不在最小化状态
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	inline bool isMaximizable() const;	
描述	返回窗口是否可以最大化	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 窗口可最大化，false 窗口不可最大化
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	inline bool isMinimizable() const;	
描述	返回窗口是否可以最小化	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 窗口可最小化，false 窗口不可最小化
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	inline bool isKeepAbove() const;	
描述	返回窗口是否是置顶状态	
参数	参数名称	参数说明

	无	无
返回值	布尔值	true 置顶状态，false 不在置顶状态
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	inline bool hasSkipTaskbar() const;	
描述	返回窗口是否跳过任务栏	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 窗口跳过任务栏，false 窗口未跳过任务栏
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	inline bool isFullscreen() const;	
描述	返回窗口是否是全屏状态	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 窗口在全屏状态，false 窗口不在全屏状态
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	inline bool isOnAllDesktops() const； noexcept;	
描述	返回窗口是否在所有桌面中显示	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 窗口在所有桌面中显示， false 窗口不在所有桌面中显示
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	inline bool isValid() const;	
描述	判断窗口id是否有效	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 窗口id有效，false 窗口id无效
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	inline bool isActive() const noexcept;	
描述	返回窗口是否是激活状态	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 窗口是激活状态，false 窗口不是激活状态
备注	无	

4.2.3 UkuiStyleHelper

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	static UkuiStyleHelper *self();	
描述	获取单例对象	
参数	参数名称	参数说明
	无	无
返回值	UkuiStyleHelper*	获取单例对象
备注	无	

子模块	libkysdk-waylandhelper	
接口类型	C++	
原型	void removeHeader(QWidget* widget);	
描述	移除窗管标题栏	
参数	参数名称	参数说明
	widget	将给定widget的窗口标题栏移除
返回值	无	无
备注	无	

4.3 应用通用功能模块

4.3.1 日志模块

子模块	日志模块	
接口类型	C++	
原型	static void logOutput(QtMsgType type , const QMessageLogContext &context , const QString &msg);	
描述	用于Qt注册日志函数，不应单独调用.	
参数	参数名称	参数说明
	type	日志类型
	context	调用打印日志接口文件信息
	msg	日志信息
返回值	无	无

备注	无

4.3.2 系统相关模块

子模块	窗管模块	
接口类型	C++	
原型	static bool setWindowMotifHint(int winId);	
描述	添加窗管协议	
参数	参数名称	参数说明
	winId	窗口 id
返回值	布尔值	true 成功，false 失败
备注	无	

子模块	窗管模块	
接口类型	C++	
原型	static bool setScalingProperties(void);	
描述	设置窗口缩放属性	
参数	参数名称	参数说明
	无	无
返回值	布尔值	true 成功，false 失败
备注	无	

子模块	session模块	
接口类型	C++	
原型	quint32 setInhibitLockScreen(AppName appName , QString reason);	
描述	禁止系统锁屏	
参数	参数名称	参数说明
	appName	应用名
	reason	禁止锁屏的原因
返回值	布尔值	成功 : 非 0 的正整数，失败 : 0
备注	无	

子模块	session模块	
接口类型	C++	
原型	bool unInhibitLockScreen(quint32 flag)	
描述	取消禁止系统锁屏	
参数	参数名称	参数说明
	flag	禁止锁屏标识
返回值	布尔值	true 成功，false 失败
备注	无	

4.3.3 d-bus 模块------即将废弃

子模块	d-bus模块

接口类型	C++	
原型	static QList callMethod(QString serviceName, QString objectPath, QString interfaceName, QString methodName, QList args = QList());	
描述	获取左边栏widget	
参数	参数名称	参数说明
	serviceName	服务名
	objectPath	对象路径
	interfaceName	接口名
	methodName	函数名
	args	参数列表
返回值	QList	函数返回值
备注	无	

4.3.4 系统信息模块

子模块	系统信息模块	
接口类型	C++	
原型	QString getLsbReleaseInformation(QString key);	
描述	根据 lsb-release 文件的 key 值 获取信息	
参数	参数名称	参数说明
	key	键值
返回值	QString	获取到的信息， 为空时可能确实为空也可能失败
备注	无	

子模块	系统信息模块	
接口类型	C++	
原型	QString getOsReleaseInformation(QString key);	
描述	根据 os-release 文件的 key 值 获取信息	
参数	参数名称	参数说明
	key	键值
返回值	QString	获取到的信息， 为空时可能确实为空也可能失败
备注	无	

子模块	系统信息模块	
接口类型	C++	
原型	QString getProjectCodeName(void)	
描述	获取 PROJECT_CODENAME 字段的值	
参数	参数名称	参数说明
	无	无
返回值	QString	非空：获取到的值，空：失败
备注	无	

子模块	系统信息模块	
接口类型	C++	
原型	QString getCpuModelName(void)	
描述	获取 CPU 型号	
参数	参数名称	参数说明
	无	无
返回值	QString	非空 : 获取到的值 , 空 : 失败
备注	无	

子模块	系统信息模块	
接口类型	C++	
原型	QString getHdPlatform(void);	
描述	获取硬件平台信息	
参数	参数名称	参数说明
	无	无
返回值	QString	非空 : 获取到的值 , 空 : 失败
备注	无	

5 基础开发 SDK

该层设计主要为应用开发提供与操作系统无关的、高通用性、基础性的功能集合。减少不同应用在实现相同功能式的差异性和复杂性。该层设计主要为应用开发提供与操作系统无关的、高通用性、基础性的功能集合。减少不同应用在实现相同功能式的差异性和复杂性。

- **安装命令**

```
$ sudo apt install libkysdk-base libkysdk-base-dev
```

5.1.1 日志记录功能

- **引入头文件**

```
#include "kysdk/kysdk-base/libkylog.h"
```

- **库文件路径**

`/usr/lib/*/libkylog.so`

'*'代表不同架构的架构目录名称，例如x86_64-linux-gnu

- **子模块信息**

> 输出日志(自1.2.0版本启用)

子模块	日志模块
接口类型	C
原型	#define klog_debug(fmt , ...) #define klog_info(fmt , ...) #define klog_notice(fmt , ...) #define klog_warning(fmt , ...) #define klog_err(fmt , ...) #define klog_crit(fmt , ...) #define klog_alert(fmt , ...) #define klog_emerg(fmt , ...)

描述	输出不同等级的日志，宏名即日志等级	
参数	fmt	格式化字符串
返回值	无	
备注	无	

初始化日志记录(自1.2.0版本启用)

子模块	日志模块	
接口类型	C	
原型	int kdk_logger_init(const char *ini)	
描述	初始化日志记录， 也可以不调用该函数直接使用上方日志记录的宏， 若以此方式运行，则程序会使用默认的日志配置文件	
参数	ini	日志配置文件路径， 若传入NULL则会使用默认的日志配置文件
返回值	0	成功
	int(非0)	失败
备注	无	

缓存区日志写入(自1.2.0版本启用)

子模块	日志模块
接口类型	C
原型	void kdk_logger_flush()
描述	在异步写入的方式下，可以调用该函数手动将缓存区中的日志写入文件中
参数	无
返回值	无
备注	无

设置日志输出自动换行(自1.2.0版本启用)

子模块	日志模块	
接口类型	C	
原型	void kdk_logger_set_autowrap(int autowarp)	
描述	设置日志输出自动换行	
参数	autowarp	1 启用 0 禁用
返回值	无	
备注	无	

5.1.2 日志转储功能

本模块提供日志转储功能，安装此包则可以转存，未安装不可以转存。

- **安装命令**

```
$ sudo apt install libkysdk-logrotate
```

- **子模块信息**

是否转存：是否安装libkysdk-logrotate包（此包在system层）；安装此包则可以转存，未安装不可以转存。

转存规则：

1. 日志文件按天轮训；
2. 一次将存储 5 个归档日志，对于第六个归档，时间最久的归档将被删除。
3. 在轮循任务完成后，已轮循的归档将使用 gzip 进行压缩。
4. 归档文件文件以创建日期命名；如xxx.log-20131216。
5. 在日志轮循期间，任何错误将被忽略，例如 “文件无法找到” 之类的错误。
6. 如果日志文件为空，轮循不会进行
7. 还在打开中的日志文件，把当前日志备份并截断。

转存目录：非 root 程序记录在~/.log目录中，root 程序记录在/var/log 下。

转存后压缩文件名称：程序名.log-打包日期.gz；例如：kylin-photo-viewer.log-20240220.gz

5.2 定时器

C 语言定时器模块，给 C/C++程序提供定时器功能接口。

- **安装命令**

```
$ sudo apt-get install libkysdk-timer libkysdk-timer-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-timer
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKTIMER kysdk-timer)target_include_directo
```

5.2.1 定时器功能

该模块提供定时器功能

- **引入头文件**

```
#include "kysdk/kysdk-base/libkytimer.h"
```

- **库文件路径**

`/usr/lib/*/libkytimer.so`

- **子模块信息**

> 初始化定时器(自1.2.0版本启用)

子模块	定时器	
接口类型	C	
原型	int kdk_timer_init()	
描述	初始化定时器核心组件	
参数	无	
返回值	0	成功
	int(非0)	失败
备注	无	

> 启动定时器(自1.2.0版本启用)

子模块	定时器

接口类型	C	
原型	size_t kdk_timer_start(unsigned int intervalms, time_handler callback, KTimerAttribute attr, KTimerType type, void* userdata, int freeOnDelete)	
描述	启动定时器	
参数	intervalms	定时器时间，以毫秒为单位
	callback	定时器到期后触发的回调函数指针
	attr	定时器属性 KTIMER_SINGLESHOT表示一次性定时器 KTIMER_PERIODIC表示周期性定时器 KTIMER_NEVER表示不会被触发的定时器
	type	定时器类型 KTIMER_ABSOLUTE表示绝对时间定时器，修改系统时间不会影响定时器的时间 KTIMER_RELATIVE表示相对时间定时器，修改系统时间会影响定时器时间
	userdata	指向用户数据的指针
	freeOnDelete	[未启用]
返回值	size_t	定时器的ID
备注	无	

停止给定的定时器(自1.2.0版本启用)

子模块	定时器	
接口类型	C	
原型	void kdk_timer_stop(size_t timerfd)	
描述	停止给定的定时器	
参数	timerfd	由kdk_timer_start返回的定时器ID
返回值	无	
备注	无	

销毁定时器(自1.2.0版本启用)

子模块	定时器
接口类型	C
原型	void kdk_timer_destroy()
描述	销毁定时器
参数	无
返回值	无
备注	无

重置定时器(自1.2.0版本启用)

子模块	定时器	
接口类型	C	
原型	void kdk_timer_reset(size_t timerfd , unsigned int intervalms)	
描述	重置定时器	
参数	timerfd	由kdk_timer_start返回的定时器ID
	intervalms	需要调整的时间间隔， 以ms为单位

返回值	无
备注	无

5.3 常用工具模块

该模块封装了对字符串操作的 C 接口，包括字符串裁剪、分割、大小写转换、查找特定字符等操作。和数据结果例如调表的相关接口

- **安装命令**

```
$ sudo apt-get install libkysdk-utils libkysdk-utils-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-utils
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKUTILS kysdk-utils)target_include_directo
```

5.3.1 C字符串功能扩展

该模块扩展c字符串操作函数

- **引入头文件**

```
#include "kysdk/kysdk-base/cstring-extension.h"
```

- **库文件路径**

`/usr/lib/*/libkyutils.so`

- **子模块信息**

> 分割字符串(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline char** strsplit(char *str, char delim)	
描述	对原字符串以给定的分隔符进行分割， 注意该函数会修改原字符串	
参数	str	需要分割的字符串指针
	delim	分隔符
返回值	char**	分割后的字符串列表， 以NULL结尾
备注	存储分割后所有字符串的字符串列表本身是由alloc申请的内存， 因此当使用完成后应当被free； 而分割出来的各个字符串不是申请的内存， 而是分别指向了原字符串中的特定位置， 因此他们不需要被分别free	

> 小写字符串(自1.2.0版本启用)

子模块	C语言字符串扩展
接口类型	C
原型	static inline void str2lower(char *str)

描述	将字符串中的所有大写字母转化为小写字母	
参数	str	需要操作的字符串指针
返回值	无	
备注	无	

大写字符串(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline void str2upper(char *str)	
描述	将字符串中的所有小写字母转化为大写字母	
参数	str	需要操作的字符串指针
返回值	无	
备注	无	

统计出现次数(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline size_t strcounts(const char *str, char ch)	
描述	统计给定字符在字符串中出现的次数	
参数	str	原字符串
	ch	需要统计的字符
返回值	size_t	字符出现的次数
备注	无	

后缀判断(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline int strendwith(const char *str, const char *postfix)	
描述	判断str是否以postfix结尾需要区分大小写	
参数	str	原字符串
	postfix	需要匹配的字符串后缀
返回值	0	str以postfix结尾
	1	str不以postfix结尾
备注	无	

查找子字符串的首次出现位置(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline int strfirstof(const char* str, char ch)	
描述	在给定的字符串中查找给定字符第一次出现的位置； 计数从0开始	
参数	str	原字符串
	ch	需要查找的字符

返回值	int	第一次出现的位置， 若未找到给定的字符， 则返回-1
备注	无	

查找子字符串的末次出现位置(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline int strlastof(const char* str, char ch)	
描述	在给定的字符串中查找给定字符最后一次出现的位置； 计数从0开始	
参数	str	原字符串
	ch	需要查找的字符
返回值	int	最后一次出现的位置， 若未找到给定的字符， 则返回-1
备注	无	

删除空格和制表符(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline void strstripblank(char *str)	
描述	删除给定字符串前后的空格和水平制表符（tab）， 注意该操作会修改原字符串	
参数	str	需要修改的字符串指针
返回值	无	
备注	无	

跳过开始的所有空格、制表符、换行符(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline const char *strskipspace(const char *p)	
描述	跳过字符串前的所有空格、制表符、换行符； 该操作不会修改原字符串	
参数	p	指向原字符串的指针
返回值	const char*	指向跳过space后的字符串指针
备注	无	

前缀判断（区分大小写）(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline int strstartswith(const char *str, const char *prefix)	
描述	判断str是否以prefix开头；区分大小写	
参数	str	原字符串
	prefix	需要匹配的字符串前缀

返回值	0	str是以prefix开头
	1	str不是以prefix开头
备注	无	

前缀判断（不区分大小写）(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline int strstartswith_nocase(const char *str, const char *prefix)	
描述	判断str是否以prefix开头；不区分大小写	
参数	str	原字符串
	prefix	需要匹配的字符串前缀
返回值	0	str是以prefix开头
	1	str不是以prefix开头
备注	无	

删减字符串前后的指定字符(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline void strstrip(char *str, char ch)	
描述	对给定的字符串进行strip操作， 删减字符串前后的指定字符；注意该操作会修改原字符串	
参数	str	需要进行strip的字符串指针
	ch	需要删除的字符
返回值	无	
备注	无	

跳过开始的所有空格、水平制表符(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline const char *strskipblank(const char *p)	
描述	跳过字符串前的所有空格和水平制表符（tab）； 该操作不会修改原字符串	
参数	p	指向原字符串的指针
返回值	const char*	指向跳过space后的字符串指针
备注	无	

删除前后的空格、制表符、换行符(自1.2.0版本启用)

子模块	C语言字符串扩展	
接口类型	C	
原型	static inline void strstripspace(char *str)	
描述	删除给定字符串前后的空格、制表符、换行符， 注意该操作会修改原字符串	
参数	str	需要进行strip操作的字符串指针

返回值	无
备注	无

5.3.2 数据结构模块

该模块实现了 C 语言链表操作相关接口。

- **引入头文件**

```
#include "kysdk/kysdk-base/skip_linklist.h"
```

- **库文件路径**

`/usr/lib/*/libkydatastruct.so`

- **子模块信息**

> 创建跳表(自1.2.0版本启用)

子模块	数据结构模块	
接口类型	C	
原型	kysdk_skiplist* kysdk_create_skiplist()	
描述	创建跳表	
参数	无	
返回值	kysdk_skiplist *	跳表
备注	无	

> 销毁跳表(自1.2.0版本启用)

子模块	数据结构模块	
接口类型	C	
原型	void kysdk_destroy_skiplist(kysdk_skiplist *list)	
描述	销毁跳表，并回收所有分配的内存；注意data.ptr指向的内存（若存在）不会被释放	
参数	list	需要操作的跳表
返回值	无	
备注	无	

> 删除 key 值对应的节点(自1.2.0版本启用)

子模块	数据结构模块	
接口类型	C	
原型	int kysdk_skiplist_delete(kysdk_skiplist *list, int key)	
描述	删除key值对应的节点	
参数	list	需要操作的跳表
	key	键
返回值	0	成功
	-1	失败
备注	无	

> 插入节点(自1.2.0版本启用)

子模块	数据结构模块

接口类型	C	
原型	int kysdk_skiplist_insert(kysdk_skiplist *list, int key, kysdk_listdata data)	
描述	插入节点	
参数	list	需要操作的跳表
	key	键
	data	数据
返回值	0	成功
	-1	失败
备注	无	

根据给定的 key 搜索 data 内容(自1.2.0版本启用)

子模块	数据结构模块	
接口类型	C	
原型	kysdk_listdata kysdk_skiplist_search(kysdk_skiplist *list, int key)	
描述	根据给定的key搜索data内容	
参数	list	需要操作的跳表
	key	键
返回值	kysdk_listdata	节点数据, 当搜索的key不存在时， data.num值为-1
备注	无	

设置跳表最高层数(自1.2.0版本启用)

子模块	数据结构模块	
接口类型	C	
原型	int kysdk_skiplist_setmaxlevels(kysdk_skiplist *list, unsigned int maxlevels)	
描述	设置跳表最高层数，该选项必须在跳表为空时使用	
参数	list	需要操作的跳表
	maxlevels	层数
返回值	0	成功
	-1	失败
备注	无	

5.3.3 单位进制转换

- **引入头文件**

```
#include "/usr/include/kysdk/kysdk-base/libkyutils.h"
```

- **库文件路径**

`/usr/lib/*/libkyutils.so`

- **子模块信息**

字符格式单位进制转换(自1.2.0版本启用)

子模块	单位进制转换	
接口类型	C	
原型	int kdkVolumeBaseCharacterConvert(const char* origin_data, KDKVolumeBaseType result_base, char* result_data)	
描述	数字格式单位进制转换	
入参	origin_data	原始数字类型数据
	result_base	原始的进制单位
出参	result_data	期望的进制单位
返回值	0	成功
	1	非法进制
	2	非法参数
	3	非法数据格式
	4	系统运行异常引发的未知错误
备注	无	

> 数字格式单位进制转换(自1.2.0版本启用)

子模块	单位进制转换	
接口类型	C	
原型	int kdkVolumeBaseNumericalConvert(double origin_numerical, KDKVolumeBaseType origin_base, KDKVolumeBaseType result_base, double* result_numerical)	
描述	数字格式单位进制转换	
入数	origin_numerical	原始数字类型数据
	origin_base	原始的进制单位
出数	result_base	期望的进制单位
	result_numerical	期望进制下的数字类型数据
返回值	0	成功
	1	非法进制
	2	非法参数
	3	非法数据格式
	4	系统运行异常引发的未知错误
备注	无	

5.4 配置文件操作

KYSDK 配置文件处理库，支持标准格式、XML(未实现)、JSON(未实现)的配置文件处理，包括配置文件的读与写操作。

- **安装命令**

```
$ sudo apt-get install libkysdk-config libkysdk-config-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-config
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKCONFIG kysdk-config)target_include_direc
```

5.4.1 配置文件操作功能

- **引入头文件**

```
#include "kysdk/kysdk-base/libkyconf.h"
```

- **库文件路径**

`/usr/lib/*/libkyconf.so`

- **子模块信息**

初始化配置文件(自1.2.0版本启用)

子模块	配置文件操作	
接口类型	C	
原型	int kdk_conf_init(const char* confpath)	
描述	初始化配置文件	
参数	confpath	配置文件的路径
返回值	int(非负值)	成功
	int(负值)	错误码
备注	无	

销毁指定的配置文件句柄(自1.2.0版本启用)

子模块	配置文件操作	
接口类型	C	
原型	void kdk_conf_destroy(int id)	
描述	销毁指定的配置文件句柄	
参数	id	由kdk_conf_init返回的句柄值
返回值	无	
	备注	无

重新载入配置文件(自1.2.0版本启用)

子模块	配置文件操作	
接口类型	C	
原型	int kdk_conf_reload(int id)	
描述	重新载入配置文件	
参数	id	由kdk_conf_init返回的句柄值
返回值	0	成功
	int	错误码
备注	无	

获取指定配置项的值(自1.2.0版本启用)

子模块	配置文件操作	
接口类型	C	

原型	const char* kdk_conf_get_value(int id, const char* group, const char* key)	
描述	获取指定配置项的值	
参数	id	由kdk_conf_init返回的句柄值
	group	组名称
	key	配置项名称
返回值	const char*	配置项所拥有的值, 若key不存在， 则返回一个空字符串
备注	无	

枚举 key 值(自1.2.0版本启用)

子模块	配置文件操作	
接口类型	C	
原型	char** const kdk_conf_list_key(int id, const char* group)	
描述	列举id对应配置文件的指定Group下的key值， 结尾以NULL指针表示	
参数	id	由kdk_conf_init返回的句柄值
	group	需要列举的Group名称
返回值	const char** const	以NULL结尾的字符串列表， 每个字符串都是一个key名称
备注	字符串列表本身是由alloc分配的内存，需要被free释放； 字符串不需要释放	

枚举配置文件的 Group(自1.2.0版本启用)

子模块	配置文件操作	
接口类型	C	
原型	char** const kdk_conf_list_group(int id)	
描述	列举id对应配置文件的所有Group，结尾以NULL指针表示	
参数	id	由kdk_conf_init返回的句柄值
返回值	const char** const	以NULL结尾的字符串列表， 每个字符串都是一个组名称
备注	字符串列表本身是由alloc分配的内存，需要被free释放； 字符串不需要释放	

5.5 Gsettings配置

- **安装命令**

```
$ sudo apt-get install libkysdk-gsetting libkysdk-gsetting-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-gsetting
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKGSETTING kysdk-gsetting)target_include_d
```

5.5.1 GSettings配置操作

- **添加头文件**

```
#include "libkygsetting.h"
```

- **库文件路径**

`/usr/lib/*/libkygsettings.so`

- **子模块信息**

> 设置gesettings指定key值(自2.0.0.0版本启用)

子模块	GSettings配置操作	
接口类型	C	
原型	int kdk_gsettings_set(const char *schema_id, const char *key, const char *format, ...)	
描述	设置gesettings指定key值	
参数	schema_id	配置文件id
	key	配置文件key
	format	参数格式
返回值	0	false
	1	true
备注	无	

> 重置gesettings指定key值(自2.0.0.0版本启用)

子模块	GSettings配置操作	
接口类型	C	
原型	int kdk_settings_reset(const char *schema_id, const char *key)	
描述	重置gesettings指定key值	
参数	schema_id	配置文件id
	key	配置文件key
返回值	0	false
	1	true
备注	无	

> 设置gesettings指定string类型key值(自2.0.0.0版本启用)

子模块	GSettings配置操作	
接口类型	C	
原型	int kdk_settings_set_string(const char *schema_id, const char *key, const char *value)	
描述	设置gesettings指定string类型key值	
参数	schema_id	配置文件id
	key	配置文件key

	value	值
返回值	0	false
	1	true
备注	无	

设置gesettings指定int类型key值(自2.0.0.0版本启用)

子模块	GSettings配置操作	
接口类型	C	
原型	int kdk_settings_set_int(const char *schema_id, const char *key, int value)	
描述	设置gesettings指定int类型key值	
参数	schema_id	配置文件id
	key	配置文件key
	value	值
返回值	0	false
	1	true
备注	无	

获取gesettings指定key值(自2.0.0.0版本启用)

子模块	GSettings配置操作	
接口类型	C	
原型	void* kdk_gsettings_get(const char *schema_id, const char *key, const char *format, ...)	
描述	获取gesettings指定key值	
参数	schema_id	配置文件id
	key	配置文件key
	format	类型
返回值	void *	数据指针
备注	返回值需要释放	

获取gesettings指定string类型key值(自2.0.0.0版本启用)

子模块	GSettings配置操作	
接口类型	C	
原型	char* kdk_settings_get_string(const char *schema_id, const char *key)	
描述	获取gesettings指定string类型key值	
参数	schema_id	配置文件id
	key	配置文件key
返回值	char *	键值
备注	返回值需要释放	

获取gesettings指定int类型key值(自2.0.0.0版本启用)

子模块	GSettings配置操作
接口类型	C

原型	int kdk_settings_get_int(const char *schema_id, const char *key)	
描述	获取gesettings指定int类型key值	
参数	schema_id	配置文件id
	key	配置文件key
返回值	int	键值
备注	无	

> 获取gesettings指定double类型key值(自2.0.0.0版本启用)

子模块	GSettings配置操作	
接口类型	C	
原型	double kdk_settings_get_double(const char *schema_id, const char *key)	
描述	获取gesettings指定double类型key值	
参数	schema_id	配置文件id
	key	配置文件key
返回值	double	键值
备注	无	

5.6 埋点数据

该模块封装了上传埋点数据操作。

- **安装命令**

```
$ sudo apt-get install libkysdk-diagnostics libkysdk-diagnostics-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-diagnostics
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(KYSDKDIAGNOSTICS kysdk-diagnostics)target_inc
```

5.6.1 埋点数据功能

- **添加头文件**

```
#include "kysdk/kysdk-base/libkydiagnostics.h"
```

- **库文件路径**

`/usr/bin/*/libkydiagnostics.so`

- **子模块信息**

> 上传埋点数据(自1.2.0版本启用)

子模块	上传埋点数据
接口类型	C
原型	int kdk_buried_point(char *appName, char *messageType,

	KBuriedPoint *data, int length)	
描述	上传埋点数据	
参数	appName	埋点的包名
	messageType	消息类型
	data	需要埋点的数据数组
	length	数组长度
返回值	0	成功
	-1	失败
备注	无	

5.7 统一配置

该模块提供了统一配置读写功能

- **安装命令**

```
$ sudo apt-get install libkysdk-conf2 libkysdk-conf2-dev
```

- **构建示例**

（1）.pro 文件构建项目

```
CONFIG += link_pkgconfigPKGCONFIG += kysdk-conf2
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)find_package(PkgConfig REQUIRED)pkg_check_modules(CONF2 kysdk-conf2)target_include_directories(
```

5.7.1 统一配置模块

- **添加头文件**

```
#include "kysdk/kysdk-base/libkysettings.h"
```

- **库文件路径**

`/usr/lib/*/libkyconf2.so`

- **子模块信息**

> 获取配置句柄(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	KSettings *kdk_conf2_new(const char *id, const char *version)	
描述	获取指定id指定版本的句柄	
参数	id	包含了所有父组件名以'.'分割的字符串。不包括版本号
	version	版本号，传NULL使用默认版本
返回值	KSettings *	配置句柄
备注	无	

> 释放配置句柄(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	void kdk_conf2_ksettings_destroy(KSettings *ksettings)	
描述	释放句柄	
参数	ksettings	配置句柄
返回值	无	
备注	无	

获取配置句柄对应的配置id(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char *kdk_conf2_get_id(KSettings *ksettings)	
描述	获取配置句柄对应的配置id	
参数	ksettings	配置句柄
返回值	char *	句柄对应的id
备注	返回值需要释放	

获取配置句柄对应的配置版本(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char *kdk_conf2_get_version(KSettings *ksettings)	
描述	获取配置句柄对应的配置版本	
参数	ksettings	配置句柄
返回值	char *	句柄对应的配置版本
备注	返回值需要释放	

获取子组件句柄(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	KSettings *kdk_conf2_get_child(KSettings *ksettings, const char *name)	
描述	获取子组件句柄	
参数	ksettings	配置句柄
	name	子组件名
返回值	KSettings *	子组件句柄
备注	无	

获取所有子组件名(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char **kdk_conf2_list_children(KSettings *ksettings)	
描述	获取所有子组件名	
参数	ksettings	配置句柄

返回值	char **	子组件名列表
备注	返回值需要释放	

获取所有键名(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char **kdk_conf2_list_keys(KSettings *ksettings)	
描述	获取所有键名	
参数	ksettings	配置句柄
返回值	char **	键名列表
备注	返回值需要释放	

获取键值取值范围(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char *kdk_conf2_get_range(KSettings *ksettings, const char *key)	
描述	获取键值取值范围	
参数	ksettings	配置句柄
	key	键名
返回值	char *	取值范围 取值区间 "min,max" 枚举"{nick:value}"
备注	返回值需要释放	

检查要设置值是否在取值区间(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_range_check(KSettings *ksettings, const char *key, const char *value)	
描述	检查要设置值是否在取值区间	
参数	ksettings	配置句柄
	key	键名
	value	要设置的值
返回值	0	false
	1	true
备注	无	

设置键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_set_value(KSettings *ksettings, const char *key, const char *value)	
描述	设置键的值	
参数	ksettings	配置句柄

	key	键名
	value	要设置的值
返回值	0	false
	1	true
备注	无	

读取键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char *kdk_conf2_get_value(KSettings *ksettings, const char *key)	
描述	读取键的键值	
参数	ksettings	配置句柄
	key	键名
返回值	char *	键值
备注	返回值需要释放	

读取键的默认值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char *kdk_conf2_get_defalut_value(KSettings *ksettings, const char *key)	
描述	读取键的默认值	
参数	ksettings	配置句柄
	key	键名
返回值	char *	键值
备注	返回值需要释放	

设置 `boolean` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_set_boolean(KSettings *ksettings, const char *key, int value)	
描述	设置boolean类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
	value	要设置的值
返回值	0	false
	1	true
备注	无	

