HeliosServices框架使用及API文档

背景

为了迎合QuecPython的发展, 简化用户对底层网络, 日志, 移远云服务, 定位, 媒体,等复杂的数据处理和数据处理数据保护等操作,特此我们专门用QuecPython框架来简化复杂的创建维护操作,我们想通过数据驱动来改善,我们目前复杂的编码过程, 由此发展为QuecPython框架的由来

解决的问题

携带发布订阅BUS总线

- 支持消息和处理的绑定
- 异步的消息处理机制
- 用户不需关心对应的处理函数,只管向对应的topic发消息即可

携带消息队列广播机制

- 可自定义消息类型,执行异步和同步模式,支持事件派发和事件管理, 支持标准队列和观察者模式)
- 解决了两个线程之间传递消息,以及多个事件广播的问题
- 给模块进行了标准化的解耦过程, 提供了统一稳定的事件管理中心来保证通信的质量

NET(网络):

- 提供网络服务(解决了客户网络的依赖, 客户只需要注册回调接口, 就可能发现注网和的状态,也支持主动查询, 设置APN等操作)
- 支持断网通知, 断网重连通知等,支持订阅消息
- 支持网络主动查询

LOG(日志):

- 提供标准化的日志系统(解决日志标准化的问题, 提供日志标准化的输出并携带时间,同样支持异步和同步的配置, 支持云端发送, 落盘等行为)
- 支持不同uart口输出, 落盘输出等
- 支持订阅消息发布消息

CLOUD(移远云服务):

- 提供基于移远云的OTA升级和日志发送接口(临终遗言日志上报, OTA的升级, OTA组件等, 同样支持异步和同步模式)
- 支持自动化固件升级
- 支持自动化文件升级
- 支持提交云端日志可配合我们日志系统使用

MEDIA(媒体服务)

- 提供媒体功能(支持发布异步的媒体消息, 统一媒体消息管理等等)
- 支持异步tts, audio播报
- 支持tts,audio消息发布,消息广播,消息订阅
- 支持tts, audio播报的自动管理,保证消息播报的可靠性

EXCEPTION(异常服务)

- 提供异常服务支持
- 支持同步异步消息
- 支持异常消息订, 异常消息发送

PM(低功耗服务)

- 可选的功耗管理
- 默认低功耗模式
- 支持高刷唤醒

SYS_BUS(发布订阅总线)

- 支持自定义topic实现发布订阅
- 支持发布订阅, 通过自动化的线程渠道去分配线程处理业务
- 支持高并发的处理

Queue(标准队列)

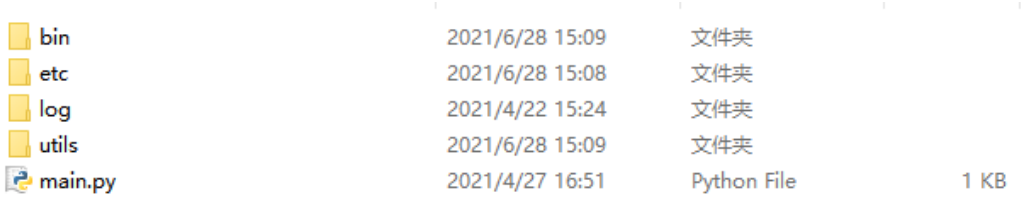
- 支持不同线程中消息的传递
- 阻塞实现, 支持多线程的put ,get 原子操作

目录结构

目录详解

- usr
  - bin
  - etc
  - utils
  - log

usr(必须存在)

- 用户目录
- 展示
  - 

bin

- 脚本和启动器目录(下面放着一些公有服务,和一些公有的组件)

components

- 一些公共组件目录

etc(非必须存在)

- 配置文件目录

app_config

- APP业务服务配置文件目录

system_config

- 系统服务配置文件目录

log(非必须存在)

- 日志存储服务
- 末日选择日志存储为本地存储时会自动创建

utils(必须存在)

- 通用工具目录(目前放着通用的工具类等)

目录展示

main.py(用户代码)

消息队列总线API

sys_bus消息总线(用于发布订阅)

