

Helios Services指南(2)-进阶

Helios Services进阶概述

进阶篇,我们主要描述各个功能是如何使用的,从入门篇我们已经帮环境搭好了,从现在起,我们将正式开始和使用上述功能,并展示使用成果


Helio Services使用文档

容器

主要是包裹和集成所有服务和配置的集中化管理的东西,下面演示起核心使用,其他功能请查看API文档

启动容器

- [OK] 为启动成功
- [FAILED]为服务启动失败
- sys 和 app 代表服务所属的分区,sys是系统级别的, app是用户级别的
- 启动成功的服务我们就可以调用了,我们也可以查看所有启动成功的服务



```
1 >>> from usr.bin.guard import GuardContext
2 >>> guard_context = GuardContext()
3 >>> guard_context.refresh()
4 [ OK ] create sys monitor net service
5 [ OK ] create sys monitor log service
6 [ OK ] create app monitor media service
7 [ OK ] create app monitor exception service
8 [ FAILED ] load cloud monitor error reason:[cloud service load error]
9 [ OK ] create app monitor pm service
10 >>> guard_context.monitor_map
11 {'pm': <PMServiceMonitor object at 7e97ce70>, 'exception': <ExceptionServiceMonitor object at 7e979fb0>, 'log': <LogServiceMonitor
object at 7e97cd0d>, 'media': <MediaServiceMonitor object at 7e97caf0>}
12 >>>
```

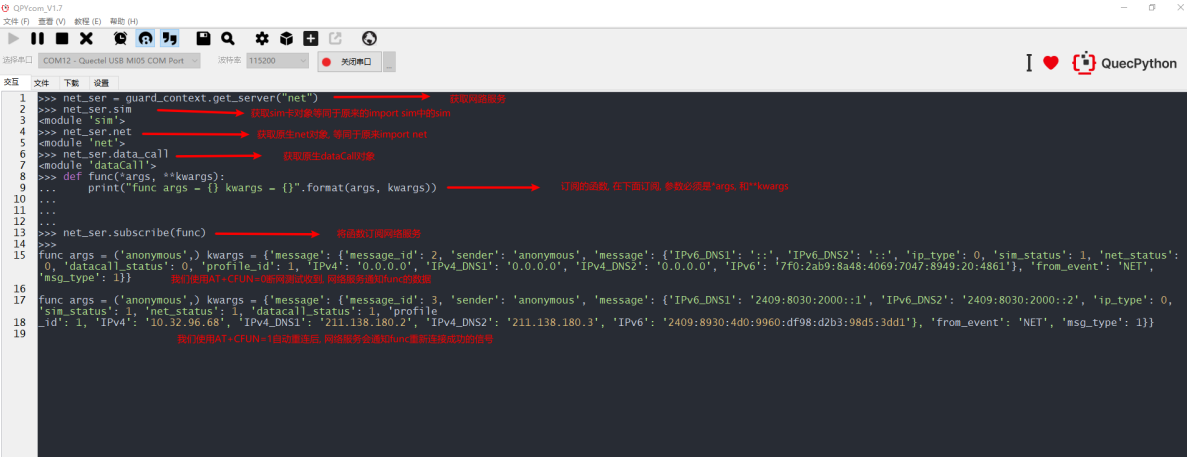
刷新容器,就是将所有容器启动

查看所有服务的监控状态,和启动状态

网络服务

基本使用

- 我们可以获取到sim, net, dataCall对象无需导入
- 我们可以订阅网络状态



```
1 >>> net_ser = guard_context.get_server("net")
2 >>> net_ser.sim
3 <module 'sim'>
4 >>> net_ser.net
5 <module 'net'>
6 >>> net_ser.data_call
7 <module 'dataCall'>
8 >>> def func(*args, **kwargs):
9 ...     print("func args = {} kwargs = {}".format(args, kwargs))
10 ...
11 ...
12 ...
13 >>> net_ser.subscribe(func)
14 >>>
15 func args = ('anonymous',) kwargs = {'message': {'message_id': 2, 'sender': 'anonymous', 'message': {'IPv6_DNS1': '::', 'IPv6_DNS2': '::', 'ip_type': 0, 'sim_status': 1, 'net_status': 0, 'datacall_status': 0, 'profile_id': 1, 'IPv4': '0.0.0.0', 'IPv4_DNS1': '0.0.0.0', 'IPv4_DNS2': '0.0.0.0', 'IPv6': '::', 'IPv6_DNS1': '2409:8030:2000::1', 'IPv6_DNS2': '2409:8030:2000::2', 'ip_type': 0, 'sim_status': 1, 'net_status': 1, 'datacall_status': 1, 'profile_id': 1, 'IPv4': '10.32.96.68', 'IPv4_DNS1': '211.138.180.2', 'IPv4_DNS2': '211.138.180.3', 'IPv6': '2409:8930:4d0:9960:d98:d2b3:98d5:3dd1', 'from_event': 'NET', 'msg_type': 1}}
16 func args = ('anonymous',) kwargs = {'message': {'message_id': 3, 'sender': 'anonymous', 'message': {'IPv6_DNS1': '2409:8030:2000::1', 'IPv6_DNS2': '2409:8030:2000::2', 'ip_type': 0, 'sim_status': 1, 'net_status': 1, 'datacall_status': 1, 'profile_id': 1, 'IPv4': '10.32.96.68', 'IPv4_DNS1': '211.138.180.2', 'IPv4_DNS2': '211.138.180.3', 'IPv6': '2409:8930:4d0:9960:d98:d2b3:98d5:3dd1', 'from_event': 'NET', 'msg_type': 1}}
17
18
19
```