读取 `boolean` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	

接口类型	C	
原型	int kdk_conf2_get_boolean(KSettings *ksettings, const char *key)	
描述	读取boolean类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	0	false
	1	true
备注	无	

读取 `boolean` 类型的默认值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_get_default_boolean(KSettings *ksettings, const char *key)	
描述	读取boolean类型的默认值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	0	false
	1	true
备注	无	

设置 `double` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_set_double(KSettings *ksettings, const char *key, double value)	
描述	设置double类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
	value	要设置的值
返回值	0	false
	1	true
备注	无	

读取 `double` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	double kdk_conf2_get_double(KSettings *ksettings, const char *key)	
描述	读取double类型的键值 需确保键存在且类型正确	

参数	ksettings	配置句柄
	key	键名
返回值	double	键值
备注	无	

读取 `double` 类型的默认值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	double kdk_conf2_get_defalut_double(KSettings *ksettings, const char *key)	
描述	读取double类型的默认值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	double	键值
备注	无	

设置枚举类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_set_enum(KSettings *ksettings, const char *key, int value)	
描述	设置枚举类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
	value	要设置的值
返回值	0	false
	1	true
备注	无	

读取枚举类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_get_enum(KSettings *ksettings, const char *key)	
描述	读取枚举类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	int	枚举值
备注	无	

读取枚举类型的默认值(自2.4.1.0版本启用)

子模块	统一配置模块	

接口类型	C	
原型	int kdk_conf2_get_default_enum(KSettings *ksettings, const char *key)	
描述	读取枚举类型的默认值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	int	枚举值
备注	无	

设置 `int` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_set_int(KSettings *ksettings, const char *key, int value)	
描述	设置int类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
	value	要设置的值
返回值	0	false
	1	true
备注	无	

读取 `int` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_get_int(KSettings *ksettings, const char *key)	
描述	读取int类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	int	键值
备注	无	

读取 `int` 类型的默认值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_get_default_int(KSettings *ksettings, const char *key)	
描述	读取int类型的默认值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	int	键值

备注	无

设置 `int64` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_set_int64(KSettings *ksettings, const char *key, long value)	
描述	设置long类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
	value	要设置的值
返回值	0	false
	1	true
备注	无	

读取 `int64` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	long kdk_conf2_get_int64(KSettings *ksettings, const char *key)	
描述	读取long类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	long	键值
备注	无	

读取 `int64` 类型的默认值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	long kdk_conf2_get_default_int64(KSettings *ksettings, const char *key)	
描述	读取long类型的默认值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	long	键值
备注	无	

设置 `uint` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块
接口类型	C
原型	int kdk_conf2_set_uint(KSettings *ksettings, const char *key, unsigned int value)
描述	设置unsigned int类型的键值

	需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
	value	要设置的值
返回值	0	false
	1	true
备注	无	

读取 `uint` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	unsigned int kdk_conf2_get_uint(KSettings *ksettings, const char *key)	
描述	读取unsigned int类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	unsigned int	键值
备注	无	

读取 `uint` 类型的默认值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	unsigned int kdk_conf2_get_default_uint(KSettings *ksettings, const char *key)	
描述	读取unsigned int类型的默认值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	unsigned int	键值
备注	无	

设置 `uint64` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_set_uint64(KSettings *ksettings, const char *key, unsigned long value)	
描述	设置unsigned long类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
	value	要设置的值
返回值	0	false
	1	true
备注	无	

读取 `uint64` 类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	unsigned long kdk_conf2_get_uint64(KSettings *ksettings, const char *key)	
描述	读取unsigned long类型的值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	unsigned long	键值
备注	无	

读取 `uint64` 类型的默认值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	unsigned long kdk_conf2_get_default_uint64(KSettings *ksettings, const char *key)	
描述	读取unsigned long类型的默认值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	unsigned long	键值
备注	无	

设置字符串类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_set_string(KSettings *ksettings, const char *key, const char *value)	
描述	设置字符串类型的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
	value	要设置的值
返回值	0	false
	1	true
备注	无	

读取字符串类型的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char *kdk_conf2_get_string(KSettings *ksettings, const char *key)	
描述	读取字符串类型的键值 需确保键存在且类型正确	

参数	ksettings	配置句柄
	key	键名
返回值	char *	键值
备注	返回值需要释放	

读取字符串类型的默认值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char *kdk_conf2_get_default_string(KSettings *ksettings, const char *key)	
描述	读取字符串类型的默认值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	char *	键值
备注	返回值需要释放	

设置字符串列表的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_set_strv(KSettings *ksettings, const char *key, const char *const *value)	
描述	设置字符串列表的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
	value	要设置的值， 以NULL结尾的字符串列表
返回值	0	false
	1	true
备注	无	

读取字符串列表的键值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char **kdk_conf2_get_strv(KSettings *ksettings, const char *key)	
描述	读取字符串列表的键值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	char **	键值
备注	返回值需要释放	

读取字符串列表的默认值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char **kdk_conf2_get_default_strv(KSettings *ksettings, const char *key)	
描述	读取字符串列表的默认值 需确保键存在且类型正确	
参数	ksettings	配置句柄
	key	键名
返回值	char **	键值
备注	返回值需要释放	

读取键的描述(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char *kdk_conf2_get_descrition(KSettings *ksettings, const char *key)	
描述	读取键的描述	
参数	ksettings	配置句柄
	key	键名
返回值	char *	键的描述
备注	返回值需要释放	

读取键的简述(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char *kdk_conf2_get_summary(KSettings *ksettings, const char *key)	
描述	读取键的简述	
参数	ksettings	配置句柄
	key	键名
返回值	char *	键的简述
备注	返回值需要释放	

读取键的类型字符串(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char *kdk_conf2_get_type(KSettings *ksettings, const char *key)	
描述	读取键的类型字符串	
参数	ksettings	配置句柄
	key	键名
返回值	char *	键的类型
备注	返回值需要释放	

重置键值为默认值(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	void kdk_conf2_reset(KSettings *ksettings, const char *key)	
描述	重置键值为默认值	
参数	ksettings	配置句柄
	key	键名
返回值	无	
备注	无	

检测键值是否可设置(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_is_writable(KSettings *ksettings, const char *key)	
描述	检测键值是否可设置	
参数	ksettings	配置句柄
	key	键名
返回值	0	false
	1	true
备注	无	

检测组件配置中是否存在某个key(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_has_key(KSettings *ksettings, const char *key)	
描述	检测组件配置中是否存在某个key	
参数	ksettings	配置句柄
	key	键名
返回值	0	false
	1	true
备注	无	

注册回调函数响应信号(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	unsigned long kdk_conf2_connect_signal(KSettings *ksettings, const char *signal_name, KCallBack handler, void *user_data)	
描述	注册回调函数响应信号	
参数	ksettings	配置句柄
	signal_name	信号名
	handler	回调函数指针
	user_date	用户数据

返回值	unsigned long	信号连接id
备注	无	

信号详情

子模块	统一配置模块	
接口类型	C	
信号	changed::detailed	void changed(KSettings *setting, const char *key, void *user_data)
	reload	void reload(void)
备注	无	

重新读取配置文件(自2.4.1.0版本启用)

子模块	统一配置模块
接口类型	C
原型	void kdk_conf2_reload(void)
描述	重新读取配置文件
参数	无
返回值	无
备注	无

读取指定应用所有id(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	char **kdk_conf2_list_schemas(const char *app, const char *version)	
描述	查看指定应用拥有的所有schema id。可指定版本， 不指定则读取默认版本	
参数	app	应用名
	version	版本号
返回值	char **	id列表
备注	无	

重定位(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	KSettings *kdk_conf2_new_extends_id(const char *old_id, const char *new_id, const char *version)	
描述	在统一视图中创建一个新的id，内容继承自old_id。 并返回句柄	
参数	old_id	旧id
	new_id	新id
	version	版本号
返回值	KSettings*	新id句柄
备注	无	

导出配置(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_save_user_configure(const char *path)	
描述	将数据库中的用户配置导出为配置文件存放到目标路径	
参数	path	目标路径
返回值	0	false
	1	true
备注	无	

判断id是否存在于统一视图(自2.4.1.0版本启用)

子模块	统一配置模块	
接口类型	C	
原型	int kdk_conf2_is_schema(const char *id,const char *version)	
描述	判断传入的id是不是一个存在的项目	
参数	id	组件id
	version	版本号
返回值	0	false
	1	true
备注	无	

6 系统安全 SDK

6.1 桌面管控

- **安装命令：**

```
$sudo apt-get install libkysdk-desktopctrl libkysdk-desktopctrl-dev
```

- **头文件路径：**

```
/usr/include/kysdk/kysdk-security/libkydesktopctrl.h
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkydesktopctrl.so
```

6.1.1 控制面板管控

- **D-BUS接口：**
- 名称:com.kylin.kysdk.softwarecontrol
- 路径:/com/kylin/kysdk/ukui_control_center
- 接口:com.kylin.kysdk.controlcenter

设置功能模块是否可见(自1.2.0版本启用)

子模块	设置功能模块是否可见
接口类型	C

原型	int kdk_controplpanel_set_module_visible(int module, bool visible);	
描述	调用此接口传入模块名以及是否可见的bool值	
参数	module	模块ID(kdk_controlpanel_module枚举值)
	visible	true-可见，false-隐藏
返回值	int	0-成功，非0-失败
备注	无	

- **模块ID：**

```
typedef enum _kdk_controlpanel_module {

    CP_SYS = 0,         //系统 一级
    CP_SYS_DISPLAY,     //显示器
    CP_SYS_TOUCHSCREEN, //2107触摸屏
    CP_SYS_AUTOBOOT,    //2107开机启动
    CP_SYS_DEFAULTAPP,  //2107默认应用
    CP_SYS_POWER,       //电源
    CP_SYS_AUDIO,       //声音
    CP_SYS_NOTICE,      //通知
    CP_SYS_VINO,        //远程桌面
    CP_SYS_ABOUT,       //关于

    CP_DT = 16, //时间语言
    CP_DT_AREA, //区域语言
    CP_DT_DAT,  //时间日期

    CP_ACNT = 32,       //账户 一级
    CP_ACNT_CLOUD,      //云账户
    CP_ACNT_USERINFO,   //账户信息
    CP_ACNT_BIOMETRICS, //生物识别
    CP_ACNT_LOGINOPTION,//登录选项

    CP_ACNT_USERINFO_ALLPAGE,  //账户信息所有页面
    CP_ACNT_USERINFO_NOPWDLOGIN,//免密登录
    CP_ACNT_USERINFO_AUTOLOGIN, //自动登录
    CP_ACNT_USERINFO_ADDUSER,   //添加用户
    CP_ACNT_USERINFO_CHGUSERGRP,//编辑用户组
    CP_ACNT_USERINFO_DISPLAY_LASTLOGINUSER, //禁止显示最后一次登录的用户名


    CP_DEV = 48,    //设备 一级
    CP_DEV_AUDIO,    //2107声音
    CP_DEV_KEYBOARD, //键盘
    CP_DEV_MOUSE,    //鼠标
    CP_DEV_PRINTER,  //打印机
    CP_DEV_SHORTCUT, //快捷键
    CP_DEV_TOUCHPAD, //触摸板
    CP_DEV_BLUETOOTH, //蓝牙
    CP_DEV_PROJECTION, //多屏协同

    CP_NET = 64,   //网络 一级
    CP_NET_CONNECT, //有线网络
    CP_NET_PROXY, //代理
    CP_NET_VINO,  //2107桌面共享
    CP_NET_VPN,   //VPN
    CP_NET_WLAN,   //无线
    CP_NET_HOTSPOT, //热点

    CP_NOTICE = 80, //2107通知关于 一级
    CP_NOTICE_ABOUT, //2107关于
    CP_NOTICE_NOTICE, //2107通知
    CP_NOTICE_EXPERIENCEPLAN,//2107
    CP_NOITCE_SEARCH, //2107搜索

    CP_PERSONAL = 96,      //个性化 一级
    CP_PERSONAL_BACKGROUND, //背景
    CP_PERSONAL_FONTS,      //字体
    CP_PERSONAL_SCREENLOCK, //锁屏
    CP_PERSONAL_SCREENSAVER, //屏保
    CP_PERSONAL_THEME,      //主题
    CP_PERSONAL_DESKTOP,    //2107桌面

    CP_UPDATE = 112,    //更新 一级
    CP_UPDATE_BACKUP,   //备份还原
    CP_UPDATE_DEFENDER,  //2107安全
    CP_UPDATE_UPGRADE,  //更新

    CP_SECURITY = 128,  //安全 一级
    CP_SECURITY_DEFENDER, //安全中心
```

```
CP_APP = 144,       //应用 一级
CP_APP_AUTOBOOT,    //开机启动
CP_APP_DEFAULT,     //默认应用

CP_SEARCH = 160, //搜索 一级
CP_SEARCH_SEARCH, //全局搜索

CP_COMMON = 176, //通用 一级
CP_COMMON_BOOT, //系统启动


//个性化-背景-子项
CP_PERSONAL_BG_BACKGROUND = 200, //背景
CP_PERSONAL_BG_MODE,     //显示方式
CP_PERSONAL_BG_LPICTURE, //本地图片
CP_PERSONAL_BG_OPICTURE, //线上图片
CP_PERSONAL_BG_DEFAULT, //恢复默认
CP_PERSONAL_BG_PICTURE, //图片选择区

//个性化-锁屏-子项
CP_PERSONAL_SL_SHOWLOGIN = 230, //显示锁屏壁纸在登录界面
CP_PERSONAL_SL_LPICTURE, //本地图片
CP_PERSONAL_SL_OPICTURE, //线上图片
CP_PERSONAL_SL_DEFAUTL, //恢复默认
CP_PERSONAL_SL_PICTURE, //图片选择区
CP_PERSONAL_SL_LOCKSCREEN, //激活屏保时锁定屏幕
CP_PERSONAL_SL_IDLELOCK, //2203此时间段后锁屏

//个性化-屏保-子项
CP_PERSONAL_SS_OPENSCREEN = 260, //此段时间开启屏保
CP_PERSONAL_SS_PROGRAM, //屏幕保护程序
CP_PERSONAL_SS_IDLETIME, //显示休息时间
CP_PERSONAL_SS_LOCKSCREEN, //激活屏保时锁定屏幕

//系统-电源-子项
CP_SYS_POWER_CLOSEDISPLAY = 290, //此段时间后关闭显示器
CP_SYS_POWER_SLEEP,              //此段时间后系统进入睡眠操作


//需求208837 add by linleiyong 2024.05.13
CP_SYS_VINO_REMOTE_CONNECT = 300,//远程连接该设备
CP_SYS_VINO_ALLOW_VNC_CONNECT,  //允许使用VNC远程连接
CP_SYS_VINO_ALLOW_CTRL_SCREEN, //允许远程控制屏幕
CP_SYS_VINO_CONFIRM_EVERYTIME, //每次连接时确认
CP_SYS_VINO_VNC_PASSWORD,      //VNC远程连接需要输入密码

//网络-代理-子项
CP_NET_PROXY_SYSTEM = 320,  //系统代理
CP_NET_PROXY_APP,           //应用代理
CP_NET_PROXY_APT,           //APT代理

//更新-更新-子项
CP_UPDATE_UPGRADE_NOTIFY = 350, //有更新应用时通知
CP_UPDATE_UPGRADE_AUTO,         //自动更新
CP_UPDATE_UPGRADE_LIMIT,        //下载限速
CP_UPDATE_UPGRADE_ADVANCE,      //高级选项

//以上正常功能如超过1024，修改为1024以上
KYSDK_MODULE_DISABLED_MODIFY = (1<<10), //禁止操作模块1024

//1禁止操作账户信息
CP_ACNT_USERINFO_DISABLED    = KYSDK_MODULE_DISABLED_MODIFY|CP_ACNT_USERINFO_ALLPAGE,              //禁止操作账户信息
CP_ACNT_USERINFO_NOPWDLOGIN_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_ACNT_USERINFO_NOPWDLOGIN,   //禁止免密登录
CP_ACNT_USERINFO_AUTOLOGIN_DISABLED  = KYSDK_MODULE_DISABLED_MODIFY|CP_ACNT_USERINFO_AUTOLOGIN,    //禁止开机自动登录
CP_ACNT_USERINFO_ADDUSER_DISABLED    = KYSDK_MODULE_DISABLED_MODIFY|CP_ACNT_USERINFO_ADDUSER,      //禁止添加用户
CP_ACNT_USERINFO_CHGUSERGRP_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_ACNT_USERINFO_CHGUSERGRP,   //禁止编辑用户组
CP_ACNT_USERINFO_DISPLAY_LASTLOGINUSER_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_ACNT_USERINFO_DISPLAY_LASTLOGINUSER, //禁止显

//2禁止操作登录选项
CP_ACNT_LOGINOPTION_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_ACNT_LOGINOPTION,
```

```
    CP_ACNT_BIOMETRICS_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_ACNT_BIOMETRICS,

    //3禁止操作云账户
    CP_ACNT_CLOUD_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_ACNT_CLOUD,

    //4禁止操作显示器
    CP_SYS_DISPLAY_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_SYS_DISPLAY,

    //5禁止操作声音
    CP_SYS_AUDIO_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_SYS_AUDIO,

    //6禁止操作电源
    CP_SYS_POWER_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_SYS_POWER,

    //7禁止操作通知
    CP_SYS_NOTICE_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_SYS_NOTICE,

    //8禁止操作远程桌面
    CP_SYS_VINO_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_SYS_VINO,

    //9禁止操作关于
    CP_SYS_ABOUT_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_SYS_ABOUT,

    //10禁止操作蓝牙
    CP_DEV_BLUETOOTH_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_DEV_BLUETOOTH,

    //11禁止操作打印机
    CP_DEV_PRINTER_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_DEV_PRINTER,

    //12禁止操作鼠标
    CP_DEV_MOUSE_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_DEV_PRINTER,

    //13禁止操作触控板
    CP_DEV_TOUCHPAD_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_DEV_PRINTER,

    //14禁止操作键盘
    CP_DEV_KEYBOARD_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_DEV_PRINTER,

    //15禁止操作快捷键
    CP_DEV_SHORTCUT_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|CP_DEV_PRINTER,

    //16禁止操作多屏协同
    CP_DEV_PROJECTION_DISABLED = KYSDK_MODULE_DISABLED_MODIFY| CP_DEV_PROJECTION,
    //17禁止操作有线网络
    CP_NET_CONNECT_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_NET_CONNECT,
    //18禁止操作无线局域网
    CP_NET_WLAN_DISABLED = KYSDK_MODULE_DISABLED_MODIFY| CP_NET_WLAN,
    //19禁止操作代理
     CP_NET_PROXY_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_NET_PROXY,
    //20禁止操作VPN
    CP_NET_VPN_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_NET_VPN,
    //21禁止操作移动热点
    CP_NET_HOTSPOT_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_NET_HOTSPOT,

    //22禁止操作背景
    CP_PERSONAL_BACKGROUND_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_PERSONAL_BACKGROUND,
    //23禁止操作主题
    CP_PERSONAL_THEME_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_PERSONAL_THEME,
    //24禁止操作锁屏
    CP_PERSONAL_SCREENLOCK_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_PERSONAL_SCREENLOCK,
    //25禁止操作屏保
    CP_PERSONAL_SCREENSAVER_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_PERSONAL_SCREENSAVER,
    //26禁止操作字体
    CP_PERSONAL_FONTS_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_PERSONAL_FONTS,
    //27禁止操作任务栏

    //28禁止操作时间与日期
     CP_DT_DAT_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_DT_DAT,
    //29禁止操作区域语言
    CP_DT_AREA_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_DT_AREA,
```

```
    //30禁止操作更新（原补丁更新策略）
    CP_UPDATE_UPGRADE_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_UPDATE_UPGRADE,
    //31禁止操作备份还原
    CP_UPDATE_BACKUP_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_UPDATE_BACKUP,
    //32禁止操作安全中心
    CP_SECURITY_DEFENDER_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_SECURITY_DEFENDER,
    //33禁止操作开机启动
    CP_APP_AUTOBOOT_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_APP_AUTOBOOT,
    //34禁止操作默认应用
    CP_APP_DEFAULT_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_APP_DEFAULT,
    //35禁止操作全局搜索
    CP_SEARCH_SEARCH_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_SEARCH_SEARCH,
    //36禁止操作系统启动
    CP_COMMON_BOOT_DISABLED = KYSDK_MODULE_DISABLED_MODIFY|    CP_COMMON_BOOT,

    CP_END

} kdk_controlpanel_module;
```

获取控制面板是否可见(自1.2.0版本启用)

子模块	获取控制面板是否可见	
接口类型	C	
原型	bool kdk_controlpanel_get_module_visible(int module);	
描述	获取模块是否可见，传入模块ID，返回该模块的结果	
参数	module	模块ID(kdk_controlpanel_module枚举值)
返回值	int	true-可见，false-隐藏
备注	无	

设置控制面板状态(自2.4.0版本启用)

子模块	设置控制面板状态	
接口类型	C	
原型	int kdk_controplpanel_set_module_status(int module, int status);	
描述	调用此接口传入模块名和模块状态	
参数	module	模块ID(kdk_controlpanel_module枚举值)
	status	0-隐藏 1-可见 2-禁用 3-启用
返回值	bool	0-成功，非0-失败
备注	无	

批量设置控制面板状态(自1.2.0版本启用)

子模块	批量设置控制面板状态	
接口类型	Dbus	
原型	int SetStatusList(array(int module,int status))	
描述	调用此接口传入模块名和模块状态的数组	
参数	module	模块ID(kdk_controlpanel_module枚举值)
	status	0-隐藏 1-可见 2-禁用 3-启用
返回值	int	0-成功，非0-失败
备注	module中大于1024枚举,status中2,3为版本(2.4.0)中扩展	

6.1.2 开始菜单管控

- **D-BUS接口：**

- 名称：com.kylin.kysdk.softwarecontrol
- 路径：/com/kylin/kysdk/ukui_menu
- 接口：com.kylin.kysdk.ukui_menu
- ** 管控模式ID：**

```
typedef enum _kdk_model_type {
    MODEL_NORMAL = 0,
    MODEL_BLACKLIST,
    MODEL_WHITELIST,
} kdk_model_type;
```

设置开始菜单管控模式(自1.2.0版本启用)

子模块	1.设置开始菜单管控模式	
接口类型	C	
原型	int kdk_startmenu_set_model(int type);	
描述	调用此接口传入模式ID设置开始菜单管控模式	
参数	type	模式ID(kdk_model_type枚举值)
返回值	int	0-成功，非0-失败
备注	无	

添加应用到开始菜单管控黑名单(自1.2.0版本启用)

子模块	2.添加应用到开始菜单管控黑名单	
接口类型	C	
原型	int kdk_startmenu_add_blacklist(const char *appname, const char *path);	
描述	调用此接口传入应用名和desktop文件路径	
参数	appname	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败
备注	无	

删除开始菜单黑名单中对应的数据项(自1.2.0版本启用)

子模块	3.删除开始菜单黑名单中对应的数据项	
接口类型	C	
原型	int kdk_startmenu_del_blacklist(const char *appname, const char *path);	
描述	删除开始菜单黑名单中对应的数据项	
参数	appname	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败
备注	无	

清空开始菜单黑名单(自1.2.0版本启用)

子模块	4.清空开始菜单黑名单
接口类型	C
原型	int kdk_startmenu_clear_blacklist();
描述	调用此接口清空开始菜单黑名单

参数	无	
返回值	int	0-成功，非0-失败
备注	无	

添加应用快捷方式到白名单(自1.2.0版本启用)

子模块	5.添加应用快捷方式到白名单	
接口类型	C	
原型	int kdk_startmenu_add_whitelist(const char *appname, const char *path);	
描述	调用此接口添加应用到开始菜单白名单	
参数	appname	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败
备注	无	

删除开始菜单应用从白名单(自1.2.0版本启用)

子模块	6.删除开始菜单应用从白名单	
接口类型	C	
原型	int kdk_startmenu_del_whitelist(const char *appname, const char *path);	
描述	调用此接口删除开始菜单管控白名单中对应的数据项	
参数	name	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败
备注	无	

清空开始菜单管控白名单(自1.2.0版本启用)

子模块	7.清空开始菜单管控白名单	
接口类型	C	
原型	int kdk_startmenu_clear_whitelist();	
描述	调用此接口清空开始菜单白名单	
返回值	int	0-成功，非0-失败
备注	无	

6.1.3 桌面应用管控

- **D-BUS接口：**
- 名称：com.kylin.kysdk.softwarecontrol
- 路径：/com/kylin/kysdk/peony
- 接口：com.kylin.kysdk.peony

设置管控模式(自1.2.0版本启用)

子模块	1.设置管控模式	
接口类型	C	
原型	int kdk_desktop_set_model(int type);	
描述	调用此接口传入模式ID设置管控模式	

参数	type	模式ID(kdk_model_type枚举值)
返回值	int	0-成功，非0-失败
备注	无	

添加应用到桌面应用管控黑名单(自1.2.0版本启用)

子模块	2.添加应用到桌面应用管控黑名单	
接口类型	C	
原型	int kdk_desktop_add_blacklist(const char *appname, const char *path);	
描述	调用此接口传入应用名和desktop文件路径	
参数	appname	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败
备注	无	

删除桌面应用管控黑名单中对应的数据项(自1.2.0版本启用)

子模块	3.删除桌面应用管控黑名单中对应的数据项	
接口类型	C	
原型	kdk_desktop_del_blacklist(const char *appname, const char *path);	
描述	删除桌面应用管控黑名单中对应的数据项	
参数	appname	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败
备注	无	

清空桌面应用管控黑名单(自1.2.0版本启用)

子模块	4.清空桌面应用管控黑名单	
接口类型	C	
原型	int kdk_desktop_clear_blacklist();	
描述	调用此接口清空桌面应用管控黑名单	
参数	无	
返回值	int	0-成功，非0-失败
备注	无	

添加应用到桌面应用管控白名单(自1.2.0版本启用)

子模块	5.添加应用到桌面应用管控白名单	
接口类型	C	
原型	int kdk_desktop_add_whitelist(const char *appname, const char *path);	
描述	调用此接口添加应用到桌面应用管控白名单	
参数	appname	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败

备注	无

删除桌面应用管控白名单中对应的数据项(自1.2.0版本启用)

子模块	6.删除桌面应用管控白名单中对应的数据项	
接口类型	C	
原型	int kdk_desktop_del_whitelist(const char *appname, const char *path);	
描述	调用此接口删除桌面应用管控白名单中对应的数据项	
参数	name	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败
备注	无	

清空桌面应用白名单(自1.2.0版本启用)

子模块	7.清空桌面应用白名单	
接口类型	C	
原型	int kdk_desktop_clear_whitelist();	
描述	调用此接口清空桌面应用白名单	
版本号	无	
返回值	int	0-成功，非0-失败
备注	无	

6.1.4 软件商店管控

- **D-BUS接口：**
- 名称：com.kylin.kysdk.softwarecontrol
- 路径：/com/kylin/kysdk/kylin_software_center
- 接口：com.kylin.kysdk.kylin_software_center

软件商店设置管控模式(自1.2.0版本启用)

子模块	1.软件商店设置管控模式	
接口类型	C	
原型	int kdk_appstore_set_model(int type);	
描述	调用此接口传入模式ID设置管控模式	
参数	type	模式ID(kdk_model_type枚举值)
返回值	int	0-成功，非0-失败
备注	无	

添加应用到软件商店管控黑名单(自1.2.0版本启用)

子模块	2.添加应用到软件商店管控黑名单	
接口类型	C	
原型	int kdk_appstore_add_blacklist(const char *appname);	
描述	调用此接口传入应用名和desktop文件路径	
参数	appname	应用名称
返回值	int	0-成功，非0-失败
备注	无	

删除软件商店管控黑名单中对应的数据项(自1.2.0版本启用)

子模块	3.删除软件商店管控黑名单中对应的数据项	
接口类型	C	
原型	int kdk_appstore_del_blacklist(const char *appname);	
描述	删除软件商店管控黑名单中对应的数据项	
参数	appname	应用名称
返回值	int	0-成功，非0-失败
备注	无	

清空软件商店管控黑名单(自1.2.0版本启用)

子模块	4.清空软件商店管控黑名单	
接口类型	C	
原型	int kdk_appstore_clear_blacklist();	
描述	调用此接口清空软件商店管控黑名单	
参数	无	
版本号	无	
返回值	int	0-成功，非0-失败
备注	无	

添加应用到软件商店管控白名单(自1.2.0版本启用)

子模块	2.添加应用到软件商店管控白名单	
接口类型	C	
原型	int kdk_appstore_add_whitelist(const char *appname);	
描述	调用此接口传入应用名	
参数	appname	应用名称
返回值	int	0-成功，非0-失败
备注	无	

删除软件商店管控白名单中对应的数据项(自1.2.0版本启用)

子模块	3.删除软件商店管控白名单中对应的数据项	
接口类型	C	
原型	int kdk_appstore_del_whitelist(const char *appname);	
描述	删除软件商店管控白名单中对应的数据项	
参数	appname	应用名称
返回值	int	0-成功，非0-失败
备注	无	

清空软件商店应用白名单列表(自1.2.0版本启用)

子模块	4.清空软件商店应用白名单列表	
接口类型	C	
原型	int kdk_appstore_clear_whitelist();	
描述	调用此接口清空软件商店管控黑名单	
参数	无	

返回值	int	0-成功，非0-失败
备注	无	

6.1.5 登录管控

- **D-BUS(system bus)接口：**
- 名称：com.kylin.kysdk.softwarectrlsystem
- 路径：/com/kylin/kysdk/ukui_greete
- 接口：com.kylin.kysdk.ukui_greete

登录管控设置管控模式(自1.2.0版本启用)