订阅

> **subscribe(topic, callback)**

参数

参数	类型	含义
topic	string | int	主题
callback	function	订阅的函数, [订阅函数的参数必须有两个一个(topic, msg)]

示例

```
import sys_bus

def cb_callback(topic, msg):
    print(topic, msg)

# 支持一个topic 可以注册多个订阅函数
sys_bus.subscribe("topic1", cb_callback)
```

发布

> **publish(topic, msg)**

参数

参数	类型	含义
topic	string | int	主题
msg	void	发往主题的信息

示例

```
import sys_bus

"发布后订阅者会收到消息"
sys_bus.publish("topic1", "this is a msg")
```

解绑

> unsubscribe(topic, callback=None)

参数

参数	类型	含义
topic	string | int	主题
callback	function	解绑的函数,默认是None,不传默认解绑主题, 传默认解绑主题下的这个回调函数

示例

```
import sys_bus

# 解绑topic 下的cb_callback函数
sys_bus.unsubscribe("topic1", cb_callback)
# 解绑topic
sys_bus.unsubscribe("topic1")
```

查看注册表

> sub_table(topic=None)

参数

参数	类型	含义
topic	string | int	不传默认查看所有topic,返回k:v字典, 传则返回对应topic下的订阅函数集合

示例

```
import sys_bus

sys_bus.sub_table()
# 返回 {"topic1": set(cb_callback...)}

sys_bus.sub_table("topic1")
# 返回 set(cb_callback...)
```

Queue普通队列(用于线程通信)

提供队列服务

创建队列对象

> from queue import Queue
>
> q = Queue(maxsize=100)

- 参数

参数	参数类型	参数说明
maxsize	int	队列最大长度,默认值100

入队列

> q.put(item)

- 参数

参数	参数类型	参数说明
item	void	入队列的内容

- 返回值
  - True: 成功
  - False: 失败

出队列

会阻塞等待, get每次有put数据get会收到通知

> q.get()

- 参数

    - 无
- 返回值
    - 返回put的数据

综合示例

```
import _thread
from queue import Queue
# 初始化队列
q = Queue(maxsize=100)

def get():
    while True:
        # q.get 会阻塞等待消息过来每当有q.put执行完后 q.get会受到相关信号,解除阻塞往下执行
        item = q.get()
        print(item)

# 开启线程在哪等待消息
_thread.start_new_thread(get,())
# put消息到队列
q.put(1)
q.put(2)
```

服务API使用

GUARD使用

全局监控和配置文件的启动容器

获取框架版本号

- 获取框架版本号

version

```
>>>from usr.bin.guard import GuardContext
>>> GuardContext().version()
{'ARTIFACT_ID': 'qpy-framework', 'VERSION': '1.0.0.RELEASE', 'GROUP_ID': 'qpy.quectel.com'}
```

- 返回值

返回值	类型	说明
GROUP_ID	string	归属组织地址
ARTIFACT_ID	string	名称id
VERSION	string	版本号从1.0.0.RELEASE开始

初始化

- 初始化全局的guard环境

```
>>> from usr.bin.guard import GuardContext
>>> guard_context = GuardContext()
```

刷新容器

- 刷新容器并启动的所有服务(这里的服务启动的是非等级等于3的服务)

refresh

```
>>> guard_context.refresh()

[    OK     ] create sys monitor net service
[    OK     ] create sys monitor net service

[    OK     ] create app monitor media service
[    OK     ] create app monitor exception service

[   FAILED  ] load cloud monitor error reason:[cloud service load error]
```

- 输出
  - 相关服务的启动状态

状态	显示	描述
服务创建成功	[OK] XXX	服务创建成功
服务创建失败	[FAILED] XXX	服务创建失败
警告	[WARN] XXX	服务创建过程警告

查看所有服务

servers

- 获取所有服务

```
>>> guard_context.servers()
{'media': <MediaServiceMonitor object at 7eb6df60>, 'log':
<LogServiceMonitor object at 7eb6a550>, 'net': <NetServiceMonitor object at
7eb64e10>, 'exception': <ExceptionServiceMonitor object at 7eb6b0d0>}
```