子模块	1.登录管控设置管控模式	
接口类型	C	
原型	int kdk_greeter_set_model(int type);	
描述	调用此接口传入模式ID设置管控模式	
参数	type	模式ID(kdk_model_type枚举值)
返回值	int	0-成功，非0-失败
备注	无	

添加用户名到登录界面管控黑名单(自1.2.0版本启用)

子模块	2.添加用户名到登录界面管控黑名单	
接口类型	C	
原型	int kdk_greeter_add_blacklist(const char *username);	
描述	添加用户名到登录界面管控黑名单	
参数	username	用户名
返回值	int	0-成功，非0-失败
备注	无	

删除登录界面管控黑名单中对应的数据项(自1.2.0版本启用)

子模块	3.删除登录界面管控黑名单中对应的数据项	
接口类型	C	
原型	int kdk_greeter_del_blacklist(const char *username);	
描述	删除登录界面管控黑名单中对应的数据项	
参数	username	用户名
返回值	int	0-成功，非0-失败
备注	无	

清空登录界面管控黑名单(自1.2.0版本启用)

子模块	4.清空登录界面管控黑名单	
接口类型	C	
原型	int kdk_greeter_clear_blacklist();	
描述	调用此接口清空登录界面管控黑名单	
参数	无	
返回值	int	0-成功，非0-失败
备注	无	

添加应用到登录界面白名单(自1.2.0版本启用)

子模块	5.添加应用到登录界面白名单	
接口类型	C	
原型	int kdk_greeter_add_whitelist(const char *username);	
描述	调用此接口添加应用到登录界面白名单	
参数	username	用户名
版本号	无	
返回值	int	0-成功，非0-失败
备注	无	

删除登录界面白名单中对应的用户(自1.2.0版本启用)

子模块	6.删除登录界面白名单中对应的用户	
接口类型	C	
原型	int kdk_greeter_del_whitelist(const char *username);	
描述	调用此接口删除登录界面白名单中对应的用户	
参数	name	用户名
返回值	int	0-成功，非0-失败
备注	无	

清空登录界面管控白名单(自1.2.0版本启用)

子模块	7.清空登录界面管控白名单	
接口类型	C	
原型	int kdk_greeter_clear_whitelist();	
描述	调用此接口清空登录界面管控白名单	
返回值	int	0-成功，非0-失败
备注	无	

6.1.6 电源管控

- **D-BUS接口：**
- 名称：com.kylin.kysdk.systemsettings
- 路径：/com/kylin/kysdk/powersettings
- 接口：com.kylin.kysdk.powersettings

设置台式机系统空闲并于指定时间后挂起(自1.2.0版本启用)

子模块	设置台式机系统空闲并于指定时间后挂起	
接口类型	C	
原型	int kdk_powersetting_set_desktop_idle_hungup(unsigned int tm);	
描述	台式机系统空闲并于指定时间后挂起， 参数时间必须是0或正整数，单位是秒	
参数	tm	空闲时间,单位秒
返回值	int	0-成功，非0-失败
备注	无	

设置笔记本系统空闲并于指定时间后挂起(自1.2.0版本启用)

子模块	设置笔记本系统空闲并于指定时间后挂起	
接口类型	C	
原型	int kdk_powersetting_set_laptop_idle_hungup(unsigned int tm);	
描述	笔记本系统空闲并于指定时间后挂起， 参数时间必须是0或正整数，单位是秒	
参数	tm	空闲时间,单位秒
返回值	int	0-成功，非0-失败
备注	无	

设置笔记本系统空闲并于指定时间后关闭显示器(自1.2.0版本启用)

子模块	设置笔记本系统空闲并于指定时间后关闭显示器	
接口类型	C	
原型	int kdk_powersetting_set_laptop_idle_closedisplay(unsigned int tm);	
描述	笔记本系统空闲并于指定时间后关闭显示器， 参数时间必须是0或正整数	
参数	tm	空闲时间,单位秒
返回值	int	0-成功，非0-失败
备注	无	

台式机系统空闲并于指定时间后关闭显示器(自1.2.0版本启用)

子模块	台式机系统空闲并于指定时间后关闭显示器	
接口类型	Dbus	
原型	int kdk_powersetting_set_desktop_idle_closedisplay(unsigned int tm);	
描述	台式机系统空闲并于指定时间后关闭显示器， 参数时间必须是0或正整数	
参数	tm	空闲时间,单位秒
返回值	int	0-成功，非0-失败
备注	无	

获取台式机系统空闲并于指定时间后关闭显示器(自1.2.0版本启用)

子模块	获取台式机系统空闲并于指定时间后关闭显示器	
接口类型	C	
原型	unsigned int kdk_powersetting_get_desktop_idle_closedisplay();	
描述	获取台式机关闭显示器的系统空闲时间	
参数	无	
返回值	int	台式机关闭显示器的系统空闲时间
备注	无	

获取台式机系统空闲并于指定时间后挂起的时间(自1.2.0版本启用)

子模块	获取台式机系统空闲并于指定时间后挂起的时间
接口类型	C

原型	unsigned int kdk_powersetting_get_desktop_idle_hungup();	
描述	获取台式机系统空闲并于指定时间后挂起的时间	
参数	无	
版本号	无	
返回值	int	台式机挂起的系统空闲时间
备注	无	

获取笔记本系统空闲并于指定时间后挂起的时间(自1.2.0版本启用)

子模块	获取笔记本系统空闲并于指定时间后挂起的时间	
接口类型	Dbus	
原型	int kdk_powersetting_set_laptop_idle_hungup(unsigned int tm);	
描述	获取笔记本系统空闲并于指定时间后挂起的时间	
参数	无	
返回值	int	笔记本挂起的系统空闲时间
备注	无	

获取笔记本系统空闲并于指定时间后关闭显示器的时间(自1.2.0版本启用)

子模块	获取笔记本系统空闲并于指定时间后关闭显示器的时间	
接口类型	C	
原型	unsigned int kdk_powersetting_get_laptop_idle_closedisplay();	
描述	获取笔记本系统空闲并于指定时间后关闭显示器的时间	
参数	无	
返回值	int	笔记本挂起的系统空闲时间
备注	无	

6.1.7 屏保管控

- **D-BUS接口：**
- 名称：com.kylin.kysdk.systemsettings
- 路径：/com/kylin/kysdk/screensaver
- 接口：com.kylin.kysdk.screensaver

屏保管控设置屏保图片(自1.2.0版本启用)

子模块	屏保管控设置屏保图片	
接口类型	C	
原型	int kdk_screensaver_set_file(const char *path);	
描述	设置屏保图片	
参数	path	文件路径
返回值	int	0-成功，非0-失败
备注	无	

屏保管控启用屏保功能(自1.2.0版本启用)

子模块	屏保管控启用屏保功能
接口类型	C

原型	int kdk_screensaver_enable();	
描述	屏保管控启用屏保功能	
参数	无	
返回值	int	0-成功，非0-失败
备注	无	

屏保管控禁用屏保功能(自1.2.0版本启用)

子模块	屏保管控禁用屏保功能	
接口类型	C	
原型	int kdk_screensaver_disable();	
描述	屏保管控禁用屏保功能	
参数	无	
返回值	int	0-成功，非0-失败
备注	无	

屏保管控设置启用自动锁屏(自1.2.0版本启用)

子模块	设置启用自动锁屏	
接口类型	C	
原型	int kdk_screensaver_autolock_enable();	
描述	设置启用自动锁屏	
参数	无	
返回值	int	0-成功，非0-失败
备注	无	

屏保管控设置禁用自动锁屏(自1.2.0版本启用)

子模块	设置禁用自动锁屏	
接口类型	C	
原型	int kdk_screensaver_autolock_disable();	
描述	设置禁用自动锁屏	
参数	无	
返回值	int	0-成功，非0-失败
备注	无	

屏保管控设置自动锁屏时间(自1.2.0版本启用)

子模块	设置自动锁屏时间	
接口类型	C	
原型	int kdk_screensaver_idlelock_time(unsigned int tm);	
描述	设置自动锁屏时间	
参数	tm	空闲时间间隔,单位秒
返回值	int	0-成功，非0-失败
备注	无	

屏保管控设置自动屏保时间(自1.2.0版本启用)

子模块	设置自动屏保时间

接口类型	C	
原型	int kdk_screensaver_autolock_time(unsigned int tm);	
描述	设置自动屏保时间	
参数	tm	空闲时间间隔,单位秒
返回值	int	0-成功，非0-失败
备注	无	

6.1.8 任务栏管控

- **D-BUS接口：**
- 名称：com.kylin.kysdk.softwarecontrol
- 路径：/com/kylin/kysdk/ukui_panel
- 接口：com.kylin.kysdk.ukui_panel

任务栏管控设置管控模式(自1.2.0版本启用)

子模块	设置管控模式	
接口类型	C	
原型	int kdk_taskpanel_set_model(int type);	
描述	调用此接口传入模式ID设置管控模式	
参数	type	模式ID(kdk_model_type枚举值)
返回值	int	0-成功，非0-失败
备注	无	

添加应用到件商店管控黑名单(自1.2.0版本启用)

子模块	添加应用到件商店管控黑名单	
接口类型	C	
原型	int kdk_taskpanel_add_blacklist(const char *appname, const char *path);	
描述	调用此接口传入应用名和desktop文件路径	
参数	appname	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败
备注	无	

删除任务栏管控黑名单中对应的数据项(自1.2.0版本启用)

子模块	删除任务栏管控黑名单中对应的数据项	
接口类型	C	
原型	int kdk_taskpanel_del_blacklist(const char *appname, const char *path);	
描述	删除任务栏管控黑名单中对应的数据项	
参数	appname	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败
备注	无	

清空任务栏管控黑名单(自1.2.0版本启用)

子模块	清空任务栏管控黑名单	
接口类型	C	
原型	int kdk_taskpanel_clear_blacklist();	
描述	调用此接口清空任务栏管控黑名单	
参数	无	
返回值	int	0-成功，非0-失败
备注	无	

添加应用到件商店管控白名单(自1.2.0版本启用)

子模块	添加应用到件商店管控白名单	
接口类型	C	
原型	int kdk_taskpanel_add_whitelist(const char *appname, const char *path);	
描述	调用此接口传入应用名和desktop文件路径	
参数	appname	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败
备注	无	

删除任务栏管控白名单中对应的数据项(自1.2.0版本启用)

子模块	删除任务栏管控白名单中对应的数据项	
接口类型	C	
原型	int kdk_taskpanel_del_whitelist(const char *appname, const char *path);	
描述	删除任务栏管控白名单中对应的数据项	
参数	appname	应用名称
	path	desktop文件路径
返回值	int	0-成功，非0-失败
备注	无	

清空任务栏管控白名单(自1.2.0版本启用)

子模块	清空任务栏管控白名单	
接口类型	C	
原型	int kdk_taskpanel_clear_whitelist();	
描述	清空任务栏管控白名单	
参数	无	
返回值	int	0-成功，非0-失败
备注	无	

6.1.9 壁纸管控

- **D-BUS接口：**
- 名称：com.kylin.kysdk.systemsettings
- 路径：/com/kylin/kysdk/wallpaper
- 接口：com.kylin.kysdk.wallpaper

设置桌面壁纸(自1.2.0版本启用)

子模块	设置桌面壁纸	
接口类型	C	
原型	int kdk_wallpaper_set_file(const char *path);	
描述	设置桌面壁纸	
参数	path	文件路径
返回值	int	0-成功，非0-失败
备注	无	

6.1.10 水印管控

- **D-BUS接口：**
- 名称：com.kylin.kysdk.systemsettins
- 路径：com/kylin/kysdk/watermark
- 接口：com.kylin.kysdk.watermark

创建水印(自1.2.0版本启用)

子模块	创建水印	
接口类型	C	
原型	int kdk_watermark_create(const char *name);	
描述	创建水印	
参数	name	水印名称
返回值	int	0-成功，非0-失败
备注	无	

删除水印(自1.2.0版本启用)

子模块	删除水印	
接口类型	C	
原型	int kdk_watermark_delete(const char *name);	
描述	删除水印	
参数	name	水印名称
返回值	int	0-成功，非0-失败
备注	无	

设置水印是否可视(自1.2.0版本启用)

子模块	设置水印是否可视	
接口类型	C	
原型	int kdk_watermark_set_visibe(const char *name, bool visibel);	
描述	设置水印是否可视	
参数	name	水印名称
	visibel	true-显示，false-隐藏
返回值	int	0-成功，非0-失败
备注	无	

获取水印是否可视(自1.2.0版本启用)

子模块	获取水印是否可视

接口类型	C	
原型	bool kdk_watermark_get_visibe(const char *name);	
描述	获取水印是否可视	
参数	name	水印名称
返回值	bool	true-显示，false-隐藏
备注	无	

设置水印用户名是否显示(自1.2.0版本启用)

子模块	设置水印用户名是否显示	
接口类型	C	
原型	int kdk_watermark_set_username_visible(const char *name, bool visible);	
描述	设置水印用户名是否显示	
参数	name	水印名称
	visible	true-显示，false-隐藏
返回值	int	0-成功，非0-失败
备注	无	

获取水印用户名是否显示(自1.2.0版本启用)

子模块	获取水印用户名是否显示	
接口类型	C	
原型	bool kdk_watermark_get_username_visible(const char *name);	
描述	获取水印用户名是否显示	
参数	name	水印名称
返回值	bool	true-显示，false-隐藏
备注	无	

设置水印时间戳是否显示(自1.2.0版本启用)

子模块	设置水印时间戳是否显示	
接口类型	C	
原型	int kdk_watermark_set_timestamp_visible(const char *name, bool visible);	
描述	设置水印时间戳是否显示	
参数	name	水印名称
	visible	true-显示，false-隐藏
返回值	int	0-成功，非0-失败
备注	无	

获取水印时间戳是否显示(自1.2.0版本启用)

子模块	获取水印时间戳是否显示	
接口类型	C	
原型	bool kdk_watermark_get_timestamp_visible(const char *name);	

描述	获取水印时间戳是否显示	
参数	name	水印名称
返回值	bool	true-显示，false-隐藏
备注	无	

设置水印内容是否显示(自1.2.0版本启用)

子模块	设置水印内容是否显示	
接口类型	C	
原型	int kdk_watermark_set_context_visible(const char *name, bool visible);	
描述	设置水印内容是否显示	
参数	name	水印名称
	visible	true-显示，false-隐藏
返回值	int	0-成功，非0-失败
备注	无	

获取水印内容是否可视(自1.2.0版本启用)

子模块	获取水印内容是否可视	
接口类型	C	
原型	bool kdk_watermark_get_context_visible(const char *name);	
描述	获取水印内容是否可视	
参数	name	水印名称
返回值	bool	true-显示，false-隐藏
备注	无	

设置水印透明度(自1.2.0版本启用)

子模块	设置水印透明度	
接口类型	C	
原型	int kdk_watermark_set_opacity(const char *name, unsigned int trans);	
描述	设置水印透明度	
参数	name	水印名称
	trans	0-100的透明度
返回值	int	0-成功，非0-失败
备注	无	

获取水印透明度(自1.2.0版本启用)

子模块	获取水印透明度	
接口类型	C	
原型	unsigned int kdk_watermark_get_opacity(const char *name);	
描述	获取水印透明度	
参数	name	水印名称

返回值	int	0-100透明度
备注	无	

设置水印时间格式(自1.2.0版本启用)

子模块	设置水印时间格式	
接口类型	C	
原型	int kdk_watermark_set_timeformat(const char *name, const char *fmt);	
描述	设置水印时间格式	
参数	name	水印名称
	fmt	时间格式， 符合QDateTime时间格式
返回值	int	0-成功，非0-失败
备注	无	

获取水印时间格式(自1.2.0版本启用)

子模块	获取水印时间格式	
接口类型	C	
原型	int kdk_watermark_get_timeformat(const char *name, char *fmt);	
描述	获取水印时间格式	
参数	[in] name	水印名称
	[out]fmt	时间格式
返回值	int	0-成功，非0-失败
备注	无	

设置水印内容(自1.2.0版本启用)

子模块	设置水印内容	
接口类型	C	
原型	int kdk_watermark_set_context(const char *name, const char *context);	
描述	设置水印内容	
参数	name	水印名称
	context	水印内容
返回值	int	0-成功，非0-失败
备注	无	

获取水印内容(自1.2.0版本启用)

子模块	获取水印内容	
接口类型	C	
原型	int kdk_watermark_get_context(const char *name, char *context);	
描述	获取水印内容	
参数	name	水印名称
	context	水印内容

版本号	无	
返回值	int	0-成功，非0-失败
备注	无	

设置水印字体大小(自1.2.0版本启用)

子模块	设置水印字体大小	
接口类型	C	
原型	int kdk_watermark_set_fontsize(const char *name, int size);	
描述	设置水印字体大小	
参数	name	水印名称
	size	字体大小
返回值	int	0-成功，非0-失败
备注	这里的数字指的是相对大小	

获取水印字体大小(自1.2.0版本启用)

子模块	获取水印字体大小	
接口类型	C	
原型	int kdk_watermark_get_fontsize(const char *name);	
描述	获取水印字体大小	
参数	name	水印名称
返回值	int	字体大小
备注	无	

设置水印位置(自1.2.0版本启用)

子模块	设置水印位置	
接口类型	C	
原型	int kdk_watermark_set_position(const char *name, unsigned int left, unsigned int top, unsigned int horizontal, unsigned int vertical);	
描述	设置水印位置	
参数	[in] name	水印名称
	[in] left	中心点左边距离
	[in] right	中心点右边距离
	[in] horizontal	横向大小
	[in] vertical	纵向大小
返回值	int	0-成功，非0-失败
备注	[in]代表入参，[out]出参	

获取水印位置(自1.2.0版本启用)

子模块	获取水印位置	
接口类型	C	
原型	int kdk_watermark_get_position(const char *name, unsigned int *left, unsigned int *top, unsigned int *horizontal, unsigned int *vertical);	

描述	获取水印位置	
参数	[in] name	水印名称
	[out] left	中心点左边距离
	[out] right	中心点右边距离
	[out] horizontal	横向大小
	[out] vertical	纵向大小
返回值	int	0-成功，非0-失败
备注	[in]代表入参，[out]出参	

设置水印间距(自1.2.0版本启用)

子模块	设置水印间距	
接口类型	C	
原型	int kdk_watermark_set_density(const char *name, int horizontal, int vertical);	
描述	设置水印间距	
参数	[in] name	水印名称
	[in] horizontal	水平间距
	[in] vertical	垂直间距
返回值	int	0-成功，非0-失败
备注	如果是[-1,-1]则表示只显示一个(默认只显示一个)单位是pt	

获取水印间距(自1.2.0版本启用)

子模块	获取水印间距	
接口类型	C	
原型	int kdk_watermark_get_density(const char *name, int *horizontal, int *vertical);	
描述	获取水印密集程度	
参数	[in] name	水印名称
	[out] horizontal	水平间距
	[out] vertical	垂直间距
返回值	int	字体大小
备注	如果是[-1,-1]则表示只显示一个(默认只显示一个)单位是pt	

设置水印旋转角度(自1.2.0版本启用)

子模块	设置水印旋转角度	
接口类型	C	
原型	int kdk_watermark_set_rotate(const char *name, int rotate);	
描述	设置水印旋转角度	
参数	name	水印名称
	rotate	角度,默认0,可在(-360~360之间设置）
返回值	int	0-成功，非0-失败
备注	无	

获取水印旋转角度(自1.2.0版本启用)

子模块	获取水印旋转角度	
接口类型	C	
原型	int kdk_watermark_get_rotate(const char *name);	
描述	获取水印旋转角度	
参数	name	水印名称
返回值	int	旋转角度
备注	无	

设置水印字体颜色(自1.2.0版本启用)

子模块	设置水印字体颜色	
接口类型	C	
原型	int kdk_watermark_set_fontcolor(const char *name, unsigned int r, unsigned int g, unsigned int b);	
描述	设置水印字体颜色	
参数	name	水印名称
	r	RGB中R色值
	g	RGB中G色值
	b	RGB中B色值
返回值	int	0-成功，非0-失败
备注	无	

获取水印字体颜色(自1.2.0版本启用)

子模块	获取水印字体颜色	
接口类型	C	
原型	int kdk_watermark_get_fontcolor(const char *name, unsigned int *r, unsigned int *g, unsigned int *b);	
描述	获取水印字体颜色	
参数	[in] name	水印名称
	[out] r	RGB中R色值
	[out] g	RGB中G色值
	[out] b	RGB中B色值
返回值	int	0-成功，非0-失败
备注	无	

6.2 应用安全

应用安全提供应用联网、防安装卸载、执行控制、分级、行为审计管控以及提供风险提示。

- **安装命令：**

```
$ sudo apt-get install libkysdk-applicationsec libkysdk-applicationsec-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-applicationsec
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKAPPLICATION kysdk-applicationsec)
target_include_directories(demo PRIVATE ${KYSDKAPPLICATION_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKAPPLICATION_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKAPPLICATION_LIBRARIES})
```

6.2.1 应用联网管控

- **头文件路径：**

```
#include "/usr/include/kysdk/kysdk-security/libkyapplicationsec.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkyapplicationsec.so
```

- **子模块信息：**

> 设置应用包中所有程序均可联网(自1.2.0版本启用)

子模块	应用联网管控	
接口类型	C	
原型	int kdk_set_app_can_net(const char* pkgname)	
描述	设置应用包中所有程序均可联网	
参数	pkgname	包名
返回值	0	成功
	-5000	门禁管控禁止访问
	非零	失败
备注	无	

> 设置应用包中所有程序均不可联网(自1.2.0版本启用)

子模块	应用联网管控	
接口类型	C	
原型	int kdk_clear_app_can_net(const char* pkgname)	
描述	设置应用包中所有程序均不可联网	
参数	pkgname	包名
返回值	0	成功
	-5000	门禁管控禁止访问
	非零	失败
备注	无	

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "libkyapplicationsec.h"

int main()
{
    int rc;
    char* pkgname = "qaxbrowser-safe-stable";

    rc = kdk_set_app_can_net(pkgname);
    printf("setAppCanNet pkgname:%s rc:%d\n", pkgname, rc);

    rc = kdk_clear_app_can_net(pkgname);
    printf("clearAppCanNet pkgname:%s rc:%d\n", pkgname, rc);

    return 0;
}
```

6.2.2 应用装卸管控

6.2.2.1 应用防安装管控

- **dbus信息**
  - System Bus 接口
  - dbus 服务名称：com.kylin.kysdk.applicationsec
  - 路径名称：/com/kylin/kysdk/applicationsec
  - Interfaces：com.kylin.kysdk.applicationsec.antiinstall
- **子模块信息：**

设置防安装应用黑白名单开关标识(自1.2.0.9-0k3.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	int SetMode(int mode)	
描述	设置防安装应用黑白名单开关标识	
参数	mode	模式（0： 不开启黑白名单，1： 开启白名单，2： 开启黑名单，其它值非法）
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

获取防安装应用黑白名单开关标识(自1.2.0.9-0k3.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	int GetMode()	
描述	获取防安装应用黑白名单开关标识	
参数	无	无
返回值	0	未开启黑白名单
	1	开启白名单
	2	开启黑名单

	-5000	门禁管控禁止访问
备注	无	

添加一个防安装黑名单应用(自1.2.0.9-0k3.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	int AddBlackPackage(string package)	
描述	添加一个防安装黑名单应用	
参数	package	包名
返回值	0	成功
	-1	出错
	1	软件包已经在列表
	-5000	门禁管控禁止访问
备注	无	

添加一个防安装白名单应用(自1.2.0.9-0k3.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	int AddWhitePackage(string package)	
描述	添加一个防安装白名单应用	
参数	package	包名
返回值	0	成功
	-1	出错
	1	软件包已经在列表
	-5000	门禁管控禁止访问
备注	无	

删除一个防安装黑名单应用(自1.2.0.9-0k3.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	int DelBlackPackage(string package)	
描述	删除一个防安装黑名单应用	
参数	package	包名
返回值	0	不存在
	1	成功
	-1	出错
	-5000	门禁管控禁止访问
备注	无	

删除一个防安装白名单应用(自1.2.0.9-0k3.0版本启用)

子模块	应用防安装管控
接口类型	dbus
原型	int DelWhitePackage(string package)

描述	删除一个防安装白名单应用	
参数	package	包名
返回值	0	不存在
	1	成功
	-1	出错
	-5000	门禁管控禁止访问
备注	无	

添加防安装黑名单应用列表(自1.2.0.9-0k3.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	int AddBlackPackagesList(Array of [String] package_list)	
描述	添加防安装黑名单应用列表	
参数	package_list	包名列表
返回值	0	成功
	-1	出错
	1	软件包已经在列表
	-5000	门禁管控禁止访问
备注	无	

添加防安装白名单应用列表(自1.2.0.9-0k3.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	int AddWhitePackagesList(Array of [String] package_list)	
描述	添加防安装白名单应用列表	
参数	package_list	包名列表
返回值	0	成功
	-1	出错
	1	软件包已经在列表
	-5000	门禁管控禁止访问
备注	无	

判断应用是否在防安装黑名单中(自1.2.0.9-0k4.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	int PackageIsInBlackList(string package)	
描述	判断应用是否在防安装黑名单中	
参数	package	包名
返回值	0	不在黑名单中
	1	在黑名单中
	-5000	门禁管控禁止访问
备注	无	

判断应用是否在防安装白名单中(自1.2.0.9-0k4.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	int PackageIsInWhilteList(string package)	
描述	判断应用是否在防安装白名单中	
参数	package	包名
返回值	0	不在白名单中
	1	在白名单中
	-5000	门禁管控禁止访问
备注	无	

获取防安装黑名单应用列表(自1.2.0.9-0k4.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	Array of [String] GetBlackPackageList()	
描述	获取防安装黑名单应用列表	
参数	无	无
返回值	Array of [String]	黑名单列表
备注	门禁管控禁止访问时，返回空列表	

获取防安装白名单应用列表(自1.2.0.9-0k4.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	Array of [String] GetWhitePackageList()	
描述	获取防安装白名单应用列表	
参数	无	无
返回值	Array of [String]	白名单列表
备注	门禁管控禁止访问时，返回空列表	

清空列表(自1.2.0.9-0k3.0版本启用)

子模块	应用防安装管控	
接口类型	dbus	
原型	int ClearPackageList()	
描述	清空列表	
参数	无	无
返回值	>=0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	无	

6.2.2.2 应用防卸载管控

- **dbus信息**
  - System Bus 接口
  - dbus 服务名称：com.kylin.kysdk.applicationsec

- 路径名称：/com/kylin/kysdk/applicationsec
- Interfaces：com.kylin.kysdk.applicationsec.antiuninstall

- **子模块信息：**

设置防卸载黑白名单开关标识(自1.2.0.9-0k3.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	int SetMode(int mode)	
描述	设置防卸载黑白名单开关标识	
参数	mode	模式（0： 不开启黑白名单，1： 开启白名单，2： 开启黑名单，其它值非法）
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

获取防卸载黑白名单开关标识(自1.2.0.9-0k3.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	int GetMode()	
描述	获取防卸载黑白名单开关标识	
参数	无	无
返回值	0	未开启黑白名单
	1	开启白名单
	2	开启黑名单
	-5000	门禁管控禁止访问
备注	无	

添加一个防卸载黑名单应用(自1.2.0.9-0k3.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	int AddBlackPackage(string package)	
描述	添加一个防卸载黑名单应用	
参数	package	包名
返回值	0	成功
	-1	出错
	1	软件包已经在列表
	-5000	门禁管控禁止访问
备注	无	

添加一个防卸载白名单应用(自1.2.0.9-0k3.0版本启用)

子模块	应用防卸载管控
接口类型	dbus

原型	int AddWhitePackage(string package)	
描述	添加一个防卸载白名单应用	
参数	package	包名
返回值	0	成功
	-1	出错
	1	软件包已经在列表
	-5000	门禁管控禁止访问
备注	无	

删除一个防卸载黑名单应用(自1.2.0.9-0k3.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	int DelBlackPackage(string package)	
描述	删除一个防卸载黑名单应用	
参数	package	包名
返回值	0	不存在
	1	成功
	-1	出错
	-5000	门禁管控禁止访问
备注	无	

删除一个防卸载白名单应用(自1.2.0.9-0k3.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	int DelWhitePackage(string package)	
描述	删除一个防卸载白名单应用	
参数	package	包名
返回值	0	不存在
	1	成功
	-1	出错
	-5000	门禁管控禁止访问
备注	无	

添加防卸载黑名单应用列表(自1.2.0.9-0k3.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	int AddBlackPackagesList(Array of [String] package_list)	
描述	添加防卸载黑名单应用列表	
参数	package_list	包名列表
返回值	0	成功
	-1	出错
	1	软件包已经在列表
	-5000	门禁管控禁止访问

备注	无

添加防卸载白名单应用列表(自1.2.0.9-0k3.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	int AddWhitePackagesList(Array of [String] package_list)	
描述	添加防卸载白名单应用列表	
参数	package_list	包名列表
返回值	0	成功
	-1	出错
	1	软件包已经在列表
	-5000	门禁管控禁止访问
备注	无	

判断应用是否在防卸载黑名单中(自1.2.0.9-0k4.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	int PackageIsInBlackList(string package)	
描述	判断应用是否在防卸载黑名单中	
参数	package	包名
返回值	0	不在黑名单中
	1	在黑名单中
	-5000	门禁管控禁止访问
备注	无	

判断应用是否在防卸载白名单中(自1.2.0.9-0k4.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	int PackageIsInWhilteList(string package)	
描述	判断应用是否在防卸载白名单中	
参数	package	包名
返回值	0	不在白名单中
	1	在白名单中
	-5000	门禁管控禁止访问
备注	无	

获取防卸载黑名单应用列表(自1.2.0.9-0k4.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	Array of [String] GetBlackPackageList()	
描述	获取防卸载黑名单应用列表	
参数	无	无
返回值	Array of [String]	黑名单列表

备注	门禁管控禁止访问时，返回空列表	

获取防卸载白名单应用列表(自1.2.0.9-0k4.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	Array of [String] GetWhitePackageList()	
描述	获取防卸载白名单应用列表	
参数	无	无
返回值	Array of [String]	白名单列表
备注	门禁管控禁止访问时，返回空列表	

清空列表(自1.2.0.9-0k3.0版本启用)

子模块	应用防卸载管控	
接口类型	dbus	
原型	int ClearPackageList()	
描述	清空列表	
参数	无	无
返回值	>=0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	无	

6.2.3 应用执行控制

- **头文件路径：**

```
#include "/usr/include/kysdk/kysdk-security/libkyapplicationsec.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkyapplicationsec.so
```

- **子模块信息：**

设置应用包中所有程序均可执行(自1.2.0版本启用)

子模块	应用执行控制	
接口类型	C	
原型	int kdk_set_app_can_exec(const char* pkgname)	
描述	设置应用包中所有程序均可执行	
参数	pkgname	包名
返回值	0	成功
	-5000	门禁管控禁止访问
	非零	失败
备注	无	

设置应用包中所有程序均可不执行(自1.2.0版本启用)

子模块	应用执行控制

接口类型	C	
原型	int kdk_clear_app_can_exec(const char* pkgname)	
描述	设置应用包中所有程序均可不执行	
参数	pkgname	包名
返回值	0	成功
	-5000	门禁管控禁止访问
	非零	失败
备注	无	

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "libkyapplicationsec.h"

int main()
{
    int rc;
    char* pkgname = "qaxbrowser-safe-stable";

    rc = kdk_set_app_can_exec(pkgname);
    printf("setAppCanExec pkgname:%s rc:%d\n", pkgname, rc);

    rc = kdk_clear_app_can_exec(pkgname);
    printf("clearAppCanExec pkgname:%s rc:%d\n", pkgname, rc);

    return 0;
}
```

6.2.4 应用分级

- **dbus信息**
  - System Bus 接口
  - dbus 服务名称：com.kylin.kysdk.applicationsec
  - 路径名称：/com/kylin/kysdk/applicationsec
  - Interfaces：com.kylin.kysdk.applicationsec.classify
- **子模块信息：**

> 获取系统权限分级模式(自1.2.0.9-0k7.0版本启用)

子模块	应用分级	
接口类型	dbus	
原型	int getStatus()	
描述	获取系统权限分级模式	
参数	无	无
返回值	0	关闭kid功能和隐私保护
	1	强制模式 （开启kid及隐私保护）
	2	兼容模式 （开启kid及隐私保护）
	3	仅关闭隐私数据
	-5000	门禁管控禁止访问
备注	无	

设置状态(自1.2.0.9-0k7.0版本启用)

子模块	应用分级	
接口类型	dbus	
原型	int setStatus(int status)	
描述	设置状态	
参数	status	状态（0: 关闭kid功能和隐私保护； 1：强制模式 （开启kid及隐私保护）； 2：兼容模式 （开启kid及隐私保护）； 3：仅关闭隐私数据）
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

添加kysec配置(自1.2.0.9-0k7.24版本启用)

子模块	应用分级	
接口类型	dbus	
原型	int addKysecConf()	
描述	添加kysec配置	
参数	无	无
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

设置kysec配置(自1.2.0.9-0k7.24版本启用)

子模块	应用分级	
接口类型	dbus	
原型	int setKysecConf(String key, int value)	
描述	设置kysec配置	
参数	key	类型（kysec_kid为2）
	value	值（2为开启；3为关闭）
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

添加策略(自1.2.0.9-0k7.0版本启用)

子模块	应用分级
接口类型	dbus
原型	int addPolicy(int subType, String subject, String object, int perm)

描述	添加策略	
参数	subType	主体类型（0:应用 1：文件、程序）
	subject	策略主体
	object	策略客体
	perm	权限（0x1111重命名、删除、写、读）
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

更新策略(自1.2.0.9-0k7.0版本启用)

子模块	应用分级	
接口类型	dbus	
原型	int updatePolicy(int subType, String subject, String object, int perm)	
描述	更新策略	
参数	subType	主体类型（0:应用 1：文件、程序）
	subject	策略主体
	object	策略客体
	perm	权限（0x1111重命名、删除、写、读）
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

添加用户隐私资源(自1.2.0.9-0k7.30版本启用)

子模块	应用分级	
接口类型	dbus	
原型	int addPrivacyRes(String resource)	
描述	添加用户隐私资源	
参数	resource	资源路径
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

删除用户隐私资源(自1.2.0.9-0k7.30版本启用)

子模块	应用分级	
接口类型	dbus	
原型	int delPrivacyRes(String resource)	

描述	删除用户隐私资源	
参数	resource	资源路径
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

根据主体获取策略(自1.2.0.9-0k7.0版本启用)

子模块	应用分级	
接口类型	dbus	
原型	String getPolicyListBySub(int subType, String subject)	
描述	根据主体获取策略	
参数	subType	主体类型（0:应用 1：文件、程序）
	subject	主体类型
返回值	String	策略
备注	门禁管控禁止访问时，返回'[]'	

获取已安装应用类型列表(自1.2.0.9-0k7.7版本启用)

子模块	应用分级	
接口类型	dbus	
原型	String getInstalledAppList()	
描述	获取已安装应用类型列表	
参数	无	无
返回值	String	应用列表
备注	门禁管控禁止访问时，返回'[]'	

6.2.5 应用风险提示

6.2.5.1 应用风险提示system服务

- **dbus信息**
  - System Bus 接口
  - dbus 服务名称：com.kylin.secriskbox
  - 路径名称：/service
  - Interfaces：com.kylin.secriskbox
  - 信号：
    - 返回信号：signalResult
- **子模块信息：**

获取当前系统语言环境(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	QString getLanguage()	
描述	获取当前系统语言环境	
参数	无	无
返回值	QString	当前系统语言
备注	返回语言：中文:"zh_CN"；英文:"en_CN"；藏文:"bo_CN";	

	返回""，表示门禁管控禁止访问或没有调用方法的权限

创建提示(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int create(int type, const QString &icon, const QString &title, int &uid)	
描述	创建提示	
参数	type	提示类型（0:弹窗 1:通知）
	icon	系统图标名称
	title	提示标题
	uid	当前进程的用户uid
返回值	>0	成功：返回提示的唯一标识
	-1	创建失败
	-3	失败：设置的标题、图标至少有一项为空
	-4	失败：级别类型错误
	-6	失败：system获取session结果失败
	-7	失败：system连接session失败
	-8	失败：没有调用方法的权限
	-5000	失败：门禁管控禁止访问
备注	无	

设置弹窗阻屏(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int setDialogFull(int id)	
描述	设置弹窗阻屏	
参数	id	弹窗的唯一标识
返回值	0	成功
	-1	失败
	-6	失败：system获取session结果失败
	-7	失败：system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	仅弹窗可用。默认不是阻屏，设置该接口后阻屏。	

隐藏关闭按钮(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务

接口类型	dbus	
原型	int setDialogHideClose(int id)	
描述	隐藏关闭按钮	
参数	id	弹窗的唯一标识
返回值	0	成功
	-1	失败
	-6	失败：system获取session结果失败
	-7	失败：system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	仅弹窗可用。默认不隐藏，设置该接口后隐藏关闭按钮。	

添加内容图标(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int addContentIcon(int id,const QString &icon)	
描述	添加内容图标	
参数	id	弹窗的唯一标识
	icon	系统图标名称
返回值	0	成功
	-1	失败
	-6	失败：system获取session结果失败
	-7	失败：system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	仅弹窗可用	

添加内容(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int addContent(int id, const QString &content)	
描述	添加内容	
参数	id	提示的唯一标识
	content	内容
返回值	0	成功
	-1	失败
	-6	失败：system获取session结果失败

	-7	失败： system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	添加的第一条内容为摘要，剩余的几条为正文。	

添加按钮(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int addButton(int id, const QString &name, const QString &strActionCmd)	
描述	添加按钮	
参数	id	提示的唯一标识
	name	界面显示的按钮名称
	strActionCmd	点击按钮执行的命令
返回值	0	成功
	-1	失败
	-6	失败： system获取session结果失败
	-7	失败： system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	通知：添加界面显示按钮时，name为按钮名称， strActionCmd为点击按钮执行的命令； 添加点击窗体响应时，name为点击窗体执行的命令， strActionCmd为"default"。包含default按钮， 按钮数量最多三个。界面最多显示两个按钮， default按钮不显示。 弹窗：添加的按钮全部为界面要显示的按钮。 按钮数量最多三个。 按钮添加后将从右到左添加到弹框界面中。	

设置超时(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int setTimeout(int id, int second)	
描述	设置提示自动关闭时间	
参数	id	提示的唯一标识
	second	超时时间（单位为秒）
返回值	0	成功
	-1	失败
	-6	失败： system获取session结果失败
	-7	失败： system连接session失败

	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	弹窗：默认为30秒。>0：弹窗关闭时间；-1： 表示弹窗没有超时时间，一直显示。 通知：默认为6秒。>0：表示通知显示持续时间；0： 表示通知将直接进入通知中心，不会显示；-1： 表示通知常驻，不会收纳进通知中心	

设置弹窗鉴权用户名(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int setDialogAuth(int id, const QStringList &authList)	
描述	设置弹窗鉴权用户名	
参数	id	弹窗的唯一标识
	authList	管理员用户名列表
返回值	0	成功
	-1	失败
	-6	失败： system获取session结果失败
	-7	失败： system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	仅弹窗可用	

设置鉴权信息(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int setDialogAuthInfo(int id, const QString &info)	
描述	设置鉴权信息	
参数	id	弹窗的唯一标识
	info	鉴权信息
返回值	0	成功
	-1	失败
	-6	失败： system获取session结果失败
	-7	失败： system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	仅弹窗可用。设置鉴权控件的信息，如“允许或禁止操作，需要输入您的管理员密码”。	

更新鉴权控件(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int updateDialogAuth(int id, const QString &auth)	
描述	更新鉴权控件	
参数	id	弹窗的唯一标识
	auth	鉴权提示信息
返回值	0	成功
	-1	失败
	-6	失败： system获取session结果失败
	-7	失败： system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	仅弹窗可用。密码输入错误时， 通过此接口设置要显示的提示。如“认证失败，连续输错 4 次将锁定账户”	

设置折叠(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int setNotifyFold(int id, int state)	
描述	设置折叠	
参数	id	通知的唯一标识
	state	折叠状态（0为折叠， 1为不折叠）
返回值	0	成功
	-1	失败
	-6	失败： system获取session结果失败
	-7	失败： system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	仅通知可用。默认折叠。	

设置交互模式(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int setInteractive(int id,int mode)	
描述	设置交互模式	
参数	id	弹窗的唯一标识
	mode	交互模式（0：单次交互； 1：多次交互）

返回值	0	成功
	-1	失败
	-6	失败：system获取session结果失败
	-7	失败：system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	仅弹窗可用，默认为单次交互。	

展示提示(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int show(int id)	
描述	展示提示	
参数	id	提示的唯一标识
返回值	0	成功
	-1	失败
	-2	失败：添加按钮错误（通知可显示按钮超过两个，弹窗按钮超过三个或小于1个）
	-5	失败：没有设置内容
	-6	失败：system获取session结果失败
	-7	失败：system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	无	

关闭提示(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int close(int id)	
描述	关闭提示	
参数	id	提示的唯一标识
返回值	0	成功
	-1	失败
	-6	失败：system获取session结果失败
	-7	失败：

		system连接session失败
	-8	失败：没有调用方法的权限
	-5000	门禁管控禁止访问
备注	无	

返回信号(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示system服务	
接口类型	dbus	
原型	int signalResult(int id, int state, const QString &reserved)	
描述	弹窗关闭时发出的信号	
参数	id	提示的唯一标识
	state	关闭状态
	reserved	返回信息
返回值	无	无
备注	state -1:超时退出 0:点击窗体（用在通知）123: 某个位置按钮被点击（弹窗按钮顺序从右到左，顺序）；-2: 点击右上角关闭按钮。 reserved格式为： { "type": 0， //type为0时候， 为用户鉴权信息，user表示账户名，passwd为对应密码； "authentication": { "user": "xxx"， "passwd": "xxx" } }	

6.2.5.2 应用风险提示session服务

- **dbus信息**
  - Session Bus 接口
  - dbus 服务名称：com.kylin.secriskbox
  - 路径名称：/service
  - Interfaces：com.kylin.secriskbox
  - 信号：
    - 返回信号：signalResult
- **子模块信息：**

获取当前系统语言环境(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	QString getLanguage()	
描述	获取当前系统语言环境	
参数	无	无
返回值	QString	当前系统语言
备注	返回语言：中文:"zh_CN"；英文:"en_CN"；藏文:"bo_CN"; 返回""，表示门禁管控禁止访问	

创建提示(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int create(int type, const QString &icon, const QString &title, int &uid)	
描述	创建提示	
参数	type	提示类型（0:弹窗 1:通知）

	icon	系统图标名称
	title	提示标题
	uid	当前进程的用户uid
返回值	>0	成功：返回提示的唯一标识
	-1	创建失败
	-3	失败：设置的标题、 图标至少有一项为空
	-4	失败：级别类型错误
	-5000	失败：门禁管控禁止访问
备注	无	

设置弹窗阻屏(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int setDialogFull(int id)	
描述	设置弹窗阻屏	
参数	id	弹窗的唯一标识
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	仅弹窗可用。默认不是阻屏，设置该接口后阻屏。	

隐藏关闭按钮(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int setDialogHideClose(int id)	
描述	隐藏关闭按钮	
参数	id	弹窗的唯一标识
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	仅弹窗可用。默认不隐藏，设置该接口后隐藏关闭按钮。	

添加内容图标(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int addContentIcon(int id,const QString &icon)	
描述	添加内容图标	
参数	id	弹窗的唯一标识
	icon	系统图标名称
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问

备注	仅弹窗可用

添加内容(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int addContent(int id, const QString &content)	
描述	添加内容	
参数	id	提示的唯一标识
	content	内容
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	添加的第一条内容为摘要，剩余的几条为正文。	

添加按钮(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int addButton(int id, const QString &name, const QString &strActionCmd)	
描述	添加按钮	
参数	id	提示的唯一标识
	name	界面显示的按钮名称
	strActionCmd	点击按钮执行的命令
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	通知：添加界面显示按钮时，name为按钮名称， strActionCmd为点击按钮执行的命令。 添加点击窗体响应时，name为点击窗体执行的命令， strActionCmd为"default"；包含default按钮， 按钮数量最多三个。界面最多显示两个按钮， default按钮不显示。 弹窗：添加的按钮全部为界面要显示的按钮。 按钮数量最多三个。 按钮添加后将从右到左添加到弹框界面中。	

设置超时(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int setTimeout(int id, int second)	
描述	设置提示自动关闭时间	
参数	id	提示的唯一标识
	second	超时时间（单位为秒）
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问

备注	弹窗：默认为30秒。>0：弹窗关闭时间；-1： 表示弹窗没有超时时间，一直显示。 通知：默认为6秒。>0：表示通知显示持续时间；0： 表示通知将直接进入通知中心，不会显示；-1： 表示消息通知常驻,不会收纳进通知中心

设置弹窗鉴权用户名(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int setDialogAuth(int id, const QStringList &authList)	
描述	设置弹窗鉴权用户名	
参数	id	弹窗的唯一标识
	authList	管理员用户名列表
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	仅弹窗可用	

设置鉴权信息(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int setDialogAuthInfo(int id, const QString &info)	
描述	设置鉴权信息	
参数	id	弹窗的唯一标识
	info	鉴权信息
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	仅弹窗可用。设置鉴权控件的信息，如“允许或禁止操作，需要输入您的管理员密码”。	

更新鉴权控件(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int updateDialogAuth(int id, const QString &auth)	
描述	更新鉴权控件	
参数	id	弹窗的唯一标识
	auth	鉴权提示信息
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	仅弹窗可用。密码输入错误时， 通过此接口设置要显示的提示。如“认证失败，连续输错 4 次将锁定账户”	

设置折叠(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int setNotifyFold(int id, int state)	
描述	设置折叠	
参数	id	通知的唯一标识
	state	折叠状态（0为折叠， 1为不折叠）
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	仅通知可用。默认折叠。	

设置交互模式(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int setInteractive(int id,int mode)	
描述	设置交互模式	
参数	id	弹窗的唯一标识
	mode	交互模式（0：单次交互； 1：多次交互）
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	仅弹窗可用，默认为单次交互	

展示提示(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	int show(int id)	
描述	展示提示	
参数	id	提示的唯一标识
返回值	0	成功
	-1	失败
	-2	添加按钮错误。 通知可显示按钮超过两个， 弹窗按钮超过三个或小于1个。
	-5	没有设置内容
	-5000	门禁管控禁止访问
备注	无	

关闭提示(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务
接口类型	dbus

原型	int close(int id)	
描述	关闭提示	
参数	id	提示的唯一标识
返回值	0	成功
	-1	失败
	-5000	门禁管控禁止访问
备注	无	

返回信号(自1.2.0.9-0k6.1版本启用)

子模块	应用风险提示session服务	
接口类型	dbus	
原型	void signalResult(int id, int state, const QString &reserved)	
描述	弹窗关闭时发出的信号	
参数	id	提示的唯一标识
	state	关闭状态
	reserved	返回信息
返回值	无	无
备注	state -1:超时退出 0:点击窗体（用在通知）123: 某个位置按钮被点击（弹窗按钮顺序从右到左，顺序）；-2: 点击右上角关闭按钮。 reserved格式为： { "type": 0， //type为0时候， 为用户鉴权信息，user表示账户名，passwd为对应密码； "authentication": { "user": "xxx"， "passwd": "xxx" } }	

6.2.6 应用行为审计

- **头文件路径：**

```
#include "/usr/include/kysdk/kysdk-security/libkyapplicationsec.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkyapplicationsec.so
```

- **子模块信息：**

设置应用审计日志状态(自1.2.0.9-0k0.2版本启用)

子模块	应用行为审计	
接口类型	C	
原型	int kdk_set_app_audit_status(int apptype, int status)	
描述	设置应用审计日志状态	
参数	apptype	应用类型（1：麒麟打印 2：麒麟刻录）
	status	状态（0：关闭 1：开启）
返回值	0	成功
	-5000	门禁管控禁止访问
	非零	失败
备注	无	

获取应用审计日志状态(自1.2.0.9-0k0.2版本启用)

子模块	应用行为审计	
接口类型	C	
原型	int kdk_get_app_audit_status(int apptype)	
描述	获取应用审计日志状态	
参数	apptype	应用类型（1：麒麟打印 2：麒麟刻录）
返回值	0	关闭
	1	开启
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "libkyapplicationsec.h"

int main()
{
    int rc;
    int status = 1;
    int apptype = 2;

    rc = kdk_set_app_audit_status(apptype, status);
    printf("setAppAuditStatus apptype:%d  status:%d rc:%d\n", apptype,  status , rc);

    rc = kdk_get_app_audit_status(apptype);
    printf("getAppAuditStatus apptype:%d rc:%d\n", apptype, rc);

    return 0;
}
```

6.3 进程安全

该层设计主要为应用提供进程管控相关功能接口。

- **安装命令：**

```
$ sudo apt-get install libkysdk-processsec libkysdk-processsec-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-processsec
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKPROCESSSEC kysdk-processsec)
target_include_directories(demo PRIVATE ${KYSDKPROCESSSEC_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKPROCESSSEC_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKPROCESSSEC_LIBRARIES})
```

6.3.1 进程防杀死

封装 C 接口设置进程防杀死

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkyprocesssec.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkyprocesssec.so
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.processsec
  - 路径名称：/com/kylin/kysdk/processsec/process
  - Interfaces：com.kylin.kysdk.processsec.process
  - 信号：无
- **子模块信息**

设置进程不可被杀死(自1.2.0.9-0k1.0版本启用)

子模块	进程防杀死	
接口类型	C	
原型	int kdk_process_set_anti_killed(const char *filepath)	
描述	设置进程不可被杀死	
参数	filepath	文件路径
返回值	0	设置成功
	1	节点已存在
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

取消设置进程不可被杀死(自1.2.0.9-0k1.0版本启用)

子模块	进程防杀死	
接口类型	C	
原型	int kdk_process_cancel_anti_killed(const char *filepath)	
描述	取消设置进程不可被杀死	
参数	filepath	文件路径
返回值	0	设置成功
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

获取进程防杀死列表(自1.2.0.9-0k7.0版本启用)

子模块	进程防杀死

接口类型	C	
原型	char **kdk_process_get_anti_killed(int *count)	
描述	获取进程防杀死列表	
参数	count	节点数量
返回值	char **	进程防杀死列表， 使用完需手动释放字符串数组
备注	无	

- **其他接口类型接口：**

接口类型	接口	入参	返回值
dbus	SetAntiKilled (String filepath) ↦ (Int32 ret)	filepath 文件路径	返回值为设置是否成功；成功(0)/ 节点已存在(1)/门禁管控禁止访问(-5000)/ 失败(其他)

接口类型	接口	入参	返回值
dbus	CancelAntiKilled (String filepath) ↦ (Int32 ret)	filepath 文件路径	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetAntiKilled () ↦ (Array of [String] list)	无	list 进程防杀死列表

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "libkyprocesssec.h"

int main(int argc, char *argv[])
{
    int rc = 0;
    char **list = NULL;
    int count = 0;
    const char* path = "/usr/bin/kylin-music";

    rc = kdk_process_set_anti_killed(path);
    printf("set process anti kill rc = %d\n", rc);

    rc = kdk_process_cancel_anti_killed(path);
    printf("cancel process anti kill rc = %d\n", rc);

    list = kdk_process_get_anti_killed(&count);
    for(int i = 0; i < count; i++) {
        printf("list[%d] %s\n", i, list[i]);
        free(list[i]);
    }
    if(list) {
        free(list);
    }

    return 0;
}
```

6.3.2 进程执行控制

封装 C 接口设置进程执行控制

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkyprocesssec.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkyprocesssec.so
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.processsec
  - 路径名称：/com/kylin/kysdk/processsec/process
  - Interfaces：com.kylin.kysdk.processsec.process
  - 信号：无
- **子模块信息**

> 设置进程可执行(自1.2.0.9-0k1.0版本启用)

子模块	进程执行控制	
接口类型	C	
原型	int kdk_process_set_executable(const char *filepath)	
描述	设置进程可执行	
参数	filepath	文件路径
返回值	0	设置成功
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

> 取消设置进程可执行(自1.2.0.9-0k1.0版本启用)

子模块	进程执行控制	
接口类型	C	
原型	int kdk_process_cancel_executable(const char *filepath)	
描述	取消设置进程可执行	
参数	filepath	文件路径
返回值	0	设置成功
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

- **其他接口类型接口：**

接口类型	**接口**	**入参**	**返回值**
dbus	SetExec (String filepath) ↦ (Int32 arg_1)	filepath 文件路径	返回值为设置是否成功；成功(0)/门禁管控禁止访问(-5000)/失败(其他)

接口类型	**接口**	**入参**	**返回值**
dbus	CancelExec (String filepath) ↦ (Int32 arg_1)	filepath 文件路径	返回值为设置是否成功；成功(0)/门禁管控禁止访问(-5000)/失败(其他)

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "libkyprocesssec.h"

int main(int argc, char *argv[])
{
    int rc = 0;
    const char* path = "/usr/bin/kylin-music";

    rc = kdk_process_set_executable(path);
    printf("set binary process executable rc = %d\n", rc);

    rc = kdk_process_cancel_executable(path);
    printf("cancel binary process executable rc = %d\n", rc);

    return 0;
}
```

6.3.3 关键进程控制

封装 C 接口设置关键进程控制

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkyprocesssec.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkyprocesssec.so
```

- **DBus信息**
  - dbus 服务名称：com.kylin.processMonitor
  - 路径名称：/com/kylin/processMonitor
  - Interfaces：com.kylin.processMonitor
  - 信号：无
- **子模块信息**

> 设置关键进程(自1.2.0.9-0k7.0版本启用)

子模块	关键进程控制	
接口类型	C	
原型	int kdk_process_set_key_process(const char *filepath)	
描述	设置关键进程	
参数	filepath	文件路径
返回值	0	设置成功
	1	节点已存在
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

> 取消设置关键进程(自1.2.0.9-0k7.0版本启用)

子模块	关键进程控制
接口类型	C
原型	int kdk_process_cancel_key_process(const char *filepath)

描述	取消设置关键进程	
参数	filepath	文件路径
返回值	0	设置成功
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

获取关键进程列表(自1.2.0.9-0k7.0版本启用)

子模块	关键进程控制	
接口类型	C	
原型	char **kdk_process_get_key_process(int *count)	
描述	获取关键进程列表	
参数	count	节点数量
返回值	char **	关键进程列表， 使用完需手动释放字符串数组
备注	无	

- **其他接口类型接口：**

接口类型	接口	入参	返回值
dbus	setKeyProcess (String path) ↦ (Int32 ret)	path 文件路径	返回值为设置是否成功；成功(0)/ 节点已存在(1)/门禁管控禁止访问(-5000)/ 失败(其他)

接口类型	接口	入参	返回值
dbus	cancelKeyProcess (String path) ↦ (Int32 ret)	path 文件路径	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	getKeyProcess () ↦ (Array of [String] list)	无	list 关键进程列表

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "libkyprocesssec.h"

int main(int argc, char *argv[])
{
    int rc = 0;
    char **list = NULL;
    int count = 0;
    const char* path = "/usr/bin/kylin-music";

    rc = kdk_process_set_key_process(path);
    printf("set key process rc = %d\n", rc);

    rc = kdk_process_cancel_key_process(path);
    printf("cancel key process rc = %d\n", rc);

    list = kdk_process_get_key_process(&count);
    for(int i = 0; i < count; i++) {
        printf("list[%d] %s\n", i, list[i]);
        free(list[i]);
    }
    if(list) {
        free(list);
    }

    return 0;
}
```

6.3.4 进程联网控制

封装 C 接口设置进程联网控制

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkyprocesssec.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkyprocesssec.so
```

- **子模块信息**

> 设置进程可以联网(自1.2.0.9-0k1.0版本启用)

子模块	进程联网控制	
接口类型	C	
原型	int kdk_process_enable_networking(const char *filepath)	
描述	设置进程可以联网	
参数	filepath	文件路径
返回值	0	设置成功
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

> 设置进程不可联网(自1.2.0.9-0k1.0版本启用)

子模块	进程联网控制

接口类型	C	
原型	int kdk_process_disable_networking(const char *filepath)	
描述	设置进程不可联网	
参数	filepath	文件路径
返回值	0	设置成功
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "libkyprocesssec.h"

int main(int argc, char *argv[])
{
    int rc = 0;
    char **list = NULL;
    int count = 0;
    const char* path = "/usr/bin/apt";

    rc = kdk_process_disable_networking(path);
    printf("set process can't be connected to the internet rc = %d\n", rc);

    rc = kdk_process_enable_networking(path);
    printf("set process can be connected to the internet rc = %d\n", rc);

    return 0;
}
```

6.3.5 内核模块防卸载

封装 C 接口设置内核模块防卸载

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkyprocesssec.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkyprocesssec.so
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.processsec
  - 路径名称：/com/kylin/kysdk/processsec/kmod
  - Interfaces：com.kylin.kysdk.processsec.kmod
  - 信号：无
- **子模块信息**

> 设置内核模块不可卸载(自1.2.0.9-0k1.0版本启用)

子模块	内核模块防卸载	
接口类型	C	
原型	int kdk_kmod_set_anti_unloaded(const char *koname)	
描述	设置内核模块不可卸载	

参数	koname	内核模块名称
返回值	0	设置成功
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

取消设置内核模块不可卸载(自1.2.0.9-0k1.0版本启用)

子模块	内核模块防卸载	
接口类型	C	
原型	int kdk_kmod_cancel_anti_unloaded(const char *koname)	
描述	取消设置内核模块不可卸载	
参数	koname	内核模块名称
返回值	0	设置成功
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

设置内核模块不可卸载(自1.2.0.9-0k7.0版本启用)

子模块	内核模块防卸载	
接口类型	C	
原型	int kdk_kmod_set_anti_unloaded_by_path(const char *filepath)	
描述	设置内核模块不可卸载	
参数	filepath	内核模块文件路径
返回值	0	设置成功
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

取消设置内核模块不可卸载(自1.2.0.9-0k7.0版本启用)

子模块	内核模块防卸载	
接口类型	C	
原型	int kdk_kmod_cancel_anti_unloaded_by_path(const char *filepath)	
描述	取消设置内核模块不可卸载	
参数	filepath	内核模块文件路径
返回值	0	设置成功
	-5000	门禁管控禁止访问
	其他	设置失败
备注	无	

获取内核模块防卸载列表(自1.2.0.9-0k7.0版本启用)

子模块	内核模块防卸载
接口类型	C

原型	char **kdk_kmod_get_anti_unloaded(int *count)	
描述	获取内核模块防卸载列表	
参数	count	节点数量
返回值	char **	内核模块防卸载列表， 使用完需手动释放字符串数组
备注	无	

- **其他接口类型接口：**

接口类型	接口	入参	返回值
dbus	SetAntiUnloadedPath (String filepath) ↦ (Int32 arg_1)	filepath 文件路径	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	CancelAntiUnloadedPath (String filepath) ↦ (Int32 arg_1)	filepath 文件路径	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetAntiUnloaded () ↦ (Array of [String] list)	无	list 内核模块防卸载列表

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "libkyprocesssec.h"

int main(int argc, char *argv[])
{
    int rc = 0;
    char **list = NULL;
    int count = 0;

    const char *koname = "video";
    const char *kopath = "/usr/lib/modules/5.4.18-96-generic/kernel/drivers/acpi/video.ko";

    rc = kdk_kmod_set_anti_unloaded(koname);
    printf("set kmod anti unload by koname rc = %d\n", rc);

    rc = kdk_kmod_cancel_anti_unloaded(koname);
    printf("cancel set kmod anti unload by koname rc = %d\n", rc);

    rc = kdk_kmod_set_anti_unloaded_by_path(kopath);
    printf("set kmod anti unload by path rc = %d\n", rc);

    rc = kdk_kmod_cancel_anti_unloaded_by_path(kopath);
    printf("set kmod anti unload by path rc = %d\n", rc);

    list = kdk_kmod_get_anti_unloaded(&count);
    for(int i = 0; i < count; i++) {
        printf("list[%d] %s\n", i, list[i]);
        free(list[i]);
    }
    if(list) {
        free(list);
    }
```

```
    return 0;
}
```