目前存在的服务

服务名称	是否一定存在	描述
log	是	默认refresh初始化
media	是	默认refresh初始化
net	是	默认refresh初始化
exception	是	默认refresh初始化
cloud	否	默认cloud需要自己传入配置文件 [1. 主动配置传入, 2. 在etc/app_config/cloud/config.json中写入] refresh或者reload进行初始化

获取服务

get_server

- 获取服务

> guard_context.get_server(server_name)

- 参数

参数	参数类型	参数说明
server_name	服务名称	通过服务名称获取服务

- 返回值
  - 对应的服务对象

示例

```
>>> guard_context.get_server("log")
<LogService object at 7eb655a0>
```

手动注册服务

register_monitor

用于手动注册服务

```
guard_context.register_monitor(server_name, monitor)
```

- 参数

参数	类型	描述
server_name	string	服务名称
monitor	class	继承于MonitorInterface,并包裹server的启动器

- 返回值
  - True 注册成功
  - False 注册失败

重载服务

reload

- 重新加载服务,当我们手动注册服务或者有服务优先级启动的时候需要调用reload去重载服务

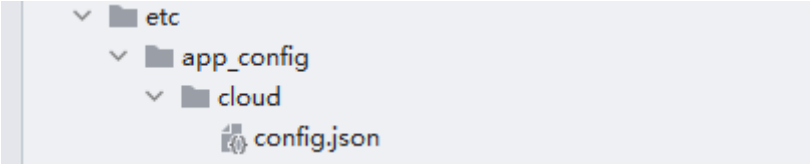
```
guard_context.reload()
```

- 参数
  - 无
- 返回值
  - 无

获取app/json内容

当我们在/usr/etc/app_config目录下设置app应用的配置文件时, guard初始化时会默认帮我们读出来, 对应目录格式

- /usr
  - /etc
    - /app_config
      - /app_name[应用名, 必须英文]
        - config.json



service_config

- 里面有所有app_config下面的json内容, 对应的关系是 {app_name: json_content}

> guard_context.service_config

示例

```
>>> guard_context.service_config["cloud"]
```

Net服务

获取网络服务

```
>>> net_ser = guard_context.get_server("net")
>>> net_ser
<NetService object at 7eb62de0>
```

订阅网络服务

subscribe

订阅服务信息, 主要是网络的断开和重新连接连接的信号

> net_ser.subscribe(callback)

- 参数

参数	类型	描述
callback	function	订阅的函数, 注意函数的参数必须是(*args, **kwargs)

- 返回值
  - 无

解除订阅

unsubscribe

解除订阅的函数

> net_ser.unsubscribe(callback)

- 参数

参数	类型	描述
callback	function	订阅的函数, 注意函数的参数必须是(*args, **kwargs)

- 返回值
  - 无

MEDIA服务

获取媒体服务

- 音频播放，支持TTS、mp3以及AMR文件播放。

```
>>> media_ser = guard_context.get_server("media")
>>> media_ser
<MediaService object at 7eb6ae80>
```

设置模式

- 设备类型，0 - 听筒，1 - 耳机，2 - 喇叭。**[默认是0听筒模式]**

> media_ser.set_mode(mode)

- 示例

```
>>> media_ser.set_mode(0)
```

获取audio

可获取原生audio对象, 用作一些额外处理

```
>>> media_ser.audio
```

- 返回
  - audio对象

获取tts

可获取原生audio对象, 用作一些额外处理

```
>>> media_ser.tts
```

- 返回
  - tts对象

设置PA

设置输出的pa的gpio引脚，并开启pa功能目前支持AB类切D类，即两个上升沿的脉冲分别在1us<脉冲<12us

> media_ser.set_pa("priority=4, breakin=0, mode=2, play_data="")

- 参数

参数	参数类型	参数说明
gpio	int	设置输出的gpio，gpio可从Pin里面获取

- 返回值

设置成功返回整数1;

设置失败返回整数0;

- 示例

```
>>> media_ser.set_pa(Pin.GPIO15)
1
```

订阅媒体服务

subscribe

订阅服务信息, 主要是网络的断开和连接的信号

> media_ser.subscribe(callback)

- 参数

参数	类型	描述
callback	function	订阅的函数, 注意函数的参数必须是(*args, **kwargs)

- 返回值
  - 无

发布媒体消息

发布audio消息

音频文件播放，支持mp3、amr和wav格式文件播放。支持优先级0~4，数字越大优先级越高，每个优先级组可同时最多加入10个播放任务，与TTS播放共用同一个播放队列,

会默认播报,

订阅的消息也会同时收到,播报的事件和数据

```
media_ser.audio_play("priority=4, breakin=0，play_data="")
```

- 参数

参数	参数类型	参数说明
priority	int	播放优先级，支持优先级0~4，数值越大优先级越高
breakin	int	打断模式，0表示不允许被打断，1表示允许被打断
play_data	string	待播放的文件名称，包含文件存放路径

- 返回值
  - 无

发布TTS消息

语音播放，支持优先级0~4，数字越大优先级越高，每个优先级组可同时最多加入10个播放任务；播放策略说明如下：

1. 如果当前正在播放任务A，并且允许被打断，此时有高优先级播放任务B，那么会打断当前低优先级播放任务A，直接播放高优先级任务B；
2. 如果当前正在播放任务A，并且不允许被打断，此时有高优先级播放任务B，那么B播放任务将会加入到播放队列中合适的位置，等待A播放完成，再依次从队列中按照优先级从高到低播放其他任务；
3. 如果当前正在播放任务A，且不允许被打断，此时来了一个同优先级播放任务B，那么B会被加入到该优先级组播放队列队尾，等待A播放完成，再依次从队列中按照优先级从高到低播放其他任务；
4. 如果当前正在播放任务A，且允许被打断，此时来了一个同优先级播放任务B，那么会打断当前播放任务A，直接播放任务B；
5. 如果当前正在播放任务A，且任务A的优先级组播放队列中已经有几个播放任务存在，且该优先级组播放队列最后一个任务N是允许被打断的，此时如果来了一个同样优先级的播放任务B，那么任务B会直接覆盖掉任务N；也就是说，某个优先级组，只有最后一个元素是允许被打断的，即breakin为1，其他任务都是不允许被打断的；
6. 如果当前正在播放任务A，不管任务A是否允许被打断，此时来了一个优先级低于任务A的请求B，那么将B加入到B对应优先级组播放队列。

订阅的消息也会同时收到,播报的事件和数据

> media_ser.tts_play("priority=4, breakin=0, mode=2, play_data="")

- 参数

参数	参数类型	参数说明
priority	int	播放优先级，支持优先级0~4，数值越大优先级越高
breakin	int	打断模式，0表示不允许被打断，1表示允许被打断
mode	int	编码模式，1 - UNICODE16(Size end conversion)，2 - UTF-8，3 - UNICODE16(Don't convert)
play_data	string	待播放字符串

解除订阅

unsubscribe

解除订阅的函数

> media_ser.unsubscribe(callback)

- 参数

参数	类型	描述
callback	function	订阅的函数, 注意函数的参数必须是(*args, **kwargs)

- 返回值
  - 无

LOG服务

获取LOG服务

```
>>> log_ser = guard_context.get_server("log")
>>> log_ser
<LogService object at 7eb655a0>
```

设置日志输出等级

设置日志输出等级

DEBUG<INFO<WARINING<ERROR<CRITICAL

> log_ser.set_level(level)

- 参数

参数	类型	描述
level	string	设置输出级别,默认是DEBUG级别,

- 返回值
  - 无

示例

```
>>> from usr.bin.log_service import LOG_LV
>>> log_ser.set_level(LOG_LV.DEBUG)
```

设置输出位置

可以设置输出的位置, 目前支持

- 输出到中端**[默认输出到中端]**
- 输出到文件,支持输出到文件中
- 支持输出到不同的uart口

输出到终端

默认就是输出到终端

```
#### 订阅日志消息

**subscribe**

  订阅日志消息, 接收到日志消息时候,订阅的消息能拿到相关的数据

> log_ser.subscribe(callback)

- 参数

| 参数      | 类型      | 描述                                                 |
| --------- | --------- | ---------------------------------------------------- |
| callback | function | 订阅的函数, 注意函数的参数必须是(*args, **kwargs) |

- 返回值
  - 无
```