6.3.6 关键进程监控

封装 DBus 接口设置关键进程异常退出监控

- **DBus信息**
  - dbus 服务名称：com.kylin.processMonitor
  - 路径名称：/com/kylin/processMonitor
  - Interfaces：com.kylin.processMonitor
  - 信号：
    - 关键进程异常退出广播信号：signal_processExit
- **子模块信息**

发送关键进程异常退出广播信号(自1.2.0.9-0k7.0版本启用)

子模块	关键进程监控	
接口类型	DBus	
原型	processExit (String path, String str) ↦ (Int32 ret)	
描述	发送关键进程异常退出广播信号 signal_processExit	
参数	path	关键进程程序路径
	str	预留参数
返回值	0	成功
	其他	失败
备注	无	

关键进程异常退出广播信号(自1.2.0.9-0k7.0版本启用)

子模块	关键进程监控	
接口类型	DBus	
原型	signal_processExit(String path, String str)	
描述	关键进程异常退出广播信号	
参数	path	关键进程程序路径
	str	预留参数
返回值	无	无
备注	无	

设置应急响应类型(自1.2.0.9-0k7.0版本启用)

子模块	关键进程监控	
接口类型	DBus	
原型	emergencyResponse (Int32 type) ↦ (Int32 ret)	
描述	设置应急响应类型	
参数	type	应急响应类型(1 锁屏；2 重启)
返回值	0	成功
	其他	失败
备注	无	

6.4 设备安全

该层设计主要为应用提供系统设备管控相关功能接口。

- **安装命令：**

```
$ sudo apt-get install kysdk-devicesec-daemon libkysdk-devicesec libkysdk-devicesec-dev
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-devicesec
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKDEVICESEC kysdk-devicesec)
target_include_directories(demo PRIVATE ${KYSDKDEVICESEC_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKDEVICESEC_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKDEVICESEC_LIBRARIES})
```

6.4.1 网卡管控

封装 C 接口设置网卡管控

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkydevicesec_netcard.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkydevicesec.so
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.devicesec
  - 路径名称：/com/kylin/kysdk/devicesec/netcard
  - Interfaces：com.kylin.kysdk.devicesec.netcard
  - 信号：无
- **子模块信息**

> 设置网卡管控状态(自2.3.0.0版本启用)

子模块	网卡管控	
接口类型	C	
原型	int kdk_device_set_netcard_status(int type, int status)	
描述	设置网卡管控状态	
参数	type	网卡类型， 参见kdk_netcard_type
	status	网卡管控状态，参见 kdk_netcard_status
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	typedef enum _kdk_netcard_type { KDK_NET_WIRELESS = 0, KDK_NET_ETHERNET,	

	} kdk_netcard_type; typedef enum _kdk_netcard_status { KDK_NET_DISABLE = 0, KDK_NET_ENABLE, } kdk_netcard_status;

获取网卡管控状态(自2.3.0.0版本启用)

子模块	网卡管控	
接口类型	C	
原型	int kdk_device_get_netcard_status(int type)	
描述	获取网卡管控状态	
参数	type	网卡类型，参见kdk_netcard_type
返回值	0	禁用
	1	启用
	-5000	门禁管控禁止访问
其他	失败	
备注	typedef enum _kdk_netcard_type { KDK_NET_WIRELESS = 0, KDK_NET_ETHERNET, } kdk_netcard_type;	

- **其他接口类型接口：**

接口类型	接口	入参	返回值
dbus	SetStatus (Int32 type, Int32 status) ↦ (Int32 ret)	type 网卡类型(0 无线网卡; 1 有线网卡) status 网卡管控状态(0 禁用；1 启用)	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetStatus (Int32 type) ↦ (Int32 status)	无	返回值为管控状态；禁用(0)/启用(1)/ 门禁管控禁止访问(-5000)/失败(其他)

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include "libkydevicesec_netcard.h"

int main(int argc, char const *argv[])
{
    int rc = 0;

    rc = kdk_device_set_netcard_status(KDK_NET_WIRELESS, KDK_NET_DISABLE);
    if(rc != 0) {
        printf("set wireless netcard disable failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_get_netcard_status(KDK_NET_WIRELESS);
    if(rc != KDK_NET_DISABLE) {
        printf("get wireless netcard is not disable, rc = %d\n", rc);
        return 1;
    }

    return 0;
}
```

6.4.2 蓝牙管控

封装 C 接口设置蓝牙管控

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkydevicesec_bluetooth.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkydevicesec.so
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.devicesec
  - 路径名称：/com/kylin/kysdk/devicesec/bluetooth
  - Interfaces：com.kylin.kysdk.devicesec.bluetooth
  - 信号：无
- **子模块信息**

> 设置蓝牙管控状态(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	int kdk_device_set_bluetooth_status(int status)	
描述	设置蓝牙管控状态	
参数	status	管控状态， 参见kdk_bluetooth_status
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	typedef enum _kdk_bluetooth_status { KDK_BLUETOOTH_DISABLE = 0,	

	KDK_BLUETOOTH_ENABLE, } kdk_bluetooth_status;

获取蓝牙管控状态(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	int kdk_device_get_bluetooth_status()	
描述	获取蓝牙管控状态	
参数	无	无
返回值	0	禁用
	1	启用
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

设置接入蓝牙设备类型的管控状态(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	int kdk_device_set_bluetooth_type_status(int type, int status)	
描述	设置接入蓝牙设备类型的管控状态	
参数	type	蓝牙设备类型， 参见kdk_bluetooth_type
	status	管控状态， 参见kdk_bluetooth_status
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败

备注	typedef enum _kdk_bluetooth_status {
	KDK_BLUETOOTH_DISABLE = 0,
	KDK_BLUETOOTH_ENABLE,
	} kdk_bluetooth_status;
	
	typedef enum _kdk_bluetooth_type {
	KDK_BLUETOOTH_TYPE_AUDIO = 0,
	KDK_BLUETOOTH_TYPE_KEYBOARDMOUSE,
	KDK_BLUETOOTH_TYPE_PHONE,
	KDK_BLUETOOTH_TYPE_COMPUTER,
	KDK_BLUETOOTH_TYPE_END
	} kdk_bluetooth_type;	

获取接入蓝牙设备类型的管控状态(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	int kdk_device_get_bluetooth_type_status(int type)	
描述	获取接入蓝牙设备类型的管控状态	
参数	type	蓝牙设备类型， 参见kdk_bluetooth_type

返回值	0	禁用
	1	启用
	-5000	门禁管控禁止访问
	其他	失败
备注	typedef enum _kdk_bluetooth_type { KDK_BLUETOOTH_TYPE_AUDIO = 0, KDK_BLUETOOTH_TYPE_KEYBOARDMOUSE, KDK_BLUETOOTH_TYPE_PHONE, KDK_BLUETOOTH_TYPE_COMPUTER, KDK_BLUETOOTH_TYPE_END } kdk_bluetooth_type;	

设置接入蓝牙设备黑白名单模式(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	int kdk_device_set_bluetooth_bwm(int mode)	
描述	设置接入蓝牙设备黑白名单模式	
参数	mode	管控模式， 参见kdk_bluetooth_mode
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	typedef enum _kdk_bluetooth_mode { KDK_BLUETOOTH_MODE_NORMAL = 0, KDK_BLUETOOTH_MODE_BLACKLIST, KDK_BLUETOOTH_MODE_WHITELIST, } kdk_bluetooth_mode;	

获取接入蓝牙设备黑白名单模式(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	kdk_device_get_bluetooth_bwm()	
描述	获取接入蓝牙设备黑白名单模式	
参数	无	无
返回值	0	普通模式
	1	黑名单模式
	2	白名单模式
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

添加接入蓝牙设备的mac地址到黑名单(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	int kdk_device_add_bluetooth_blacklist(const char *mac)	
描述	添加接入蓝牙设备的mac地址到黑名单	

参数	mac	蓝牙设备mac地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

从接入蓝牙设备黑名单删除一个蓝牙设备(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	int kdk_device_del_bluetooth_blacklist(const char *mac)	
描述	从接入蓝牙设备黑名单删除一个蓝牙设备	
参数	mac	蓝牙设备mac地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

清空接入蓝牙设备黑名单(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	int kdk_device_clear_bluetooth_blacklist()	
描述	清空接入蓝牙设备黑名单	
参数	无	无
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

获取接入蓝牙设备黑名单(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	char** kdk_device_get_bluetooth_blacklist(int *len)	
描述	获取接入蓝牙设备黑名单	
参数	len	黑名单长度
返回值	char**	黑名单列表， 使用完需手动释放字符串数组
备注	无	

添加接入蓝牙设备的mac地址到白名单(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控
接口类型	C
原型	int kdk_device_add_bluetooth_whitelist(const char *mac)
描述	添加接入蓝牙设备的mac地址到白名单

参数	mac	蓝牙设备mac地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

从接入蓝牙设备白名单删除一个蓝牙设备(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	int kdk_device_del_bluetooth_whitelist(const char *mac)	
描述	从接入蓝牙设备白名单删除一个蓝牙设备	
参数	mac	蓝牙设备mac地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

清空接入蓝牙设备白名单(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	int kdk_device_clear_bluetooth_whitelist()	
描述	清空接入蓝牙设备白名单	
参数	无	无
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

获取接入蓝牙设备白名单(自1.2.0.9-0k7.0版本启用)

子模块	蓝牙管控	
接口类型	C	
原型	char** kdk_device_get_bluetooth_whitelist(int *len)	
描述	获取接入蓝牙设备白名单	
参数	len	白名单长度
返回值	char**	白名单列表， 使用完需手动释放字符串数组
备注	无	

- **其他接口类型接口：**

接口类型	接口	入参	返回值
dbus	SetStatus (Int32 status) ↦ (Int32 ret)	status 蓝牙管控状态(0 禁用；1 启用)	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值

dbus	GetStatus () ↦ (Int32 status)	无	返回值为管控状态；禁用(0)/启用(1)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	SetTypeStatus (Int32 type, Int32 status) ↦ (Int32 ret)	type 蓝牙设备类型(0 audio; 1 keyboard&mouse; 2 phone; 3 computer) status 蓝牙管控状态(0 禁用；1 启用)	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetTypeStatus (Int32 type) ↦ (Int32 status)	type 蓝牙设备类型(0 audio; 1 keyboard&mouse; 2 phone; 3 computer)	返回值为管控状态；禁用(0)/启用(1)/ 门禁管控禁止访问(-5000)/失败(其他))

接口类型	接口	入参	返回值
dbus	SetMode (Int32 mode) ↦ (Int32 ret)	mode 管控模式(0 普通模式; 1 黑名单模式; 2 白名单模式)	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetMode () ↦ (Int32 mode)	无	返回值为管控模式；普通模式(0)/ 黑名单模式(1)/白名单模式(2)/失败(其他)

接口类型	接口	入参	返回值
dbus	AddBlacklist (String mac) ↦ (Int32 ret)	mac 蓝牙设备mac地址	返回值为添加是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	DelBlacklist (String mac) ↦ (Int32 ret)	mac 蓝牙设备mac地址	返回值为删除是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	ClearBlacklist () ↦ (Int32 ret)	无	返回值为清空是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetBlacklist () ↦ (Int32 len, Array of [String] list)	无	len 黑名单长度 list 黑名单列表

接口类型	接口	入参	返回值
dbus	AddWhitelist (String mac) ↦ (Int32 ret)	mac 蓝牙设备mac地址	返回值为添加是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	DelWhitelist (String mac) ↦ (Int32 ret)	mac 蓝牙设备mac地址	返回值为删除是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	ClearWhitelist () ↦ (Int32 ret)	无	返回值为清空是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetWhitelist () ↦ (Int32 len, Array of [String] list)	无	len 白名单长度 list 白名单列表

- **示例代码：**

```c
#----------------------------C语言示例----------------------------
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "libkydevicesec_bluetooth.h"

int main(int argc, char const *argv[])
{
    int rc = 0;
    const char *mac = "55:bd:76:2c:be:c2";
    char **list = NULL;
    int len = 0;

    rc = kdk_device_set_bluetooth_status(KDK_BLUETOOTH_DISABLE);
    if(rc != 0) {
        printf("set bluetooth status disable failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_get_bluetooth_status();
    if(rc != KDK_BLUETOOTH_DISABLE) {
        printf("get bluetooth status is not disable, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_set_bluetooth_type_status(KDK_BLUETOOTH_TYPE_AUDIO, KDK_BLUETOOTH_DISABLE);
    if(rc != 0) {
        printf("set audio type bluetooth status disable failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_get_bluetooth_type_status(KDK_BLUETOOTH_TYPE_AUDIO);
    if(rc != KDK_BLUETOOTH_DISABLE) {
        printf("get audio type bluetooth status is not disable, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_set_bluetooth_bwm(KDK_BLUETOOTH_MODE_BLACKLIST);
    if(rc != 0) {
        printf("set bluetooth blacklist mode failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_get_bluetooth_bwm();
    if(rc != KDK_BLUETOOTH_MODE_BLACKLIST) {
        printf("get bluetooth mode is not blacklist, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_add_bluetooth_blacklist(mac);
    if(rc != 0) {
        printf("add bluetooth blacklist %s failed, rc = %d\n", mac, rc);
        return 1;
    }

    list = kdk_device_get_bluetooth_blacklist(&len);
    for(int i = 0; i < len; i++) {
        printf("get blacklist[%d] %s\n", i, list[i]);
        free(list[i]);
    }
    if(list) {
        free(list);
    }

    rc = kdk_device_del_bluetooth_blacklist(mac);
    if(rc != 0) {
        printf("del bluetooth blacklist %s failed, rc = %d\n", mac, rc);
        return 1;
```

```
    }

    rc = kdk_device_clear_bluetooth_blacklist();
    if(rc != 0) {
        printf("clear bluetooth blacklist failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_add_bluetooth_whitelist(mac);
    if(rc != 0) {
        printf("add bluetooth whitelist %s failed, rc = %d\n", mac, rc);
        return 1;
    }

    list = kdk_device_get_bluetooth_whitelist(&len);
    for(int i = 0; i < len; i++) {
        printf("get whitelist[%d] %s\n", i, list[i]);
        free(list[i]);
    }
    if(list) {
        free(list);
    }

    rc = kdk_device_del_bluetooth_whitelist(mac);
    if(rc != 0) {
        printf("del bluetooth whitelist %s failed, rc = %d\n", mac, rc);
        return 1;
    }

    rc = kdk_device_clear_bluetooth_whitelist();
    if(rc != 0) {
        printf("clear bluetooth whitelist failed, rc = %d\n", rc);
        return 1;
    }

    return 0;
}
```

6.4.3 光驱管控

封装 C 接口设置光驱管控

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkydevicesec_cdrom.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkydevicesec.so
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.devicesec
  - 路径名称：/com/kylin/kysdk/devicesec/cdrom
  - Interfaces：com.kylin.kysdk.devicesec.cdrom
  - 信号：无
- **子模块信息**

> 设置光驱管控状态(自2.3.0.0版本启用)

子模块	光驱管控	
接口类型	C	
原型	int kdk_device_set_cdrom_status(int status)	
描述	设置光驱管控状态	
参数	status	光驱管控状态，参见

		kdk_cdrom_status
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败

备注	typedef enum _kdk_cdrom_status {
	KDK_CDROM_ENABLE = 1, //启用刻录机
	KDK_CDROM_DISABLE = 2, //禁用刻录机
	KDK_CDROM_RO = 5, //只读刻录机
	} kdk_cdrom_status;	

获取光驱管控状态(自2.3.0.0版本启用)

子模块	光驱管控	
接口类型	C	
原型	int kdk_device_get_cdrom_status()	
描述	获取光驱管控状态	
参数	无	无
返回值	1	启用
	2	禁用
	5	只读
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

- **其他接口类型接口：**

接口类型	接口	入参	返回值
dbus	SetStatus (Int32 status) ↦ (Int32 ret)	status 光驱管控状态(1 启用；2 禁用；5 只读)	返回值为设置是否成功；成功(0)/门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetStatus () ↦ (Int32 status)	无	返回值为管控状态；启用(1)/禁用(2)/只读(5)/门禁管控禁止访问(-5000)/失败(其他)

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include "libkydevicesec_cdrom.h"

int main(int argc, char const *argv[])
{
    int rc = 0;

    rc = kdk_device_set_cdrom_status(KDK_CDROM_DISABLE);
    if(rc != 0) {
        printf("set cd disable failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_get_cdrom_status();
    if(rc != KDK_CDROM_DISABLE) {
        printf("get cd status is not disable, rc = %d\n", rc);
        return 1;
    }

    return 0;
}
```

6.4.4 打印机管控

封装 C 接口设置打印机管控

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkydevicesec_printer.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkydevicesec.so
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.devicesec
  - 路径名称：/com/kylin/kysdk/devicesec/printer
  - Interfaces：com.kylin.kysdk.devicesec.printer
  - 信号：无
- **子模块信息**

> 设置打印机管控状态(自2.3.0.0版本启用)

子模块	打印机管控	
接口类型	C	
原型	int kdk_device_set_printer_status(int status)	
描述	设置打印机管控状态	
参数	status	打印机管控状态，参见 kdk_printer_status
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	typedef enum _kdk_printer_status { KDK_PRINTER_ENABLE = 1,	

	KDK_PRINTER_DISABLE = 2, } kdk_printer_status;

获取打印机管控状态(自2.3.0.0版本启用)

子模块	打印机管控	
接口类型	C	
原型	int kdk_device_get_printer_status()	
描述	获取打印机管控状态	
参数	无	无
返回值	1	启用
	2	禁用
	-5000	门禁管控禁止访问
	其他	失败
	备注	无

设置USB打印机管控状态(自2.4.0.0版本启用)

子模块	打印机管控	
接口类型	C	
原型	int kdk_device_set_usb_printer_status(int status)	
描述	设置USB打印机管控状态	
参数	status	打印机管控状态，参见kdk_printer_status
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	typedef enum _kdk_printer_status { KDK_PRINTER_ENABLE = 1, KDK_PRINTER_DISABLE = 2, } kdk_printer_status;	

获取USB打印管控状态(自2.4.0.0版本启用)

子模块	打印机管控	
接口类型	C	
原型	int kdk_device_get_usb_printer_status()	
描述	获取USB打印管控状态	
参数	无	无
返回值	1	启用
	2	禁用
	-5000	门禁管控禁止访问
	其他	失败
	备注	无

- **其他接口类型接口：**

接口类型	接口	入参	返回值
dbus	SetStatus (Int32 status) ↦ (Int32 ret)	status 打印机管控状态(1 启用；2 禁用)	返回值为设置是否成功；成功(0)/

			门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetStatus () ↦ (Int32 status)	无	返回值为打印机管控状态；启用(1)/禁用(2)/门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	SetUsbStatus (Int32 status) ↦ (Int32 ret)	status USB打印机管控状态(1 启用；2 禁用)	返回值为设置是否成功；成功(0)/门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetUsbStatus () ↦ (Int32 status)	无	返回值为USB打印机管控状态；启用(1)/禁用(2)/门禁管控禁止访问(-5000)/失败(其他)

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include "libkydevicesec_printer.h"

int main(int argc, char const *argv[])
{
    int rc = 0;

    rc = kdk_device_set_printer_status(KDK_PRINTER_DISABLE);
    if(rc != 0) {
        printf("set printer disable failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_get_printer_status();
    if(rc != KDK_PRINTER_DISABLE) {
        printf("get printer status is not disable, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_set_usb_printer_status(KDK_PRINTER_DISABLE);
    if(rc != 0) {
        printf("set usb printer disable failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_get_usb_printer_status();
    if(rc != KDK_PRINTER_DISABLE) {
        printf("get usb printer status is not disable, rc = %d\n", rc);
        return 1;
    }

    return 0;
}
```

6.4.5 WIFI管控

封装 C 接口设置WIFI管控

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkydevicesec_wifi.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkydevicesec.so
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.devicesec
  - 路径名称：/com/kylin/kysdk/devicesec/wlan/wifi
  - Interfaces：com.kylin.kysdk.devicesec.wlan.wifi
  - 信号：无
- **子模块信息**

设置WIFI管控状态(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	int kdk_device_set_wireless_wifi_status(int status)	
描述	设置WIFI管控状态	
参数	status	管控状态， 参见kdk_wifi_status
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	typedef enum _kdk_wifi_status { KDK_WIFI_DISABLE = 0, KDK_WIFI_ENABLE, } kdk_wifi_status;	

获取WIFI管控状态(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	int kdk_device_get_wireless_wifi_status()	
描述	获取WIFI管控状态	
参数	无	无
返回值	0	禁用
	1	启用
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

设置WIFI管控黑白名单模式(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	int kdk_device_set_wireless_wifi_bwm(int mode)	
描述	设置WIFI管控黑白名单模式	
参数	mode	管控模式， 参见kdk_wifi_mode
返回值	0	成功
	-5000	门禁管控禁止访问

	其他	失败
备注	typedef enum _kdk_wifi_mode { KDK_WIFI_MODE_NORMAL = 0, KDK_WIFI_MODE_BLACKLIST, KDK_WIFI_MODE_WHITELIST, } kdk_wifi_mode;	

获取WIFI管控黑白名单模式(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	kdk_device_get_wireless_wifi_bwm()	
描述	获取WIFI管控黑白名单模式	
参数	无	无
返回值	0	普通模式
	1	黑名单模式
	2	白名单模式
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

设置WIFI管控黑名单(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	int kdk_device_set_wireless_wifi_blacklist(const char *bssid)	
描述	设置WIFI管控黑名单	
参数	bssid	无线网络的bssid地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

删除WIFI管控黑名单节点(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	int kdk_device_del_wireless_wifi_blacklist(const char *bssid)	
描述	删除WIFI管控黑名单节点	
参数	bssid	无线网络的bssid地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

清空WIFI管控黑名单(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	int kdk_device_clear_wireless_wifi_blacklist()	
描述	清空WIFI管控黑名单	
参数	无	无
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

获取WIFI管控黑名单(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	char** kdk_device_get_wireless_wifi_blacklist(int *len)	
描述	获取WIFI管控黑名单	
参数	len	黑名单长度
返回值	char**	黑名单列表， 使用完需手动释放字符串数组
备注	无	

设置WIFI管控白名单(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	int kdk_device_add_wireless_wifi_whitelist(const char *bssid)	
描述	设置WIFI管控白名单	
参数	bssid	无线网络的bssid地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

删除WIFI管控白名单节点(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	int kdk_device_del_wireless_wifi_whitelist(const char *bssid)	
描述	删除WIFI管控白名单节点	
参数	bssid	无线网络的bssid地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

清空WIFI管控白名单(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	int kdk_device_clear_wireless_wifi_whitelist()	
描述	清空WIFI管控白名单	
参数	无	无
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

获取WIFI管控白名单(自1.2.0.9-0k7.0版本启用)

子模块	WIFI管控	
接口类型	C	
原型	char** kdk_device_get_wireless_wifi_whitelist(int *len)	
描述	获取WIFI管控白名单	
参数	len	白名单长度
返回值	char**	白名单列表， 使用完需手动释放字符串数组
备注	无	

- **其他接口类型接口：**

接口类型	接口	入参	返回值
dbus	SetStatus (Int32 status) ↦ (Int32 ret)	status WIFI管控状态(0 禁用；1 启用)	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetStatus () ↦ (Int32 status)	无	返回值为管控状态；禁用(0)/启用(1)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	SetMode (Int32 mode) ↦ (Int32 ret)	mode 管控模式(0 普通模式; 1 黑名单模式; 2 白名单模式)	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetMode () ↦ (Int32 mode)	无	返回值为管控模式；普通模式(0)/ 黑名单模式(1)/白名单模式(2)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	SetBlacklist (String bssid) ↦ (Int32 ret)	bssid 无线网络的bssid地址	返回值为添加是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	DelBlacklist (String bssid) ↦ (Int32 ret)	bssid 无线网络的bssid地址	返回值为删除是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	ClearBlacklist () ↦ (Int32 ret)	无	返回值为清空是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetBlacklist () ↦ (Int32 len, Array of [String] bssidlist)	无	len 黑名单长度 list 黑名单列表

接口类型	接口	入参	返回值
dbus	SetWhitelist (String bssid) ↦ (Int32 ret)	bssid 无线网络的bssid地址	返回值为添加是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	DelWhitelist (String bssid) ↦ (Int32 ret)	bssid 无线网络的bssid地址	返回值为删除是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	ClearWhitelist () ↦ (Int32 ret)	无	返回值为清空是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetWhitelist () ↦ (Int32 len, Array of [String] bssidlist)	无	len 白名单长度 list 白名单列表

- **示例代码：**

```c
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include "libkydevicesec_wifi.h"

int main()
{
    int rc = 0;
    const char *bssid = "36:ec:ea:f1:c2:f0";
    char **list = NULL;
    int len = 0;

    rc = kdk_device_set_wireless_wifi_status(KDK_WIFI_DISABLE);
    if(rc != 0) {
        printf("set wifi status disable failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_get_wireless_wifi_status();
    if(rc != KDK_WIFI_DISABLE) {
        printf("get wifi status is not disable, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_set_wireless_wifi_bwm(KDK_WIFI_MODE_BLACKLIST);
    if(rc != 0) {
        printf("set wifi blacklist mode failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_get_wireless_wifi_bwm();
    if(rc != KDK_WIFI_MODE_BLACKLIST) {
        printf("get wifi mode is not blacklist, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_set_wireless_wifi_blacklist(bssid);
    if(rc != 0) {
        printf("set wifi blacklist %s failed, rc = %d\n", bssid, rc);
        return 1;
    }

    list = kdk_device_get_wireless_wifi_blacklist(&len);
    for(int i = 0; i < len; i++) {
        printf("get blacklist[%d] %s\n", i, list[i]);
        free(list[i]);
    }
    if(list) {
        free(list);
    }

    rc = kdk_device_del_wireless_wifi_blacklist(bssid);
    if(rc != 0) {
        printf("del wifi blacklist %s failed, rc = %d\n", bssid, rc);
        return 1;
    }

    rc = kdk_device_clear_wireless_wifi_blacklist();
    if(rc != 0) {
        printf("clear wifi blacklist failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_set_wireless_wifi_whitelist(bssid);
    if(rc != 0) {
        printf("set wifi whitelist %s failed, rc = %d\n", bssid, rc);
        return 1;
```

```
    }

    list = kdk_device_get_wireless_wifi_whitelist(&len);
    for(int i = 0; i < len; i++) {
        printf("get whitelist[%d] %s\n", i, list[i]);
        free(list[i]);
    }
    if(list) {
        free(list);
    }

    rc = kdk_device_del_wireless_wifi_whitelist(bssid);
    if(rc != 0) {
        printf("del wifi whitelist %s failed, rc = %d\n", bssid, rc);
        return 1;
    }

    rc = kdk_device_clear_wireless_wifi_whitelist();
    if(rc != 0) {
        printf("clear wifi whitelist failed, rc = %d\n", rc);
        return 1;
    }

    return 0;
}
```

6.4.6 热点管控

封装 C 接口设置热点管控

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkydevicesec_ap.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkydevicesec.so
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.devicesec
  - 路径名称：/com/kylin/kysdk/devicesec/wlan/ap
  - Interfaces：com.kylin.kysdk.devicesec.wlan.ap
  - 信号：无
- **子模块信息**

> 设置热点管控状态(自1.2.0.9-0k7.0版本启用)

子模块	热点管控	
接口类型	C	
原型	int kdk_device_set_wireless_ap_status(int status)	
描述	设置热点管控状态	
参数	status	管控状态， 参见kdk_ap_status
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	typedef enum _kdk_ap_status { KDK_AP_DISABLE = 0, KDK_AP_ENABLE, } kdk_ap_status;	

获取热点管控状态(自1.2.0.9-0k7.0版本启用)

子模块	热点管控	
接口类型	C	
原型	int kdk_device_get_wireless_ap_status()	
描述	获取热点管控状态	
参数	无	无
返回值	0	禁用
	1	启用
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

设置热点管控黑白名单模式(自1.2.0.9-0k7.0版本启用)

子模块	热点管控	
接口类型	C	
原型	int kdk_device_set_wireless_ap_bwm(int mode)	
描述	设置热点管控黑白名单模式	
参数	mode	管控模式， 参见kdk_ap_mode
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	typedef enum _kdk_ap_mode { KDK_AP_MODE_NORMAL = 0, KDK_AP_MODE_BLACKLIST, KDK_AP_MODE_WHITELIST, } kdk_ap_mode;	

获取热点管控黑白名单模式(自1.2.0.9-0k7.0版本启用)

子模块	热点管控	
接口类型	C	
原型	kdk_device_get_wireless_ap_bwm()	
描述	获取热点管控黑白名单模式	
参数	无	无
返回值	0	普通模式
	1	黑名单模式
	2	白名单模式
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

设置热点管控黑名单(自1.2.0.9-0k7.0版本启用)

子模块	热点管控
接口类型	C

原型	int kdk_device_set_wireless_ap_blacklist(const char *mac)	
描述	设置热点管控黑名单	
参数	mac	热点mac地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

删除热点管控黑名单节点(自1.2.0.9-0k7.0版本启用)

子模块	热点管控	
接口类型	C	
原型	int kdk_device_del_wireless_ap_blacklist(const char *mac)	
描述	删除热点管控黑名单节点	
参数	mac	热点mac地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

清空热点管控黑名单(自1.2.0.9-0k7.0版本启用)

子模块	热点管控	
接口类型	C	
原型	int kdk_device_clear_wireless_ap_blacklist()	
描述	清空热点管控黑名单	
参数	无	无
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

获取热点管控黑名单(自1.2.0.9-0k7.0版本启用)

子模块	热点管控	
接口类型	C	
原型	char** kdk_device_get_wireless_ap_blacklist(int *len)	
描述	获取热点管控黑名单	
参数	len	黑名单长度
返回值	char**	黑名单列表， 使用完需手动释放字符串数组
	-5000	门禁管控禁止访问
备注	无	

设置热点管控白名单(自1.2.0.9-0k7.0版本启用)

子模块	热点管控

接口类型	C	
原型	int kdk_device_set_wireless_ap_whitelist(const char *mac)	
描述	设置热点管控白名单	
参数	mac	热点mac地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

删除热点管控白名单节点(自1.2.0.9-0k7.0版本启用)

子模块	热点管控	
接口类型	C	
原型	int kdk_device_del_wireless_ap_whitelist(const char *mac)	
描述	删除热点管控白名单节点	
参数	mac	热点mac地址
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

清空热点管控白名单(自1.2.0.9-0k7.0版本启用)

子模块	热点管控	
接口类型	C	
原型	int kdk_device_clear_wireless_ap_whitelist()	
描述	清空热点管控白名单	
参数	无	无
返回值	0	成功
	-5000	门禁管控禁止访问
	其他	失败
备注	无	

获取热点管控白名单(自1.2.0.9-0k7.0版本启用)

子模块	热点管控	
接口类型	C	
原型	char** kdk_device_get_wireless_ap_whitelist(int *len)	
描述	获取热点管控白名单	
参数	len	白名单长度
返回值	char**	白名单列表， 使用完需手动释放字符串数组
	-5000	门禁管控禁止访问
备注	无	

- **其他接口类型接口：**

接口类型	接口	入参	返回值
dbus	SetStatus (Int32 status) ↦ (Int32 ret)	status 热点管控状态(0 禁用；1 启用)	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetStatus () ↦ (Int32 status)	无	返回值为管控状态；禁用(0)/启用(1)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	SetMode (Int32 mode) ↦ (Int32 ret)	mode 管控模式(0 普通模式; 1 黑名单模式; 2 白名单模式)	返回值为设置是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetMode () ↦ (Int32 mode)	无	返回值为管控模式；普通模式(0)/ 黑名单模式(1)/白名单模式(2)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	SetBlacklist (String mac) ↦ (Int32 ret)	mac 热点mac地址	返回值为添加是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	DelBlacklist (String mac) ↦ (Int32 ret)	mac 热点mac地址	返回值为删除是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	ClearBlacklist () ↦ (Int32 ret)	无	返回值为清空是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetBlacklist () ↦ (Int32 len, Array of [String] maclist)	无	len 黑名单长度 list 黑名单列表

接口类型	接口	入参	返回值
dbus	SetWhitelist (String mac) ↦ (Int32 ret)	mac 热点mac地址	返回值为添加是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	DelWhitelist (String mac) ↦ (Int32 ret)	mac 热点mac地址	返回值为删除是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	ClearWhitelist () ↦ (Int32 ret)	无	返回值为清空是否成功；成功(0)/ 门禁管控禁止访问(-5000)/失败(其他)

接口类型	接口	入参	返回值
dbus	GetWhitelist () ↦ (Int32 len, Array of [String] maclist)	无	len 白名单长度 list 白名单列表

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include "libkydevicesec_ap.h"

int main()
{
    int rc = 0;
    const char *mac = "36:ec:ea:f1:c2:f0";
    char **list = NULL;
    int len = 0;

    rc = kdk_device_set_wireless_ap_status(KDK_AP_DISABLE);
    if(rc != 0) {
        printf("set ap status disable failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_get_wireless_ap_status();
    if(rc != KDK_AP_DISABLE) {
        printf("get ap status is not disable, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_set_wireless_ap_bwm(KDK_AP_MODE_BLACKLIST);
    if(rc != 0) {
        printf("set ap blacklist mode failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_get_wireless_ap_bwm();
    if(rc != KDK_AP_MODE_BLACKLIST) {
        printf("get ap mode is not blacklist, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_set_wireless_ap_blacklist(mac);
    if(rc != 0) {
        printf("set ap blacklist %s failed, rc = %d\n", mac, rc);
        return 1;
    }

    list = kdk_device_get_wireless_ap_blacklist(&len);
    for(int i = 0; i < len; i++) {
        printf("get blacklist[%d] %s\n", i, list[i]);
        free(list[i]);
    }
    if(list) {
        free(list);
    }

    rc = kdk_device_del_wireless_ap_blacklist(mac);
    if(rc != 0) {
        printf("del ap blacklist %s failed, rc = %d\n", mac, rc);
        return 1;
    }

    rc = kdk_device_clear_wireless_ap_blacklist();
    if(rc != 0) {
        printf("clear ap blacklist failed, rc = %d\n", rc);
        return 1;
    }

    rc = kdk_device_set_wireless_ap_whitelist(mac);
    if(rc != 0) {
        printf("set ap whitelist %s failed, rc = %d\n", mac, rc);
        return 1;
```

```
    }

    list = kdk_device_get_wireless_ap_whitelist(&len);
    for(int i = 0; i < len; i++) {
        printf("get whitelist[%d] %s\n", i, list[i]);
        free(list[i]);
    }
    if(list) {
        free(list);
    }

    rc = kdk_device_del_wireless_ap_whitelist(mac);
    if(rc != 0) {
        printf("del ap whitelist %s failed, rc = %d\n", mac, rc);
        return 1;
    }

    rc = kdk_device_clear_wireless_ap_whitelist();
    if(rc != 0) {
        printf("clear ap whitelist failed, rc = %d\n", rc);
        return 1;
    }

    return 0;
}
```

6.4.7 桌管外设管控

- **安装命令：**

```
$ sudo apt install libkysdk-desktopctrl

# 若libkysdk-desktopctrl>=2.4.0.0-0k0.2，或在2203系统上libkysdk-desktopctrl>=1.2.0.9-0k7.37build2203v1.2，或在2303系统上libkysdk-desktc
$ sudo apt install kysdk-devicesec-daemon kysdk-security-daemon libkysdk-devicesec
```

6.4.7.1 USB存储设备管控

封装 DBus 接口设置USB存储设备管控（该功能需麒麟授权，如果需要使用，请联系麒麟技术开发人员）

- **服务说明：**
  自libkysdk-desktopctrl 1.2.0.9-0k1.2版本起，取消了桌管外设管控服务自启动，若需使用相关功能请启动以下服务：

```
# 桌面管控设备插拔监控服务：
$ sudo systemctl enable kylin-mdm-monitor.service
$ sudo systemctl start kylin-mdm-monitor.service

# USB存储设备挂载监控服务：
$ systemctl enable --global storage-mount-monitor.service
$ systemctl start --global storage-mount-monitor.service

# USB存储设备管控DBus服务：
$ sudo systemctl enable dbus-com.kylin.mdm.storagecontrol.service
$ sudo systemctl start dbus-com.kylin.mdm.storagecontrol.service
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.StorageControl
  - 路径名称：/com/kylin/kysdk/StorageControl
  - Interfaces：com.kylin.kysdk.StorageControl
  - 信号：无
- **子模块信息**

> 设置USB存储设备管控状态(自1.2.0.8版本启用)

子模块	USB存储设备管控
接口类型	DBus

原型	SetStatus (Int32 status) ↦ (Int32 arg_1)	
描述	设置USB存储设备管控状态	
参数	status	管控状态(0 禁用；1 只读；2 启用)
返回值	0	成功
	其他	失败
备注	无	

获取USB存储设备管控状态(自1.2.0.8版本启用)

子模块	USB存储设备管控	
接口类型	DBus	
原型	GetStatus () ↦ (Int32 arg_0)	
描述	获取USB存储设备管控状态	
参数	无	无
返回值	0	禁用
	1	只读
	2	启用
	其他	失败
备注	无	

6.4.7.2 USB设备黑白名单管控

封装 DBus 接口设置USB设备黑白名单管控（该功能需麒麟授权，如果需要使用，请联系麒麟技术开发人员）

- **服务说明:**
  自libkysdk-desktopctrl 1.2.0.9-0k1.2版本起，取消了桌管外设管控服务自启动，若需使用相关功能请启动以下服务：

```
# 桌面管控设备插拔监控服务:
$ sudo systemctl enable kylin-mdm-monitor.service
$ sudo systemctl start kylin-mdm-monitor.service

# USB设备黑白名单管控DBus服务:
$ sudo systemctl enable dbus-com.kylin.mdm.listcontrol.service
$ sudo systemctl start dbus-com.kylin.mdm.listcontrol.service
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.ListControl
  - 路径名称：/com/kylin/kysdk/ListControl
  - Interfaces：com.kylin.kysdk.ListControl
  - 信号：无
- **子模块信息**

设置USB设备黑名单(自1.2.0.8版本启用)

子模块	USB设备黑白名单管控	
接口类型	DBus	
原型	SetBlackList (String pid, String vid) ↦ (Int32 arg_2)	
描述	设置USB设备黑名单	
参数	pid	设备产品ID(Product ID)
	vid	设备生产厂商ID(Vendor ID)
返回值	0	成功
	其他	失败

备注	无

设置USB设备白名单(自1.2.0.8版本启用)

子模块	USB设备黑白名单管控	
接口类型	DBus	
原型	SetWhiteList (String pid, String vid) ↦ (Int32 arg_2)	
描述	设置USB设备白名单	
参数	pid	设备产品ID(Product ID)
	vid	设备生产厂商ID(Vendor ID)
返回值	0	成功
	其他	失败
备注	无	

删除USB设备黑名单(自1.2.0.8版本启用)

子模块	USB设备黑白名单管控	
接口类型	DBus	
原型	DelBlackList (String pid, String vid) ↦ (Int32 arg_2)	
描述	删除USB设备黑名单	
参数	pid	设备产品ID(Product ID)
	vid	设备生产厂商ID(Vendor ID)
返回值	0	成功
	其他	失败
备注	无	

删除USB设备白名单(自1.2.0.8版本启用)

子模块	USB设备黑白名单管控	
接口类型	DBus	
原型	DelWhiteList (String pid, String vid) ↦ (Int32 arg_2)	
描述	删除USB设备白名单	
参数	pid	设备产品ID(Product ID)
	vid	设备生产厂商ID(Vendor ID)
返回值	0	成功
	其他	失败
备注	无	

获取USB设备黑名单列表(自1.2.0.8版本启用)

子模块	USB设备黑白名单管控	
接口类型	DBus	
原型	GetBlackList () ↦ (Array of [String] list)	
描述	获取USB设备黑名单列表	
参数	无	无
返回值	list	黑名单列表
备注	无	

获取USB设备白名单列表(自1.2.0.8版本启用)

子模块	USB设备黑白名单管控	
接口类型	DBus	
原型	GetWhiteList () ↦ (Array of [String] list)	
描述	获取USB设备白名单列表	
参数	无	无
返回值	list	白名单列表
备注	无	

清空USB设备黑名单(自1.2.0.8版本启用)

子模块	USB设备黑白名单管控	
接口类型	DBus	
原型	ClearBlackList () ↦ (Int32 arg_0)	
描述	清空USB设备黑名单	
参数	无	无
返回值	0	成功
	其他	失败
备注	无	

清空USB设备白名单(自1.2.0.8版本启用)

子模块	USB设备黑白名单管控	
接口类型	DBus	
原型	ClearWhiteList () ↦ (Int32 arg_0)	
描述	清空USB设备白名单	
参数	无	无
返回值	0	成功
	其他	失败
备注	无	

6.4.7.3 网卡管控

封装 DBus 接口设置无线网卡管控（该功能需麒麟授权，如果需要使用，请联系麒麟技术开发人员）

- **服务说明:**
  自libkysdk-desktopctrl 1.2.0.9-0k1.2版本起，取消了桌管外设管控服务自启动，若需使用相关功能请启动以下服务：

```
# 桌面管控设备插拔监控服务:
$ sudo systemctl enable kylin-mdm-monitor.service
$ sudo systemctl start kylin-mdm-monitor.service

# 网卡管控DBus服务:
$ sudo systemctl enable dbus-com.kylin.mdm.netcardcontrol.service
$ sudo systemctl start dbus-com.kylin.mdm.netcardcontrol.service
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.netcardcontrol
  - 路径名称：/com/kylin/kysdk/netcardcontrol
  - Interfaces：com.kylin.kysdk.netcardcontrol
  - 信号：无
- **子模块信息**

设置无线网卡管控状态(自1.2.0.8版本启用)

子模块	网卡管控	
接口类型	DBus	
原型	SetNetcardStatus (Int32 status) ↦ (Int32 arg_1)	
描述	设置无线网卡管控状态	
参数	status	管控状态(0 禁用；1 启用)
返回值	0	成功
	其他	失败
备注	无	

获取无线网卡管控状态(自1.2.0.8版本启用)

子模块	网卡管控	
接口类型	DBus	
原型	GetNetcardStatus () ↦ (Int32 arg_0)	
描述	获取无线网卡管控状态	
参数	无	无
返回值	0	禁用
	1	启用
	其他	失败
备注	无	

6.4.7.4 光驱管控

封装 DBus 接口设置光驱管控（该功能需麒麟授权，如果需要使用，请联系麒麟技术开发人员）

- **服务说明:**
  自libkysdk-desktopctrl 1.2.0.9-0k1.2版本起，取消了桌管外设管控服务自启动，若需使用相关功能请启动以下服务：

```
# 桌面管控设备插拔监控服务:
$ sudo systemctl enable kylin-mdm-monitor.service
$ sudo systemctl start kylin-mdm-monitor.service

# 光驱管控DBus服务:
$ sudo systemctl enable dbus-com.kylin.mdm.CDControl.service
$ sudo systemctl start dbus-com.kylin.mdm.CDControl.service
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.CDControl
  - 路径名称：/com/kylin/kysdk/CDControl
  - Interfaces：com.kylin.kysdk.CDControl
  - 信号：无
- **子模块信息**

设置光驱管控状态(自1.2.0.8版本启用)

子模块	光驱管控	
接口类型	DBus	
原型	SetStatus (Int32 status) ↦ (Int32 arg_1)	
描述	设置光驱管控状态	
参数	status	管控状态(0 禁用；1 只读；2 启用)
返回值	0	成功

	其他	失败
备注	无	

获取光驱管控状态(自1.2.0.8版本启用)

子模块	光驱管控	
接口类型	DBus	
原型	GetStatus () ↦ (Int32 arg_0)	
描述	获取光驱管控状态	
参数	无	无
返回值	0	禁用
	1	只读
	2	启用
	其他	失败
备注	无	

6.4.7.5 键鼠管控

封装 DBus 接口设置键鼠管控（该功能需麒麟授权，如果需要使用，请联系麒麟技术开发人员）

- **服务说明:**
  自libkysdk-desktopctrl 1.2.0.9-0k1.2版本起，取消了桌管外设管控服务自启动，若需使用相关功能请启动以下服务：

```
# 桌面管控设备插拔监控服务:
$ sudo systemctl enable kylin-mdm-monitor.service
$ sudo systemctl start kylin-mdm-monitor.service

# 键鼠管控DBus服务:
$ sudo systemctl enable dbus-com.kylin.mdm.hidcontrol.service
$ sudo systemctl start dbus-com.kylin.mdm.hidcontrol.service
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.InputControl.hid
  - 路径名称：/com/kylin/kysdk/InputControl/hid
  - Interfaces：com.kylin.kysdk.InputControl.hid
  - 信号：无
- **子模块信息**

设置键鼠设备禁用(自1.2.0.8版本启用)

子模块	键鼠管控	
接口类型	DBus	
原型	DisableHidDevices () ↦ (Int32 arg_0)	
描述	设置键鼠设备禁用	
参数	无	无
返回值	0	成功
	其他	失败
备注	无	

设置键鼠设备启用(自1.2.0.8版本启用)

子模块	键鼠管控
接口类型	DBus
原型	EnableHidDevices () ↦ (Int32 arg_0)

描述	设置键鼠设备启用	
参数	无	无
返回值	0	成功
	其他	失败
备注	无	

获取键鼠管控状态(自1.2.0.8版本启用)

子模块	键鼠管控	
接口类型	DBus	
原型	GetHidDevices () ↦ (Int32 arg_0)	
描述	获取键鼠管控状态	
参数	无	无
返回值	0	禁用
	1	启用
	其他	失败
备注	无	

6.4.7.6 手机管控

封装 DBus 接口设置手机管控（该功能需麒麟授权，如果需要使用，请联系麒麟技术开发人员）

- **服务说明：**
  自libkysdk-desktopctrl 1.2.0.9-0k1.2版本起，取消了桌管外设管控服务自启动，若需使用相关功能请启动以下服务：

```
# 手机管控DBus服务：
$ sudo systemctl enable dbus-com.kylin.mdm.phonecontrol.service
$ sudo systemctl start dbus-com.kylin.mdm.phonecontrol.service
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.PhoneControl
  - 路径名称：/com/kylin/kysdk/PhoneControl
  - Interfaces：com.kylin.kysdk.PhoneControl
  - 信号：无
- **子模块信息**

设置手机管控状态(自1.2.0.8版本启用)

子模块	手机管控	
接口类型	DBus	
原型	SetStatus (Int32 status) ↦ (Int32 arg_1)	
描述	设置手机管控状态	
参数	status	管控状态(0 禁用；1 只读；2 启用)
返回值	0	成功
	其他	失败
备注	无	

获取手机管控状态(自1.2.0.8版本启用)

子模块	手机管控
接口类型	DBus
原型	GetStatus () ↦ (Int32 arg_0)

描述	获取手机管控状态	
参数	无	无
返回值	0	禁用
	1	只读
	2	启用
	其他	失败
备注	无	

6.4.7.7 热点管控

封装 DBus 接口设置热点管控（该功能需麒麟授权，如果需要使用，请联系麒麟技术开发人员）

- **服务说明：**
  自libkysdk-desktopctrl 1.2.0.9-0k1.2版本起，取消了桌管外设管控服务自启动，若需使用相关功能请启动以下服务：

```
# 热点管控DBus服务：
$ sudo systemctl enable dbus-com-kylin-hotPointControl.service
$ sudo systemctl start dbus-com-kylin-hotPointControl.service
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.hotPointControl
  - 路径名称：/com/kylin/kysdk/hotPointControl
  - Interfaces：com.kylin.kysdk.hotPointControl
  - 信号：无
- **子模块信息**

设置热点管控状态(自1.2.0.8版本启用)

子模块	热点管控	
接口类型	DBus	
原型	SetStatus (Int32 enable) ↦ (Int32 res)	
描述	设置热点管控状态	
参数	status	管控状态(0 不管控；1 管控)
返回值	1	设置成功
	0	设置失败
备注	无	

获取热点管控状态(自1.2.0.8版本启用)

子模块	热点管控	
接口类型	DBus	
原型	GetStatus () ↦ (Int32 arg_0)	
描述	获取热点管控状态	
参数	无	无
返回值	0	不管控
	1	管控
备注	无	

6.4.7.8 内外网隔离管控

封装 DBus 接口设置内外网隔离管控（该功能需麒麟授权，如果需要使用，请联系麒麟技术开发人员）

- **服务说明：**
  自libkysdk-desktopctrl 1.2.0.9-0k1.2版本起，取消了桌管外设管控服务自启动，若需使用相关功能请启动以下服务：

```
# 内外网隔离管控DBus服务：
$ sudo systemctl enable dbus-com-kylin-netIsolationctl.service
$ sudo systemctl start dbus-com-kylin-netIsolationctl.service

# 内外网隔离管控监控服务：
$ sudo systemctl enable dbus-com-kylin-checkinoutnet.service
$ sudo systemctl start dbus-com-kylin-checkinoutnet.service
```

- **DBus信息**
  - dbus 服务名称：com.kylin.kysdk.netIsolationctl
  - 路径名称：/com/kylin/kysdk/netIsolationctl
  - Interfaces：com.kylin.kysdk.netIsolationctl
  - 信号：无
- **子模块信息**

设置内外网隔离管控状态(自1.2.0.8版本启用)

子模块	内外网隔离管控	
接口类型	DBus	
原型	SetStatus (Int32 enable, String InnerNetIp, String InnerDNS, String OuterNetIp, Int32 HowOften) ↦ (Int32 res)	
描述	设置内外网隔离管控状态	
参数	enable	管控状态 (0 不管控；1 管控)
	InnerNetIp	内网ip (若输入多个ip，用逗号分隔，如：“172.17.66.196,172.17.66.163”)
	InnerDNS	内网DNS (若输入多个DNS，用逗号分隔，如：“1.2.3.4,172.17.50.100”)
	OuterNetIp	外网的ip或网址 (若输入多个，用逗号分隔，如：“www.baidu.com,www.163.com”；若不输入，则默认为“www.baidu.com”)
	HowOften	周期检查时间间隔 (为20-6000的整数)
返回值	1	设置成功
	0	设置失败
备注	无	

获取内外网隔离管控状态(自1.2.0.8版本启用)

子模块	内外网隔离管控	
接口类型	DBus	
原型	GetStatus () ↦ (Int32 res)	
描述	获取内外网隔离管控状态	
参数	无	无
返回值	0	不管控
	1	管控
备注	无	

6.5 数据安全

- **头文件路径：**

```
#include "/usr/include/kysdk/kysdk-security/libkydatasec.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkydatasec.so
```

- **子模块信息：**

对称加密，使用AES算法（128位）ECB模式进行对称加密(自1.2.0版本启用)

子模块	使用AES128算法对称加密	
接口类型	C	
原型	int kdk_cipher_encrypt_aes128_ecb(const unsigned char *key, unsigned int keylen,const unsigned char *iv, unsigned int ivlen,const unsigned char *plaindata, unsigned int plaindatalen，unsigned char *cipherdata, unsigned int *cipherdatalen);	
描述	对称加密，使用AES算法（128位）ECB模式进行对称加密	
参数	key	密钥数据
	keylen	秘钥数据长度
	iv	向量
	ivlen	向量长度
	plaindata	明文数据
	plaindatalen	明文数据长度
	cipherdata	密文数据
	cipherdatalen	密文数据长度
返回值	int	0-成功，非0-失败
备注	无	

对称解密，使用AES算法（128位）ECB模式进行对称解密(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_cipher_decrypt_aes128_ecb(const unsigned char *key, unsiged int keylen,const cunsigned char *iv,unsigned int ivlen, const unsigned char *cipherdata,unsigned int cipherdatalen,unsigned char *plaindata,unsigned int *plaindatalen);	
描述	对称解密，使用AES算法（128位）ECB模式进行对称解密	
参数	key	密钥数据
	keylen	秘钥数据长度
	iv	向量
	ivlen	向量长度
	cipherdata	密文数据
	cipherdatalen	密文数据长度
	plaindata	明文数据
	plaindatalen	明文数据长度

返回值	int	0-成功，非0-失败
备注	无	

对称加密，使用AES算法（256位）CBC模式进行对称加密(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_cipher_encrypt_aes256_cbc(const unsigned char *key, unsiged int keylen,const cunsigned char *iv,unsigned int ivlen, const unsigned char *plaindata,unsigned int plaindatalen,unsigned char *cipherdata,unsigned int *cipherdatalen);	
描述	对称加密，使用AES算法（256位）CBC模式进行对称加密	
参数	key	密钥数据
	keylen	秘钥数据长度
	iv	向量
	ivlen	向量长度
	plaindata	明文数据
	plaindatalen	明文数据长度
	cipherdata	密文数据
	cipherdatalen	密文数据长度
返回值	int	0-成功，非0-失败
备注	无	

对称加密，使用AES算法（256位）CBC模式进行对称解密(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_cipher_decrypt_aes256_cbc(const unsigned char *key, unsiged int keylen,const cunsigned char *iv,unsigned int ivlen, const unsigned char *cipherdata,unsigned int cipherdatalen,unsigned char *plaindata,unsigned int *plaindatalen);	
描述	对称加密，使用AES算法（256位）CBC模式进行对称解密	
参数	key	密钥数据
	keylen	秘钥数据长度
	iv	向量
	ivlen	向量长度
	cipherdata	密文数据
	cipherdatalen	密文数据长度
	plaindata	明文数据
	plaindatalen	明文数据长度
返回值	int	0-成功，非0-失败
备注	无	

非对称加密，生成RSA算法的公私钥对(自1.2.0版本启用)

子模块	数据安全

接口类型	C	
原型	int kdk_rsa_generatekeys(unsigned int bit,unsigned char *pubkey, unsigned int *pubkeylen,unsigned char *privkey,unsigned int *privkeylen,unsigned char *password,unsigned int passwordlen)	
描述	非对称加密，生成RSA算法的公私钥对	
参数	bit	位数（1024/2048/4096等等）
	pubkey	公钥数据
	pubkeylen	公钥数据长度
	privkey	私钥数据
	privkeylen	私钥数据长度
	password	加密私钥数据的对称算法密钥， 如果此实参为空， 则私钥不进行加密； 如果此参数不为空， 将使用数据对私钥进行aes256-cbc加密
	passwordlen	加密私钥数据的对称算法密钥长度
返回值	int	0-成功，非0-失败
备注	无	

非对称加密，使用RSA算法进行非对称加密(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_rsa_encrypt(const unsigned char *pubkey, unsigned int pubkeylen,const unsigned char *plaindata, unsigned int plaindatalen,unsigned char *cipherdata, unsigned int *cipherdatalen);	
描述	非对称加密，使用RSA算法进行非对称加密	
参数	pubkey	公钥数据
	pubkeylen	公钥数据长度
	plaindata	明文数据
	plaindatalen	明文数据长度
	cipherdata	密文数据
	cipherdatalen	密文数据长度
返回值	int	0-成功，非0-失败
备注	无	

非对称加密，使用RSA算法进行非对称解密(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_rsa_decrypt(const unsigned char *privkey, unsigned int privkeylen,unsigned char *password, unsigned int passwordlen,const unsigned char *cipherdata, unsigned int cipherdatalen,unsigned char *plaindata, unsigned int *plaindatalen);	
描述	非对称加密，使用RSA算法进行非对称解密	
参数	privkey	私钥数据

	privkeylen	私钥数据长度
	password	加密私钥的密钥数据， 如果此实参为空， 则不对私钥进行解密操
	passwordlen	加密私钥的密钥数据长度， 如果此实参为0， 与password为空相同处理
	cipherdata	密文数据
	cipherdatalen	密文数据长度
	plaindata	明文数据
	plaindatalen	明文数据长度
返回值	int	0-成功，非0-失败
备注	无	

非对称加密，使用RSA算法进行非对称签名(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_rsa_sign(const unsigned char *data, unsigned int datalen,const unsigned char *privkey, unsigned int privkeylen,const unsigned char *password, unsigned int passwordlen,unsigned char *signdata, unsigned int *signdatalen);	
描述	非对称加密，使用RSA算法进行非对称签名	
参数	data	原始数据
	datalen	原始数据长度
	privkey	私钥数据
	privkeylen	私钥数据长度
	password	加密私钥的密钥数据， 如果此实参为空， 则不对私钥进行解密操作
	passwordlen	加密私钥的密钥数据长度， 如果此实参为0， 与password为空相同处理
	signdata	签名数据
	signdatalen	签名数据长度
返回值	int	0-成功，非0-失败
备注	无	

非对称加密，使用RSA算法进行非对称验签(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_rsa_verify(const unsigned char *signdata, unsigned int signdatalen,const unsigned char *pubkey, unsigned int pubkeylen,const unsigned char *data, unsigned int datalen);	
描述	非对称加密，使用RSA算法进行非对称签名	
参数	signdata	签名数据
	signdatalen	签名数据长度

	pubkey	公钥数据
	pubkeylen	公钥数据长度
	data	原数据
	datalen	原数据长度
返回值	int	0-成功，非0-失败
备注	无	

杂凑算法，MD5算法计算(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_md5(const unsigned char *data, unsigned int datalen, unsigned char *hash, unsigned int *hashlen);	
描述	杂凑算法，MD5算法计算	
参数	data	原数据
	datalen	原数据长度
	hash	HASH值
	hashlen	HASH值长度
返回值	int	0-成功，非0-失败
备注	无	

杂凑算法，SHA1算法计算(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_sha1(const unsigned char *data, unsigned int datalen, unsigned char *hash, unsigned int *hashlen);	
描述	杂凑算法，SHA1算法计算	
参数	data	原数据
	datalen	原数据长度
	hash	HASH值
	hashlen	HASH值长度
返回值	int	0-成功，非0-失败
备注	无	

杂凑算法，sha256算法计算(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_sha256(const unsigned char *data, unsigned int datalen, unsigned char *hash, unsigned int *hashlen);	
描述	杂凑算法，sha256算法计算	
参数	data	原数据
	datalen	原数据长度
	hash	HASH值
	hashlen	HASH值长度
返回值	int	0-成功，非0-失败

备注	无

杂凑算法，sha384算法计算(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_sha384(const unsigned char *data, unsigned int datalen, unsigned char *hash, unsigned int *hashlen);	
描述	杂凑算法，sha384算法计算	
参数	data	原数据
	datalen	原数据长度
	hash	HASH值
	hashlen	HASH值长度
返回值	int	0-成功，非0-失败
备注	无	

杂凑算法，sha512算法计算(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_sha512(const unsigned char *data, unsigned int datalen, unsigned char *hash, unsigned int *hashlen);	
描述	杂凑算法，sha512算法计算	
参数	data	原数据
	datalen	原数据长度
	hash	HASH值
	hashlen	HASH值长度
返回值	int	0-成功，非0-失败
备注	无	

BASE64编码(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_base64_encode(const unsigned char *plaindata, unsigned int plaindatalen, unsigned char *base64, unsigned int *base64len);	
描述	BASE64编码	
参数	plaindata	明文数据
	plaindatalen	明文数据长度
	base64	编码后数据
	base64len	编码后数据长度
返回值	int	0-成功，非0-失败
备注	无	

BASE64解码(自1.2.0版本启用)

子模块	数据安全

接口类型	C	
原型	int kdk_base64_decode(const unsigned char *base64, unsigned int base64len, unsigned char *plaindata, unsigned int *plaindatalen);	
描述	BASE64解码	
参数	base64	编码数据
	base64len	编码数据长度
	plaindata	明文数据
	plaindatalen	明文数据长度
返回值	int	0-成功，非0-失败
备注	无	

随机数发生器(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_rand(unsigned char *rnd, unsigned int rndlen);	
描述	随机数发生器	
参数	rnd	随机数内存
	rndlen	生成随机数的长度
返回值	int	0-成功，非0-失败
备注	无	

对称加密，使用SM4算法CBC模式进行对称加密(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_cipher_encrypt_sm4_cbc(const unsigned char *key, unsigned int keylen,const unsigned char *iv, unsigned int ivlen,const unsigned char *plaindata, unsigned int plaindatalen,unsigned char *cipherdata, unsigned int *cipherdatalen); | 对称加密， 使用SM4算法CBC模式进行对称加密	
描述	对称加密，使用SM4算法CBC模式进行对称加密	
参数	key	密钥数据
	keylen	秘钥数据长度
	iv	向量
	ivlen	向量长度
	plaindata	明文数据
	plaindatalen	明文数据长度
	cipherdata	密文数据
	cipherdatalen	密文数据长度
返回值	int	0-成功，非0-失败
备注	无	

对称加密，使用SM4算法CBC模式进行对称解密(自1.2.0版本启用)

子模块	数据安全

接口类型	C	
原型	int kdk_cipher_decrypt_sm4_cbc(const unsigned char *key, unsigned int keylen,const unsigned char *iv, unsigned int ivlen, const unsigned char *cipherdata, unsigned int cipherdatalen, unsigned char *plaindata, unsigned int *plaindatalen);	
描述	对称加密，使用SM4算法CBC模式进行对称解密	
参数	key	密钥数据
	keylen	秘钥数据长度
	iv	向量
	ivlen	向量长度
	cipherdata	密文数据
	cipherdatalen	密文数据长度
	plaindata	明文数据
	plaindatalen	明文数据长度
返回值	int	0-成功，非0-失败
备注	无	

非对称加密，生成SM2算法的公私钥对(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_sm2_generatekeys(unsigned int bit,unsigned char *pubkey, unsigned int *pubkeylen,unsigned char *privkey,unsigned int *privkeylen,unsigned char *password,unsigned int passwordlen)	
描述	非对称加密，生成SM2算法的公私钥对	
参数	bit	位数（256/512等等
	pubkey	公钥数据
	pubkeylen	公钥数据长度
	privkey	私钥数据
	privkeylen	私钥数据长度
	password	加密私钥数据的对称算法密钥， 如果此实参为空， 则私钥不进行加密； 如果此参数不为空， 将使用数据对私钥进行aes256-cbc加密
	passwordlen	加密私钥数据的对称算法密钥长度
返回值	int	0-成功，非0-失败
备注	无	

非对称加密，使用SM2算法进行非对称加密(自1.2.0版本启用)

子模块	数据安全
接口类型	C
原型	int kdk_sm2_encrypt(const unsigned char *pubkey, unsigned int pubkeylen,const unsigned char *plaindata,

	unsigned int plaindatalen,unsigned char *cipherdata, unsigned int *cipherdatalen);	
描述	非对称加密，使用SM2算法进行非对称加密	
参数	pubkey	公钥数据
	pubkeylen	公钥数据长度
	plaindata	明文数据
	plaindatalen	明文数据长度
	cipherdata	密文数据
	cipherdatalen	密文数据长度
返回值	int	0-成功，非0-失败
备注	无	

非对称加密，使用SM2算法进行解密(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_sm2_decrypt(const unsigned char *privkey, unsigned int privkeylen,unsigned char *password, unsigned int passwordlen,const unsigned char *cipherdata, unsigned int cipherdatalen,unsigned char *plaindata, unsigned int *plaindatalen);	
描述	非对称加密，使用SM2算法进行解密	
参数	privkey	私钥数据
	privkeylen	私钥数据长度
	password	加密私钥的密钥数据， 如果此实参为空， 则不对私钥进行解密操
	passwordlen	加密私钥的密钥数据长度， 如果此实参为0， 与password为空相同处理
	cipherdata	密文数据
	cipherdatalen	密文数据长度
	plaindata	明文数据
	plaindatalen	明文数据长度
返回值	int	0-成功，非0-失败
备注	无	

数据签名，使用SM2算法进行数据签名(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_sm2_sign(const unsigned char *data, unsigned int datalen,const unsigned char *privkey, unsigned int privkeylen,const unsigned char *password, unsigned int passwordlen,unsigned char *signdata, unsigned int *signdatalen);	
描述	数据签名，使用SM2算法进行数据签名	
参数	data	原始数据
	datalen	原始数据长度

	privkey	私钥数据
	privkeylen	私钥数据长度
	password	加密私钥的密钥数据， 如果此实参为空， 则不对私钥进行解密操作
	passwordlen	加密私钥的密钥数据长度， 如果此实参为0， 与password为空相同处理
	signdata	签名数据
	signdatalen	签名数据长度
返回值	int	0-成功，非0-失败
备注	无	

数据验签，使用SM2算法进行数据验签(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_sm2_verify(const unsigned char *signdata, unsigned int signdatalen,const unsigned char *pubkey, unsigned int pubkeylen,const unsigned char *data, unsigned int datalen);	
描述	数据验签，使用SM2算法进行数据验签	
参数	signdata	签名数据
	signdatalen	签名数据长度
	pubkey	公钥数据
	pubkeylen	公钥数据长度
	data	原数据
	datalen	原数据长度
返回值	int	0-成功，非0-失败
备注	无	

杂凑算法，sm3算法计算(自1.2.0版本启用)

子模块	数据安全	
接口类型	C	
原型	int kdk_sm3(const unsigned char *data, unsigned int datalen, unsigned char *hash, unsigned int *hashlen);	
描述	杂凑算法，sm3算法计算	
参数	data	原数据
	datalen	原数据长度
	hash	HASH值
	hashlen	HASH值长度
返回值	int	0-成功，非0-失败
备注	无	

6.6 文件安全

文件安全提供文件保护

- **安装命令：**