初始化日志客户端

获取日志客户端

•```python
```
>>> from usr.bin.guard import GuardContext
>>> guard_context = GuardContext()
>>> lg_adapter = guard_context.getLogger("main")
```

获取日志客户端

getLogger

> guard_context.get_logger(name)

- 参数

参数	类型	描述
name	string	客户端名称

发布日志消息

DEBUG日志

> lg_adapter.debug(msg)

INFO日志

> lg_adapter.debug(msg)

WARINING日志

> lg_adapter.debug(msg)

ERROR日志

> lg_adapter.debug(msg)

CRITICAL日志

> lg_adapter.debug(msg)

- 参数

参数	类型	描述
msg	string	要输出的日志

解除订阅

unsubscribe

解除订阅的函数

> log_ser.unsubscribe(callback)

- 参数

参数	类型	描述
callback	function	订阅的函数, 注意函数的参数必须是(*args, **kwargs)

- 返回值
  - 无

EXCEPTION服务

获取异常服务

```
>>> exception_ser = guard_context.get_server("exception")
>>> exception_ser
<ExceptionService object at 7eb63860>
```

订阅异常消息

subscribe

订阅日志消息, 接收到日志消息时候,订阅的消息能拿到相关的数据

> exception_ser.subscribe(callback)

- 参数

参数	类型	描述
callback	function	订阅的函数, 注意函数的参数必须是(*args, **kwargs)

- 返回值
  - 无

发布异常消息

发布消息, 订阅异常消息将会收到发布的信号

> exception.publish(msg)

- 参数

参数	类型	描述
msg	string	异常的消息

- 返回值
  - 无

解除订阅

unsubscribe

解除订阅的函数

> exception_ser.unsubscribe(callback)

- 参数

参数	类型	描述
callback	function	订阅的函数, 注意函数的参数必须是(*args, **kwargs)

- 返回值
  - 无

CLOUD服务

获取云服务

```
>>> cloud_ser = guard_context.get_server("cloud")
>>> cloud_ser
<CloudService object at 7eb63860>
```

订阅云服务消息

subscribe

订阅云服务消息, 将会受到云服务的相关消息

> cloud_ser.subscribe(callback)

- 参数

参数	类型	描述
callback	function	订阅的函数, 注意函数的参数必须是(*args, **kwargs)

- 返回值
  - 无

解除订阅

unsubscribe

解除订阅云服务消息

```
cloud_ser.unsubscribe(callback)
```

- 参数

参数	类型	描述
callback	function	订阅的函数, 注意函数的参数必须是(*args, **kwargs)

- 返回值
  - 无

提交云端日志

进阶篇有详细概述~~~

commit_log

提交云端日志

```
cloud_ser.commit_log(message)
```

- 参数

参数	类型	描述
message	function	上传的消息

- 返回值
  - 无

获取是否升级标志

get_app_code

获取应用升级的标志

> cloud_ser.get_app_code()

- 参数
  - 无
- 返回值

参数	描述
3001	有升级计划
3002	无升级计划
4001	获取token失败
4002	获取升级地址失败
4003	下载固件失败
4004	参数获取失败
4005	模式不存在
4021	保存配置错误
4022	获取配置错误
4023	网络错误
4024	请求服务器失败

PM低功耗服务

获取低功耗服务, `默认启用低功耗`, 不需要设置

```
>>> pm_ser = guard_context.get_server("pm")
>>> pm_ser
<PmService object at 7eb63860>
```

静止休眠锁

pm低功耗投票静止休眠

> pm_ser.lock()

- 参数
  - 无

- 返回值
  - 无

释放休眠锁

释放休眠锁

```
pm_ser.unlock()
```

- 参数
  - 无
- 返回值
  - 无

设置自动修眠

flag = 1自动休眠, flag=0 不自动休眠

```
pm_ser.auto_sleep(flag)
```

- 参数

参数	类型	说明
flag	int	1为自动休眠, 0为不自动休眠

- 返回值
  - 无