```
$ sudo apt-get install libkysdk-filesec-dev libkysdk-filesec
```

- **构建示例：**

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-filesec
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKFILE kysdk-filesec)
target_include_directories(demo PRIVATE ${KYSDKFILE_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKFILE_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKFILE_LIBRARIES})
```

6.6.1 文件保护

- **头文件路径：**

```
#include "/usr/include/kysdk/kysdk-security/libkyfilesec.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkyfilesec.so
```

- **dbus信息**
  - System bus 接口
  - dbus 服务名称：com.kylin.kysdk.filesec
  - 路径名称：/com/kylin/kysdk/filesec/protected
  - Interfaces：com.kylin.kysdk.filesec.protected
- **子模块信息：**

> 设置文件只读属性(自1.2.0版本启用)

子模块	文件安全	
接口类型	C	
原型	int kdk_set_file_readonly(const char* filepath)	
描述	设置文件只读属性	
参数	filepath	文件路径
返回值	0	成功
	-5000	门禁管控禁止访问
	非零	失败
备注	无	

> 取消文件只读属性(自1.2.0版本启用)

子模块	文件安全	
接口类型	C	
原型	int kdk_cancel_file_readonly(const char* filepath)	
描述	取消文件只读属性	

参数	filepath	文件路径
返回值	0	成功
	-5000	门禁管控禁止访问
	非零	失败
备注	无	

添加文件保护(自1.2.0.9-0k3.0版本启用)

子模块	文件安全	
接口类型	C	
原型	int kdk_set_file_protected(const char *path)	
描述	添加文件保护	
参数	path	文件路径
返回值	0	成功
	-1	程序未授权
	-2	参数传入异常
	-3	参数路径不存在
	-4	添加程序至文件保护失败
	-5000	门禁管控禁止访问
备注	无	

解除文件保护(自1.2.0.9-0k3.0版本启用)

子模块	文件安全	
接口类型	C	
原型	int kdk_clear_file_protected(const char *path)	
描述	解除文件保护	
参数	path	文件路径
返回值	0	成功
	-1	程序未授权
	-2	参数传入异常
	-3	参数路径不存在
	-4	添加程序至文件保护失败
	-5000	门禁管控禁止访问
备注	无	

将目录内所有文件添加至文件保护(自1.2.0.9-0k3.0版本启用)

子模块	文件安全	
接口类型	C	
原型	int kdk_set_dir_protected(const char *dpath)	
描述	将目录内所有文件添加至文件保护	
参数	dpath	目录路径
返回值	>=0	成功添加文件保护数量
	-1	程序未授权
	-2	参数传入异常

	-3	参数路径不存在
	-4	添加目录内所有程序至文件保护失败
	-5000	门禁管控禁止访问
备注	无	

将目录内所有文件解除文件保护(自1.2.0.9-0k3.0版本启用)

子模块	文件安全	
接口类型	C	
原型	int kdk_clear_dir_protected(const char *dpath)	
描述	将目录内所有文件解除文件保护	
参数	dpath	目录路径
返回值	>=0	成功解除文件保护数量
	-1	程序未授权
	-2	参数传入异常
	-3	参数路径不存在
	-4	移除目录内所有程序至文件保护失败
	-5000	门禁管控禁止访问
备注	无	

获取所有文件保护节点(自1.2.0.9-0k3.0版本启用)

子模块	文件安全	
接口类型	C	
原型	char **kdk_get_file_protected(int *count)	
描述	获取所有文件保护节点	
参数	count	节点数量
返回值	char**	节点数组
	NULL	失败或记录不存在
备注	无	

- **其他接口类型接口：**

接口类型	接口	入参	返回值
dbus	int SetFileProtected(String filepath)	path:文件路径	int 0: 成功 -1：程序未授权 -2：参数传入异常 -3：参数路径不存在 -4：添加程序至文件保护失败 -5000：门禁管控禁止访问

接口类型	接口	入参	返回值
dbus	Array of [String] GetFileProtected()	无	char** 节点数组； NULL：失败或记录不存在

接口类型	接口	入参	返回值
dbus	int ClearFileProtected(String filepath)	filepath:文件路径	int 0：成功 -1：程序未授权 -2：参数传入异常 -3：参数路径不存在

			-4：解除程序文件保护失败 -5000：门禁管控禁止访问

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <string.h>

#include <libkyfilesec.h>
#include "assert.h"

int main()
{
    int iret = 0;
    char* file_path = "/tmp/kdk_file_test";
    char* dir_path = "/tmp/kdk_dir_test";
    int len;
    char **p;

    iret = kdk_set_file_readonly(file_path);
    printf("set file readonly file:%s result:%d\n", file_path, iret);

    iret = kdk_cancel_file_readonly(file_path);
    printf("cancel file readonly file:%s result:%d\n", file_path, iret);

    iret = kdk_set_file_protected(file_path);
    printf("set file protected file:%s iret = %d\n",file_path, iret);

    iret = kdk_clear_file_protected(file_path);
    printf("clear file protected file:%s iret = %d\n",file_path, iret);

    iret = kdk_set_dir_protected(dir_path);
    printf("set dir protected dir_path:%s iret = %d\n",dir_path, iret);

    iret = kdk_clear_dir_protected(dir_path);
    printf("clear dir protected dir_path:%s iret = %d\n",dir_path, iret);

    p = kdk_get_file_protected(&len);
    printf("get_file_protected, len:%d\n",len);
    for (int i = 0; i < len; i++) {
        printf("%s\n", p[i]);
    }

    return 0;
}
```

6.7 网络安全

基于Iptables/Netfilter制定网络安全策略接口集，通过对网络防火墙管控提升系统网络安全防护能力。实现新增规则、删除规则、清空规则、下发防火墙策略等接口。并可以灵活配置防火墙源地址、目的地址、源端口、目的端口，实现自定义链操作，以及麒麟防火墙的相关规则配置。

- **安装命令：**

```
$ sudo apt-get install libkysdk-networksec libkysdk-networksec-dev
```

- **构建示例：**

（1）qt .pro 构建项目

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-networksec
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKNETWORKSEC kysdk-networksec)
target_include_directories(demo PRIVATE ${KYSDKNETWORKSEC_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKNETWORKSEC_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKNETWORKSEC_LIBRARIES})
```

6.7.1 通用防火墙配置

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkynetworksec.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkynetworksec.so
```

- **子模块信息：**

申请一个防火墙配置上下文(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	kdk_firewall_ctx *kdk_firewall_ctx_new()	
描述	申请一个防火墙配置上下文	
参数	无	
返回值	kdk_firewall_ctx *	上下文指针
备注	无	

释放防火墙操作上下文内存(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	void kdk_firewall_ctx_free(kdk_firewall_ctx *ctx)	
描述	释放防火墙操作上下文内存	
参数	ctx	上下文指针
返回值	无	
备注	无	

设置防火墙设置表(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_table(kdk_firewall_ctx *ctx, kdk_firewall_table table)	
描述	设置防火墙设置表，包括：raw、mangle、nat、filter	
参数	ctx	上下文指针
	table,参见kdk_firewall_table	操作表
返回值	0	成功
	非0	失败
备注	typedef enum _kdk_firewall_table { FW_TABLE_RAW = 0, FW_TABLE_MANGLE = 1,	

	FW_TABLE_NAT = 2, FW_TABLE_FILTER = 3 } kdk_firewall_table;

设置防火墙命令项(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_command(kdk_firewall_ctx *ctx, kdk_firewall_command command)	
描述	设置防火墙命令项，例如：新增规则、删除规则、 清空规则等	
参数	ctx	上下文指针
	command	命令, 参见kdk_firewall_command
返回值	0	成功
	非0	失败

备注	typedef enum _kdk_firewall_command {
	FW_COMMAND_ADD = 0, //添加规则
	FW_COMMAND_DEL = 1, //删除规则
	FW_COMMAND_INSERT = 2, //插入规则
	FW_COMMAND_NEWCHAIN = 3, //创建新链
	FW_COMMAND_DELETECHAIN = 4, //删除链
	FW_COMMAND_RENAMECHAIN = 5, //重命名链
	FW_COMMAND_FLUSH = 6 //删除所有规则
	} kdk_firewall_command;	

设置防火墙链(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_chain(kdk_firewall_ctx *ctx, kdk_firewall_chain chain)	
描述	设置防火墙链，例如： PREROUTING/INPUT/NAT/OUTPUT/POSTROUTING	
参数	ctx	上下文指针
	chain	链 参见kdk_firewall_chain
返回值	0	成功
	非0	失败
备注	typedef enum _kdk_firewall_chain { FW_CHAIN_INPUT = 0, FW_CHAIN_FORWARD = 1, FW_CHAIN_OUTPUT = 2, FW_CHAIN_PREROUTING = 3, FW_CHAIN_POSTROUTING = 4 } kdk_firewall_chain;	

设置防火墙自定义链(自1.2.0版本启用)

子模块	防火墙配置
接口类型	C
原型	int kdk_firewall_set_chain_ex(kdk_firewall_ctx *ctx, const char *chain)

描述	设置防火墙自定义链	
参数	ctx	上下文指针
	chain	链， 可以是自定义链也可以是原有链
返回值	0	成功
	非0	失败
备注	无	

设置防火墙新旧链名称(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_old_and_new_chain(kdk_firewall_ctx *ctx, const char *oldchain, const char *newchain)	
描述	设置防火墙新旧链名称， 此函数主要使用方是更新链名称使用	
参数	ctx	上下文指针
	oldchain	旧链名称
	newchain	新链名称
返回值	0	成功
	非0	失败
备注	无	

设置防火墙协议(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_protocol(kdk_firewall_ctx *ctx, kdk_firewall_protocol protocol)	
描述	设置防火墙协议	
参数	ctx	上下文指针
	protocol	协议， 参见kdk_firewall_protocol
返回值	0	成功
	非0	失败
备注	typedef enum _kdk_firewall_protocol { FW_PROTOCOL_TCP = 0, FW_PROTOCOL_UDP = 1, FW_PROTOCOL_ICMP = 2, } kdk_firewall_protocol;	

设置防火墙协议,字符串格式(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_protocol_ex(kdk_firewall_ctx *ctx, const char *protocol)	
描述	设置防火墙协议，字符串格式，参见/etc/protocols内容	

参数	ctx	上下文指针
	protocol	协议，参见/etc/protocols
返回值	0	成功
	非0	失败
备注	无	

设置防火墙源地址(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_source_address(kdk_firewall_ctx *ctx, const char *saddr)	
描述	设置防火墙源地址	
参数	ctx	上下文指针
	saddr	源地址
返回值	0	成功
	非0	失败
备注	无	

设置防火墙目的地址(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_destnation_address(kdk_firewall_ctx *ctx, const char *daddr)	
描述	设置防火墙规则的目的地址	
参数	ctx	上下文指针
	daddr	目的地址
返回值	0	成功
	非0	失败
备注	无	

设置防火墙规则的输入接口(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_input_interface(kdk_firewall_ctx *ctx, const char *iface)	
描述	设置防火墙规则的输入接口	
参数	ctx	上下文指针
	iface	网络接口，例如：eth0、eth+
返回值	0	成功
	非0	失败
备注	无	

设置防火墙规则的输出接口(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_output_interface(kdk_firewall_ctx *ctx, const char *iface)	
描述	设置防火墙规则的输出接口	
参数	ctx	上下文指针
	iface	网络接口，例如：eth0、eth+
返回值	0	成功
	非0	失败
备注	无	

设置防火墙规则的源端口(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_source_port(kdk_firewall_ctx *ctx, unsigned short port)	
描述	设置防火墙规则的源端口	
参数	ctx	上下文指针
	port	端口号
返回值	0	成功
	非0	失败
备注	无	

设置防火墙规则的源端口范围(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_source_port_range(kdk_firewall_ctx *ctx, unsigned short from, unsigned short to)	
描述	设置防火墙规则的源端口范围	
参数	ctx	上下文指针
	from	端口最小值
	to	端口最大值
返回值	0	成功
	非0	失败
备注	无	

使用服务名称设置防火墙规则的源端口(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int	

	kdk_firewall_set_source_port_with_servicename(kdk_firewall_ctx *ctx, const char *servicename)	
描述	使用服务名称设置防火墙规则的源端口	
参数	ctx	上下文指针
	servicename	服务名称，例如：ssh、ftp等
返回值	0	成功
	非0	失败
备注	无	

设置防火墙规则的目的端口(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_destnation_port(kdk_firewall_ctx *ctx, unsigned short port)	
描述	设置防火墙规则的目的端口	
参数	ctx	上下文指针
	port	端口号
返回值	0	成功
	非0	失败
备注	无	

设置防火墙规则的目的端口范围(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_destnation_port_range(kdk_firewall_ctx *ctx, unsigned short from, unsigned short to)	
描述	设置防火墙规则的目的端口范围	
参数	ctx	上下文指针
	from	端口最小值
	to	端口最大值
返回值	0	成功
	非0	失败
备注	无	

使用服务名称设置防火墙规则的目的端口(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_destnation_port_with_servicename(kdk_firewall_ctx *ctx, const char *servicename)	
描述	使用服务名称设置防火墙规则的目的端口	
参数	ctx	上下文指针

	servicename	服务名称，例如：ssh、ftp等
返回值	0	成功
	非0	失败
备注	无	

设置防火墙规则的目的地(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_target(kdk_firewall_ctx *ctx, kdk_firewall_target target)	
描述	设置防火墙规则的目的地	
参数	ctx	上下文指针
	target	目的地，参见kdk_firewall_target
返回值	0	成功
	非0	失败
备注	typedef enum _kdk_firewall_target { FW_TARGET_ACCEPT = 0, FW_TARGET_DROP = 1, FW_TARGET_REJECT = 2, } kdk_firewall_target;	

设置防火墙规则的目的地自定义(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_set_target_ex(kdk_firewall_ctx *ctx, const char *target)	
描述	设置防火墙规则的目的地自定义	
参数	ctx	上下文指针
	target	目的地，可以是原有目的地REJECT/ACCEPT/DROP，也可以是自定义链
返回值	0	成功
	非0	失败
备注	无	

下发防火墙策略(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_issue_policy(kdk_firewall_ctx *ctx)	
描述	下发防火墙策略	
参数	ctx	上下文指针

返回值	0	成功
	非0	失败
备注	无	

创建一个自定义链(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_create_custom_chain(const char *chain)	
描述	创建一个自定义链	
参数	chain	自定义链
返回值	0	成功
	非0	失败
备注	无	

删除一个自定义链(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_delete_custom_chain(const char *chain)	
描述	删除一个自定义链	
参数	chain	自定义链
返回值	0	成功
	非0	失败
备注	无	

重命名一个自定义链(自1.2.0版本启用)

子模块	防火墙配置	
接口类型	C	
原型	int kdk_firewall_rename_custom_chain(const char *oldchain, const char *newchain)	
描述	重命名一个自定义链	
参数	oldchain	旧链
	newchain	新链
返回值	0	成功
	非0	失败
备注	无	

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#include <kysdk/kysdk-security/libkynetworksec.h>

//iptables -I INPUT -p TCP --dport 9999 -j DROP
static int auto_test_drop_tcp(int type)
{
    int rc = 0;
    kdk_firewall_ctx *ctx = NULL;

    ctx = kdk_firewall_ctx_new();
    if (!ctx) { rc = -1; goto end; }

    rc = kdk_firewall_set_chain(ctx, FW_CHAIN_INPUT);
    if (rc != 0) { goto end; }

    rc = kdk_firewall_set_command(ctx, type);
    if (rc != 0) { goto end; }

    rc = kdk_firewall_set_protocol(ctx, FW_PROTOCOL_TCP);
    if (rc != 0) { goto end; }

    rc = kdk_firewall_set_destnation_port(ctx, 9999);
    if (rc != 0) { goto end; }

    rc = kdk_firewall_set_target(ctx, FW_TARGET_DROP);
    if (rc != 0) {
        goto end;
    }

    rc = kdk_firewall_issue_policy(ctx);
end:

    if (ctx) kdk_firewall_ctx_free(ctx);
    printf("rc=%d\n", rc);
    return 0;
}

int main()
{
    return auto_test_drop_tcp();
}
```

6.7.2 麒麟防火墙

- **头文件路径：**

```
#include "kysdk/kysdk-security/libkyfirewall.h"
```

- **so库路径：**

```
/usr/lib/kysdk/kysdk-security/libkynetworksec.so
```

- **子模块信息：**

> 设置KSC防火墙网络模式的出、入站管控策略(自2.3.0版本启用)

子模块	麒麟防火墙
接口类型	C
原型	int kdk_firewall_set_policy(int mode, int in_policy, int out_policy)

描述	设置KSC防火墙网络模式的出、入站管控策略	
参数	mode	网络模式(公共网络、专用网络)
	in_policy	入站策略
	out_policy	出站策略
返回值	0	成功
	-1	失败

备注	enum kdk_firewall_mode {
	FW_ALL_MODE, // 同时启用公用与专用网络
	FW_PUBLIC, // 公用网络
	FW_PRIVATE, // 专用网络
	FW_OFF, // 关闭防火墙
	FW_RESERVE // 系统保留
	};
	
	enum kdk_firewall_policy {
	INBOUND_DENY_ALL, // 入站阻止所有连接
	INBOUND_DENY, // 入站阻止（默认）
	INBOUND_ALLOW, // 入站允许
	OUTBOUND_DENY, // 出站阻止
	OUTBOUND_ALLOW // 出站允许（默认）
	};	

防火墙模式启用(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_set_policy(int mode, int in_policy, int out_policy)	
描述	设置KSC防火墙网络模式的出、入站管控策略	
参数	mode	网络模式(公共网络、专用网络)
	in_policy	入站策略
	out_policy	出站策略
返回值	0	成功
	-1	失败

备注	enum kdk_firewall_mode {
	FW_ALL_MODE, // 同时启用公用与专用网络
	FW_PUBLIC, // 公用网络
	FW_PRIVATE, // 专用网络
	FW_OFF, // 关闭防火墙
	FW_RESERVE // 系统保留
	};
	
	enum kdk_firewall_policy {
	INBOUND_DENY_ALL, // 入站阻止所有连接
	INBOUND_DENY, // 入站阻止（默认）
	INBOUND_ALLOW, // 入站允许
	OUTBOUND_DENY, // 出站阻止
	OUTBOUND_ALLOW // 出站允许（默认）
	};	

防火墙网络模式启用(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_enable_mode(int mode)	
描述	防火墙网络模式启用	
参数	mode	网络模式(公共网络、 专用网络)
返回值	0	成功
	-1	失败

备注	enum kdk_firewall_mode {
	FW_ALL_MODE, // 同时启用公用与专用网络
	FW_PUBLIC, // 公用网络
	FW_PRIVATE, // 专用网络
	FW_OFF, // 关闭防火墙
	FW_RESERVE // 系统保留
	};	

防火墙网络模式禁用(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_disable_mode(int mode)	
描述	防火墙网络模式禁用	
参数	mode	网络模式(公共网络、 专用网络)
返回值	0	成功
	-1	失败

备注	enum kdk_firewall_mode {
	FW_ALL_MODE, // 同时启用公用与专用网络
	FW_PUBLIC, // 公用网络
	FW_PRIVATE, // 专用网络
	FW_OFF, // 关闭防火墙
	FW_RESERVE // 系统保留
	};	

获取当前KSC防火墙网络模式(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_get_mode()	
描述	获取当前KSC防火墙网络模式	
参数	mode	网络模式(公共网络、 专用网络)
返回值	kdk_firewall_mode	成功
	-1	失败

备注	enum kdk_firewall_mode {
	FW_ALL_MODE, // 同时启用公用与专用网络
	FW_PUBLIC, // 公用网络
	FW_PRIVATE, // 专用网络
	FW_OFF, // 关闭防火墙
	FW_RESERVE // 系统保留
	};	

初始化麒麟防火墙规则结构体(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	KDK_FW_RULE *kdk_firewall_init_rule(void)	
描述	初始化麒麟防火墙规则结构体	
参数	无	
返回值	KDK_FW_RULE *	成功
	NULL	失败
备注	默认规则:名称为default，应用为所有，方向为所有， 网络为所有，协议为所有，本地、远程IP为任意IP，本地、 远程端口为所有端口，操作为阻止。 该接口自行分配内存， 调用后需使用kdk_firewall_destory_rule释放内存	

释放麒麟防火墙规则结构体内存(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	void kdk_firewall_destory_rule(KDK_FW_RULE *rule)	
描述	释放麒麟防火墙规则结构体内存	
参数	rule	麒麟防火墙规则结构体指针
返回值	无	
备注	无	

防火墙规则配置项赋值(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_set_rule_item(KDK_FW_RULE *rule, int item, void *value)	
描述	防火墙规则配置项赋值	
参数	rule	麒麟防火墙规则结构体指针
	item	待配置的字符串类型配置项, 详见kdk_firewall_rule_item枚举
	value	配置值， 注意item与配置值类型对应
返回值	0	成功
	-1	失败
备注	enum kdk_firewall_rule_item { FW_NAME, // 规则名称 FW_COMM, // 应用（进程名称） FW_DIRECTION, // 方向 FW_MODE, // 网络 FW_PROTOCOL, // 协议 FW_LOCAL_IP, // 本地IP FW_LOCAL_PORTS, // 本地端口 FW_REMOTE_IP, // 远程IP FW_REMOTE_PORTS, // 远程端口 FW_ACTION, // 操作	

	FW_STATUS // 状态 };

新建KSC防火墙规则(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_create_rule(KDK_FW_RULE *rule)	
描述	新建KSC防火墙规则	
参数	rule	麒麟防火墙规则结构体指针
返回值	0	成功
	kdk_firewall_invalid_rule	失败

备注	enum kdk_firewall_invalid_rule {
	RULE_EXIST_ERROR = -13, // 规则已存在
	NULL_RULE_ERROR, // 规则为空
	RULE_NAME_ERROR, // 规则名为空
	DIRECTION_ERROR, // 方向错误
	NET_MODE_ERROR, // 网络模式错误
	ACTION_ERROR, // 操作错误
	STATUS_ERROR, // 状态错误
	PROTOCOL_ERROR, // 协议错误
	LOCAL_IP_ERROR, // 本地IP错误
	REMOTE_IP_ERROR, // 远程IP错误
	LOCAL_PORT_ERROR, // 本地端口错误
	REMOTE_PORT_ERROR, // 远程端口错误
	NORMAL_ERROR // 其他错误
	};	

启用KSC防火墙规则(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_enable_rule(char *rule_name)	
描述	启用KSC防火墙规则	
参数	rule_name	防火墙规则名
返回值	0	成功
	-1	失败
备注	无	

禁用KSC防火墙规则(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_disable_rule(char *rule_name)	
描述	禁用KSC防火墙规则	
参数	rule_name	防火墙规则名
返回值	0	成功
	-1	失败
备注	无	

删除KSC防火墙规则(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_delete_rule(char *rule_name)	
描述	删除KSC防火墙规则	
参数	rule_name	防火墙规则名
返回值	0	成功
	-1	失败
备注	无	

更新KSC防火墙规则(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_update_rule(KDK_FW_RULE *rule)	
描述	更新KSC防火墙规则	
参数	rule	麒麟防火墙规则结构体指针
返回值	0	成功
	kdk_firewall_invalid_rule	失败

备注	enum kdk_firewall_invalid_rule {
	RULE_EXIST_ERROR = -13, // 规则已存在
	NULL_RULE_ERROR, // 规则为空
	RULE_NAME_ERROR, // 规则名为空
	DIRECTION_ERROR, // 方向错误
	NET_MODE_ERROR, // 网络模式错误
	ACTION_ERROR, // 操作错误
	STATUS_ERROR, // 状态错误
	PROTOCOL_ERROR, // 协议错误
	LOCAL_IP_ERROR, // 本地IP错误
	REMOTE_IP_ERROR, // 远程IP错误
	LOCAL_PORT_ERROR, // 本地端口错误
	REMOTE_PORT_ERROR, // 远程端口错误
	NORMAL_ERROR // 其他错误
	};
	暂定规则名为规则标志符，不可更改	

获取KSC防火墙规则(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	KDK_FW_RULE *kdk_firewall_get_rule(char *rule_name)	
描述	获取KSC防火墙规则	
参数	rule_name	防火墙规则名
返回值	KDK_FW_RULE *	麒麟防火墙规则结构体指针
	NULL	失败
备注	无	

获取KSC防火墙所有规则(自2.3.0版本启用)

子模块	麒麟防火墙
接口类型	C
原型	int kdk_firewall_get_all_rules(KDK_FW_RULE ***rules)

描述	获取KSC防火墙所有规则	
参数	rules	KDK_FW_RULE结构体数组指针
返回值	-1	失败
	其他	返回获取到的规则数量
备注	无	

导出KSC防火墙规则(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_export_rules(char *rules_path, char *rules[], int count)	
描述	导出KSC防火墙规则	
参数	rule_path	规则导出文件路径
	rules	导出规则名字符串组
	count	导出规则数目
返回值	0	成功
	-1	失败
备注	无	

导入KSC防火墙规则(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_import_rules(char *rules_path)	
描述	导入KSC防火墙规则	
参数	rule_path	规则导入文件路径
返回值	成功返回导入规则数目	
	-1	失败
备注	无	

将当前网络连接对应的网卡与防火墙模式绑定(自2.3.0版本启用)

子模块	麒麟防火墙	
接口类型	C	
原型	int kdk_firewall_bind_network(int mode, char *nic)	
描述	将当前网络连接对应的网卡与防火墙模式绑定	
参数	mode	网络模式
	nic	当前网络连接对应的网卡名称
返回值	0	成功
	-1	失败
备注	无	

- **示例代码：**

```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#include <kysdk/kysdk-security/libkyfirewall.h>

static int switch_mode(int type)
{
    int rc;
    if (type == 0) {
        rc = kdk_firewall_disable_mode(FW_PUBLIC);
        rc = kdk_firewall_disable_mode(FW_PRIVATE);
    } else if (type == 1) {
        rc = kdk_firewall_enable_mode(FW_PUBLIC);
        rc = kdk_firewall_enable_mode(FW_PRIVATE);
    }
    return rc;
}

static int test_kyfirewall_rule()
{
    int rc = 0;
    KDK_FW_RULE *p = kdk_firewall_init_rule();
    if (!p) {
        rc = -1;
        goto end;
    }

    rc = kdk_firewall_set_rule_item(p, FW_NAME, (void*)"kysdk_test");
    if (rc) { goto end; }

    rc = kdk_firewall_set_rule_item(p, FW_COMM, (void*)"all");
    if (rc) { goto end; }

    rc = kdk_firewall_set_rule_item(p, FW_DIRECTION, (void*)FW_ALL_DIRECITON);
    if (rc) { goto end; }

    rc = kdk_firewall_set_rule_item(p, FW_MODE, (void*)FW_ALL_MODE);
    if (rc) { goto end; }

    rc = kdk_firewall_set_rule_item(p, FW_PROTOCOL, (void*)"tcp");
    if (rc) { goto end; }


    rc = kdk_firewall_set_rule_item(p, FW_ACTION, (void*)FW_ALLOW);
    if (rc) { goto end; }

    rc = kdk_firewall_set_rule_item(p, FW_STATUS, (void*)FW_RULE_ON);
    if (rc) { goto end; }

    KDK_FW_RULE *tmp = kdk_firewall_get_rule("kysdk_test");
    if (!tmp) {
        rc = -1;
        goto create;
    }
    kdk_firewall_destory_rule(tmp);

    rc = kdk_firewall_delete_rule("kysdk_test");
    if (rc) { goto end; }

create:
    rc = kdk_firewall_create_rule(p);
    if (rc) { goto end; }

    rc = kdk_firewall_update_rule(p);
    if (rc) { goto end; }
```

```
    rc = kdk_firewall_enable_rule("kysdk_test");
    if (rc) { goto end; }

    rc = kdk_firewall_disable_rule("kysdk_test");
    if (rc) { goto end; }


    rc = kdk_firewall_delete_rule("kysdk_test");
    if (rc) { goto end; }

    KDK_FW_RULE **list = NULL;
    rc = kdk_firewall_get_all_rules(&list);
    if (rc != 0 && list) {
        for (int i = 0; i < rc; i++) {
            kdk_firewall_destory_rule(list[i]);
        }
        free(list);
    }

end:
    kdk_firewall_destory_rule(p);
    return rc;
}

int main()
{
    return test_kyfirewall_rule();
}
```

6.8 用户认证

操作系统用户身份识别认证

- **安装命令：**

```
$ sudo apt-get install libkysdk-userauth libkysdk-userauth-dev
```

- **构建示例：**

（1）qt .pro 构建项目

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-userauth
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKUSERAUTH kysdk-userauth)
target_include_directories(demo PRIVATE ${KYSDKUSERAUTH_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKUSERAUTH_LIBRARY_DIRS})
target_link_libraries(demo PRIVATE ${KYSDKUSERAUTH_LIBRARIES})
```

6.8.1 验证用户信息

- **头文件路径：**

```
 #include "kysdk/kysdk-security/libkyuserauth.h"
```

- **so库路径：**

```
 /usr/lib/kysdk/kysdk-security/libkyuserauth.so
```

- **子模块信息：**

基于Polkit验证用户信息(自2.3.0版本启用)

子模块	验证用户信息	
接口类型	C	
原型	int kdk_authority_check_by_polkit()	
描述	验证用户信息	
参数	无	
返回值	1	成功
	0	失败
备注	调用此接口会弹出Polkit认证窗口	

- **示例代码：**

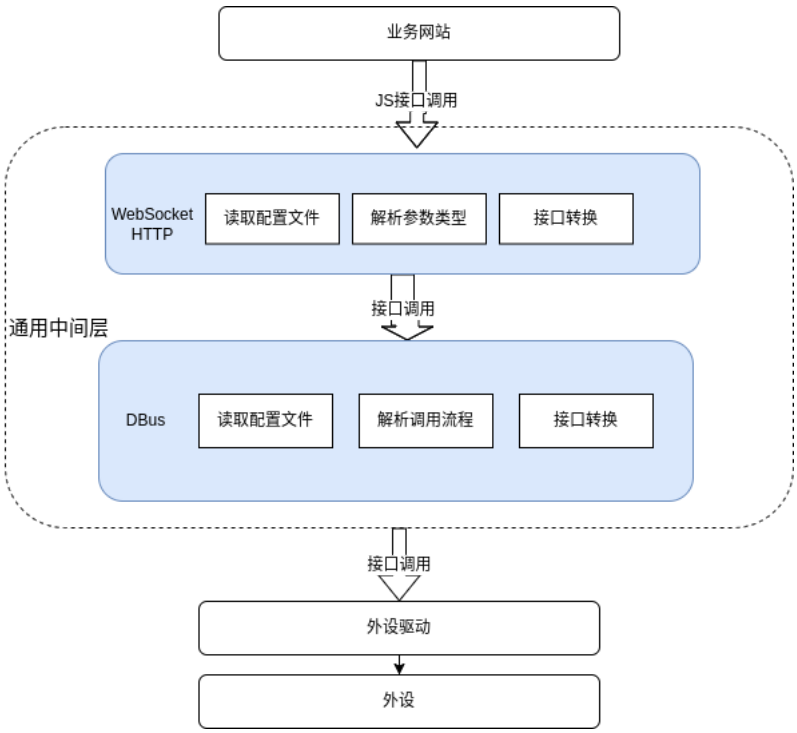
```
#----------------------------C语言示例----------------------------
#include <stdio.h>
#include <libkyuserauth.h>

int main()
{
    int auth_ret;
    auth_ret=kdk_authority_check_by_polkit();
    printf("认证结果: %s\n", auth_ret == 1 ? "通过认证" : "认证失败");
    return 0;
}
```

7 通用中间层方案

7.1 功能

- 外设调用权限
  - 通过中间层解决网页应用访问本地资源权限不足问题
- 统一调用接口
  - 统一应用调用接口、参数、返回值，减少重复性适配工作
- 配置文件
  - 针对已经适配操作系统的外设驱动，可以通过配置文件实现外设的快速接入



7.2 支持外设

7.2.1 身份证读卡器

7.2.1.1 支持接口

启动接口

接口	语言	入参	返回值说明	备注
open(devpath)	js	devpath为设备节点路径	0:调用成功；-1:调用失败	websocket提供js接口
127.0.0.1:8888/idcardreader/idopen?devpath=parameter	js	devpath为设备节点路径	{"Result": x}； Result:0为调用成功；-1为调用失败	http提供js接口

关闭接口

接口	语言	入参	返回值说明	备注
close(devpath)	js	devpath为设备节点路径	0:调用成功；-1:调用失败	websocket提供js接口
127.0.0.1:8888/idcardreader/idclose?devpath=parameter	js	devpath为设备节点路径	{"Result": x}； Result:0为调用成功；-1为调用失败	http提供js接口

获取卡信息

接口	语言	入参	返回值说明	备注
readCard()	js	无	调用失败返回:-1 调用成功返回:读卡信息， 格式为"姓名:xx,性别:xx,民族:xx, 出生日期:xx,住址:xx,身份号码:xx, 签发机关:xx,有效期限:xx,照片: 身份证照片base64"；	websocket提供js接口
127.0.0.1:8888/idcardreader/idreadCard	js	无	{"Result":x ,"ResultMessage": xx}; Result:0为调用成功；-1为调用失败 ResultMessage:读卡信息， 格式为"姓名:xx,性别:xx,民族:xx, 出生日期:xx,住址:xx,身份号码:xx, 签发机关:xx,有效期限:xx, 照片身份证:照片base64"；	http提供js接口

7.2.2 扫描仪

7.2.2.1 支持接口

接口	语言	入参	返回值说明	备注
scanOpen(devpath)	js	devpath为设备节点路径	0:调用成功；-1:调用失败	websocket提供js接口
127.0.0.1:8888/scanner/scanOpen?devpath=parameter	js	devpath为设备节点路径	{"Result":x}； Result:0为调用成功；-1为调用失败	http提供js接口

关闭接口

接口	语言	入参	返回值说明	备注
scanClose(devpath)	js	devpath为设备节点路径	0:调用成功；-1:调用失败	websocket提供js接口
127.0.0.1:8888/scanner/scanClose?devpath=parameter	js	devpath为设备节点路径	{"Result":x}； Result:0为调用成功；-1为调用失败	http提供js接口

扫描一张

接口	语言	入参	返回值说明	备注
scanOne()	js	无	调用失败返回: -1 调用成功返回:pdf文件存储路径	websocket提供js接口
127.0.0.1:8888/scanner/scanOne	js	无	{"Result":x, "ResultMessage": xx}; Result:0为调用成功；-1为调用失败 ResultMessage:pdf存放路径	http提供js接口

扫描全部

接口	语言	入参	返回值说明	备注
scanAll(type)	js	type为扫描设置（0 单面扫描；1 双面扫描）	调用失败返回: -1 调用成功返回:pdf文件存储路径	websocket提供js接口
127.0.0.1:8888/scanner/scanAll?type=parameter	js	type为扫描设置（0 单面扫描；1 双面扫描）	{"Result":x,"ResultMessage": xx}; Result:0为调用成功；-1为调用失败 ResultMessage:pdf存放路径	http提供js接口

7.2.3 打印机

7.2.3.1 支持接口

设置打印参数

接口	语言	入参	
setPrinterOptions(number_up,media,number_up_layout,sides)	js	number_up为打印页数 （例如1,2） media为纸张类型（例如A3， A4） number_up_layout为打印布局 （例如从左到右从上到下:lrtb ，从右到左从下到上:rlbt） sides为双面打印设置 （例如单面:one-sided, 双面:two-sided-long-edge）	0:调用

接口	语言	入参	
127.0.0.1:8888/printer/setPrinterOptions?number_up=parameter1&media=parameter2&number_up_layout=parameter3&sides=parameter4	js	number_up为打印页数（例如1，2） media为纸张类型（例如A3，A4） number_up_layout为打印布局（例如从左到右从上到下:lrtb，从右到左从下到上:rlbt） sides为双面打印设置（例如单面:one-sided,双面:two-sided-long-edge）	{"Resu Result

获取打印机列表

接口	语言	入参	返回值说明	备注
getPrinterList()	js	无	调用失败返回: -1 调用成功返回: 打印机列表	websocket提供js接口
127.0.0.1:8888/printer/getPrinterList	js	无	{"Result":x,"ResultMessage":"[xxx]"} Result:0为调用成功；-1为调用失败 ResultMessage为打印机列表	http提供js接口

打印文件

接口	语言	入参	返回值说明	备注
printFile(printername,filepath)	js	printername为打印机名 filepath为打印文件绝对路径	0:调用成功；-1:调用失败	websocket提供js接口
127.0.0.1:8888/printer/printFile?printername=parameter1&filepath=parameter2	js	printername为打印机名 filepath为打印文件绝对路径	{"Result":x}； Result:0为调用成功；-1为调用失败	http提供js接口

7.2.4 手写板

7.2.4.1 支持接口

打开接口

接口	语言	入参	返回值说明	备注
signOpen(devpath)	js	devpath为设备节点路径	0:调用成功；-1:调用失败	websocket提供js接口
127.0.0.1:8888/signban/signopen?devpath=parameter	js	devpath为设备节点路径	{"Result":x}； Result:0为调用成功；-1为调用失败	http提供js接口

关闭接口

接口	语言	入参	返回值说明	备注
signClose(devpath)	js	devpath为设备节点路径	0:调用成功；-1:调用失败	websocket提供js接口
127.0.0.1:8888/signban/signclose?devpath=parameter	js	devpath为设备节点路径	{"Result":x}； Result:0为调用成功；-1为调用失败	http提供js接口

开始签字接口

接口	语言	入参	返回值说明	备注
signBegin()	js	无	调用失败返回: -1 调用成功返回: 手写屏信息的base64值	websocket提供js接口

接口	语言	入参	返回值说明	备注
127.0.0.1:8888/signban/signbegin	js	无	{"Result":x,"ResultMessage": "xxx"} Result:0为调用成功；-1为调用失败 ResultMessage: 手写屏信息的base64值	http提供js接口

7.2.5 高拍仪

7.2.5.1 支持接口

> 打开摄像头

接口	语言	入参	返回值说明	备注
cameraOpen()	js	无	调用失败返回: -1 调用成功返回: 照片的base64值	websocket提供js接口
127.0.0.1:8888/camera/CameraOpen	js	无	{"Result":x,"ResultMessage": "xxx"} Result:0为调用成功；-1为调用失败 ResultMessage:照片的base64值	http提供js接口

8 桌面环境 SDK

该层设计主要为应用开发提供与桌面环境有关的工具来协助不同应用开发同种功能的开发时间。高通用性、基础性的集合。

安装命令：

```
$sudo apt install libkysdk-desktop libkysdk-desktop-dev
```

8.1 声音模块

声音模块 libkysdk-soundeffects 属于 kysdk-desktop 的子模块，安装方式如下：

```
sudo apt install libkysdk-soundeffects libkysdk-soundeffects-dev
```

根据不同项目类型，可参考以下 demo：

（1）.pro 文件构建项目

qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-soundeffects
```

（2）CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(Qt5 COMPONENTS Widgets REQUIRED)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKSOUNDEFFECTS_PKG kysdk-soundeffects)
target_include_directories(demo PRIVATE ${KYSDKSOUNDEFFECTS_PKG_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKSOUNDEFFECTS_PKG_LIBRARY_DIRS})
target_link_libraries(demo Qt5::Widgets ${KYSDKSOUNDEFFECTS_PKG_LIBRARIES})
```

在具体项目中，需在代码中引入对应的头文件以及命名空间，如：

```
#include "ksoundeffects.h"
using namespace kdk;
```

8.1.1 音效

> 功能描述：ksoundeffects，通过传入枚举参数的不同，可以在不同场景下唤起系统音效。

枚举类型	enum SoundType{ AUDIO_VOLUME_CHANGE,BATTERY_LOW,COMPLETE,DEVICE_ADDED_FAILED,DEVICE_REMOVED,DEVICE_ADDED,DIALOG_

子模块	libkysdk-soundeffects	
接口类型	C++	
原型	static void playSound(SoundType type);	
描述	通过改变参数，播放系统音效不同	
参数	参数名称	参数说明
	type	需要播放的音效的枚举
返回值	无	无
备注	无	

8.2 通知模块

该模块安装方式如下：

```
sudo apt install libkysdk-notification libkysdk-notification-dev
```

根据不同项目类型，可参考以下 demo 构建项目：
（1） .pro 文件构建项目：
qt 项目.pro 文件中增加：

```
CONFIG += link_pkgconfig
PKGCONFIG += kysdk-notification
```

（2） CMakeLists.txt 构建项目

```
cmake_minimum_required(VERSION 3.5)
find_package(Qt5 COMPONENTS Widgets REQUIRED)
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYSDKNOTIFICATION_PKG kysdk-notification)
target_include_directories(demo PRIVATE ${KYSDKNOTIFICATION_PKG_INCLUDE_DIRS})
target_link_directories(demo PRIVATE ${KYSDKNOTIFICATION_PKG_LIBRARY_DIRS})
target_link_libraries(demo Qt5::Widgets ${KYSDKNOTIFICATION_PKG_LIBRARIES})
```

8.2.1 通知

> 功能描述：消息通知类，调用notify()发消息后会在屏幕右上角以弹窗形式展示消息内容和支持的操作，支持设置应用名称，应用图标，消息标题，消息主体内容，显示时长，添加action等

> 注意：libkysdk-notification在2.3版本之前在应用支撑SDK里，2.3版本移到桌面环境SDK

枚举类型	enum SoundType {Default = -1, AllTheTime = 0 } ;
枚举值说明	Default = -1 系统默认显示消息时长， AllTheTime = 0 消息常驻

子模块	libkysdk-notification
接口类型	C++
原型	uint notify();
描述	发送消息通知请求，返回消息通知id

参数	参数名称	参数说明
	无	无
返回值	uint	返回消息通知id
备注	无	

子模块	libkysdk-notification	
接口类型	C++	
原型	void setDefaultAction(const QString& appName);	
描述	设置默认跳转应用，点击消息弹窗时进行跳转	
参数	参数名称	参数说明
	appName	应用名称
返回值	无	无
备注	无	

子模块	libkysdk-notification	
接口类型	C++	
原型	void addAction(const QString& appName, const QString& text);	
描述	在消息弹窗中添加跳转按钮，最多可以添加三个	
参数	参数名称	参数说明
	appName	应用名称
	text	文本内容
返回值	无	无
备注	无	

子模块	libkysdk-notification	
接口类型	C++	
原型	void setShowTime(int milliseconds);	
描述	设置消息弹窗的显示时长，Default为系统默认时长，AllTheTime为常驻消息	
参数	参数名称	参数说明
	milliseconds	显示时长
返回值	无	无
备注	无	

子模块	libkysdk-notification	
接口类型	C++	
原型	void setAppName(const QString& appName);	
描述	设置消息弹窗应用名称	
参数	参数名称	参数说明
	appName	应用名称
返回值	无	无
备注	无	

子模块	libkysdk-notification

接口类型	C++	
原型	void setBodyText(const QString& bodyText);	
描述	设置消息弹窗主内容	
参数	参数名称	参数说明
	bodyText	设置弹窗主内容
返回值	无	无
备注	无	

子模块	libkysdk-notification	
接口类型	C++	
原型	void setSummary(const QString& summary);	
描述	设置消息弹窗标题	
参数	参数名称	参数说明
	summary	设置弹窗标题
返回值	无	无
备注	无	

子模块	libkysdk-notification	
接口类型	C++	
原型	void setAppIcon(const QString& iconName);	
描述	设置消息弹窗应用图标，仅支持系统图标	
参数	参数名称	参数说明
	iconName	设置的图标名称
返回值	无	无
备注	无	

子模块	libkysdk-notification	
接口类型	C++	
原型	void setReplaceId(const uint id);	
描述	设置替换消息弹窗的id替换通知即更新通知内容， 在通知还未消失时，更新通知弹窗的主题、正文、 跳转动作和按钮。	
参数	参数名称	参数说明
	id	替换消息弹窗的id
返回值	无	无
备注	无	

子模块	libkysdk-notification	
接口类型	C++	
原型	static void closeNotification(uint id);	
描述	主动关闭消息弹窗	
参数	参数名称	参数说明
	id	关闭消息弹窗的id
返回值	无	无

备注	无

9 专用名词解释

1. **SDK:**(software development kit)软件开发工具包一般都是一些软件工程师为特定的软件包、软件框架、硬件平台、操作系统等建立应用软件时的开发工具的集合。
2. **封装：** 托盘 即隐藏对象的属性和实现细节，仅对外公开接口，控制在程序中属性的读和修改的访问级别；将抽象得到的数据和行为（或功能）相结合，形成一个有机的整体，也就是将数据与操作数据的源代码进行有机的结合，形成“类”，其中数据和函数都是类的成员。
3. **cpu：** 中央处理器（Central Processing Unit，简称CPU）作为计算机系统的运算和控制核心，是信息处理、程序运行的最终执行单元。
4. **x86-64：** （ 又称x64，即英文词64-bit extended，64位拓展 的简写）是x86架构的64位拓展，向后兼容于16位及32位的x86架构。
5. **主频：** 主频即CPU的时钟频率，计算机的操作在时钟信号的控制下分步执行，每个时钟信号周期完成一步操作，时钟频率的高低在很大程度上反映了CPU速度的快慢。
6. **虚拟化：** 可以让一个CPU工作起来像多个CPU在并行运行，从而使得在一部电脑内同时运行多个操作系统成为可能。
7. **线程：** 是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。
8. **网卡：** 网卡是一块被设计用来允许计算机在计算机网络上进行通讯的计算机硬件。
9. **mac地址：** 它是一个用来确认网络设备位置的位址。
10. **IPv4地址：** 又称互联网通信协议第四版，是网际协议开发过程中的第四个修订版本，也是此协议第一个被广泛部署的版本。
11. **IPv6地址：** IPv6最大的优势就在于其地址数量远大于IPv4协议 [5] ，IPv6协议将IPv6的地址长度增至128bit，分成8个部分，每个部分为16bit。
12. **bios:** Basic Input Output System，计算机在运行时，首先会进入BIOS，它在计算机系统中起着非常重要的作用。一块主板性能优越与否，很大程度上取决于主板上的BIOS管理功能是否先进。
13. **主板：** 是计算机最基本的同时也是最重要的部件之一。主板一般为矩形电路板，上面安装了组成计算机的主要电路系统，一般有BIOS芯片、I/O控制芯片、键盘和面板控制开关接口、指示灯插接件、扩充插槽、主板及插卡的直流电源供电接插件等元件
14. **序列号：** 一般为电子产品的全球唯一标识码。常用于防伪。最大的特点就是唯一。
15. **usb:**（Universal Serial Bus，缩写：USB）是一种串口总线标准，也是一种输入输出接口的技术规范。
16. **显卡：** 将计算机系统需要的显示信息进行转换驱动显示器，并向显示器提供逐行或隔行扫描信号，控制显示器的正确显示，是连接显示器和个人计算机主板的重要组件。
17. **pid：** 同进程号。
18. **可视面积：** 显示器可以显示图形的最大范围，我们平常说的17英寸、15英寸实际上指显像管的尺寸，而实际可视区域(就是屏幕)远远到不了这个尺寸。
19. **分辨率：** 是显示器在显示图像时的分辨率，分辨率是用点来衡量的，显示器上这个“点”就是指像素。
20. **虚拟内存：** 是计算机系统内存管理的一种技术。它使得应用程序认为它拥有连续的可用的内存（一个连续完整的地址空间），而实际上，它通常是被分隔成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。
21. **swap分区：** Swap分区在系统的物理内存不够用的时候，把硬盘内存中的一部分空间释放出来，以供当前运行的程序使用。那些被释放的空间可能来自一些很长时间没有什么操作的程序，这些被释放的空间被临时保存到Swap分区中，等到那些程序要运行时，再从Swap分区中恢复保存的数据到内存中。
22. **端口号：** 客户端可以通过ip地址找到对应的服务器端，但是服务器端是有很多端口的，每个应用程序对应一个端口号，通过类似门牌号的端口号，客户端才能真正的访问到该服务器。
23. **进程号：** 大多数操作系统的内核用于唯一标识进程的一个数值。
24. **内核：** 内核是操作系统最基本的部分。它是为众多应用程序提供对计算机硬件的安全访问的一部分软件，这种访问是有限的，并且内核决定一个程序在什么时候对某部分硬件操作多长时间。
25. **网关：** 又称网间连接器、协议转换器。网关在网络层以上实现网络互连，是复杂的网络互连设备，仅用于两个高层协议不同的网络互连。网关既可以用于广域网互连，也可以用于局域网互连。 网关是一种充当转换重任的计算机系统或设备。
26. **防火墙：** 技术是通过有机结合各类用于安全管理与筛选的软件和硬件设备，帮助计算机网络于其内、外网之间构建一道相对隔绝的保护屏障，以保护用户资料与信息安全性的一种技术。
27. **子网：** 为了确定网络区域，分开主机和路由器的每个接口，从而产生了若干个分离的网络岛，接口端连接了这些独立网络的端点。这些独立的网络岛叫做子网。
28. **域名：** 由一串用点分隔的名字组成的互联网上某一台计算机或计算机组的名称，用于在数据传输时对计算机的定位标识。
29. **AI:** 人工智能（Artificial Intelligence），英文缩写为AI。 它是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的技术科学。
30. **OCR:** OCR （Optical Character Recognition，光学字符识别）是指电子设备（例如扫描仪或数码相机）检查纸上打印的字符，通过检测暗、亮的模式确定其形状，然后用字符识别方法将形状翻译成计算机文字的过程。
31. **OOM:** 全称“Out Of Memory”，内存不足的。
32. **经纬度：** 经纬度是经度与纬度组成的坐标系统，是一种利用三度空间的球面来定义地球上的空间的球面坐标系统，能够标示地球上的任何一个位置。
33. **主题：** 主题的不同，用户在使用手机时感觉身历其境，不再只是面对一成不变手机操作界面、图片和色彩。
34. **链表：** 是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。
35. **json:**（JavaScript Object Notation, JS对象简谱）是一种轻量级的数据交换格式。
36. **xml:** 可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。
37. **句柄：** 是存储指向动态分配（堆）对象指针的类。

38. **埋点：** 是指在需要采集数据的“操作节点”将数据采集的程序代码附加在功能程序代码中，对操作节点上用户行为或事件进行捕获、处理和发送相关技术及其实施过程。
39. **管控：** 用来控制软件的某些行为。
40. **热点：** 手机wifi热点是将手机接收的GPRS、3G或4G信号转化为wifi信号发出去的技术，让手机、平板或笔记本等随身携带设备可以通过无线网卡或wlan模块，能够在户外或者没有网络的地方也能上网，实现网络资源共享。
41. **光驱：** 电脑用来读写光碟内容的机器，也是在台式机和笔记本便携式电脑里比较常见的一个部件。
42. **对称加密：** 采用单钥密码系统的加密方法，同一个密钥可以同时用作信息的加密和解密，这种加密方法称为对称加密。
43. **非对称加密：** 使用两把完全不同但又是完全匹配的一对钥匙—公钥和私钥。在使用不对称加密算法加密文件时，只有使用匹配的一对公钥和私钥，才能完成对明文的加密和解密过程。
44. **RSA：** 一种非对称加密算法，与对称加密算法不同的是,RSA算法有两个不同的密钥,一个是公钥,一个是私钥。
45. **AES：** 密码学中的高级加密标准（Advanced Encryption Standard，AES），又称Rijndael加密法。
46. **MD5：** 一种被广泛使用的密码散列函数，可以产生出一个128位（16字节）的散列值（hash value），用于确保信息传输完整一致。
47. **SHA1:** 是一种密码散列函数，美国国家安全局设计，并由美国国家标准技术研究所（NIST）发布为联邦数据处理标准（FIPS）。
48. **sha256：** SHA256算法使用的哈希值长度是256位。
49. **sha512:** SHA512算法使用的哈希值长度是512位。
50. **http：** 超文本传输协议（Hypertext Transfer Protocol，HTTP）是一个简单的请求-响应协议，它通常运行在TCP之上。它指定了客户端可能发送给服务器什么样的消息以及得到什么样的响应。
51. **https：** HTTPS （全称：Hypertext Transfer Protocol Secure [5] ），是以安全为目标的 HTTP 通道，在HTTP的基础上通过传输加密和身份认证保证了传输过程的安全性。