获取网络服务

获取sim对象等同于原来import sim的sim

获取原生net对象, 等同于原来import net

订阅原生dataCall对象

订阅的函数, 在下面订阅, 参数必须是args 和 **kwargs

所建函数订阅网络服务

我们使用AT+CFUN=0能网络成功到, 网络服务通知func的数据

我们使用AT+CFUN=1启动网络后, 网络服务在通知func最新连接成功的信号

检测联网

- 开启后可检查网络状态, 返回值请参考 [HeliosService API文档](#)

```
交互 文件 下载 设置
1 >>> net_ser.wait_connect(10)
2 =====
3 PROJECT_NAME : QuecPython_Helios_Framework
4 PROJECT_VERSION : this latest version
5 FIRMWARE_VERSION : EC600NCNLR01A04M08_PY
6 POWERON_REASON : 2
7 SIM_CARD_STATUS : 1
8 =====
9 (3, 1)
10 >>>
```

其他功能

请参考AP文档进行调用

日志服务

基本使用

- 获取一个带命名的日志客户端
- 输出日志
- 设置日志输出级别

```
1 >>> test_log = guard_context.get_logger("test") → 获取test日志
2 >>> test_log.debug("log1") → 输出test的debug日志
3
4 2021-08-27 10:19:36 Friday test [DEBUG] - log1
5
6 >>> dev_log = guard_context.get_logger("dev")
7 >>> dev_log.debug("dev log")
8 2021-08-27 10:19:57 Friday dev [DEBUG] - dev log
9
10 >>> dev_log.error("dev log")
11 2021-08-27 10:20:02 Friday dev [ERROR] - dev log
12
13
14 >>> log_ser = guard_context.get_server("log") → 获取日志服务
15 >>> log_ser.set_level("WARNING") → 设置日志输出级别,
16 >>> dev_log.debug("dev log") 只有大于或等于此级别的日志才会被输出
17 >>> dev_log.error("dev log")
18 2021-08-27 10:25:54 Friday dev [ERROR] - dev log
19
20 >>>
```

- 订阅日志

```
9
10 >>> def func(*args, **kwargs): → 订阅的函数
11 ...     print("func args = {} kwargs = {}".format(args, kwargs))
12 ...
13 ...
14 ...
15 >>> log_ser.subscribe(func) → 订阅日志服务
16 >>> dev_log.error("dev log")
17 2021-08-27 10:37:41 Friday dev [ERROR] - dev log
18
19 func args = ('anonymous',) kwargs = {'message': {'message_id': 6, 'sender': 'anonymous', 'message': '2021-08-27 10:37:41 Friday dev [ERROR] - dev log\n', 'from_event': 'LOG',
20 'msg_type': 255}}
21 >>>
22 → 输出日志的时候,订阅的函数收到消息
```

其他功能

请参考AP文档进行调用

媒体服务

基本使用

- 获取原生tts,audio
- 订阅消息
- 设置模式
- 设置pa [参考API文档](#)

```
1 >>> media_ser = guard_context.get_server("media")
2 >>> media_ser.tts
3 tts
4 >>> media_ser.audio
5 <Audio>
6 >>> media_ser.tts.play(play_data="123213")
7 >>> media_ser.audio.play(play_data="U:/media/0.mp3")
8 media_ser.audio.play(play_data="U:/media/0.mp3")
9 >>> media_ser.set_tts(0)
10 >>> media_ser.set_audio(0)
11 >>> media_ser.set_mode(0)
12 >>> def func(*args, **kwargs):
13 ...     print("func args = {} kwargs = {}".format(args, kwargs))
14 ...
15 ...
16 ...
17 ...
18 >>> media_ser.subscribe(func)
19 >>> media_ser.audio.play(play_data="U:/media/0.mp3")
20 func args = ('anonymous',) kwargs = {'message': {'message_id': 4, 'sender': 'anonymous', 'message': {'breakin': 0, 'play_data': 'U:/media/0.mp3', 'priority': 4}, 'from_event': 'MEDIA', 'msg_type': 0}}
21 >>> media_ser.tts.play(play_data="123213")
22 func args = ('anonymous',) kwargs = {'message': {'message_id': 5, 'sender': 'anonymous', 'message': {'play_data': '123213', 'breakin': 0, 'mode': 2, 'priority': 4}, 'from_event': 'MEDIA', 'msg_type': 1}}
23 >>>
```

获取媒体服务
获取原生tts对象
获取原生audio对象
播放tts
播放audio
set_tts只设置tts的模式
set_audio只设置audio的模式
set_mode同时设置tts和audio的模式
收到服务的信息。
msg_type=0是代表来自audio的信息
订阅媒体服务
播放audio
msg_type=1代表来自tts的信息

其他功能

- 请查看API文档

低功耗服务

基本使用

- 默认是开启低功耗模式
- 存在投票机制
- 查看偷拍不为0即无法进入休眠
- 可以取消投票, 每一张投票都需要一次取消

```
1 >>> pm_ser = guard_context.get_server("pm")
2 >>> pm_ser.count()
3 0
4 >>> pm_ser.lock()
5 >>> pm_ser.lock()
6 >>> pm_ser.lock()
7 >>> pm_ser.count()
8 3
9 >>> pm_ser.unlock()
10 >>> pm_ser.count()
11 2
12 >>> pm_ser.unlock()
13 >>> pm_ser.count()
14 1
15 >>> |
```

获取pm服务, 默认低功耗
查看投票
投票, 静止睡眠
取消投票
当票数为0的时候进入低功耗

其他功能

请查看API相关文档

移远云服务

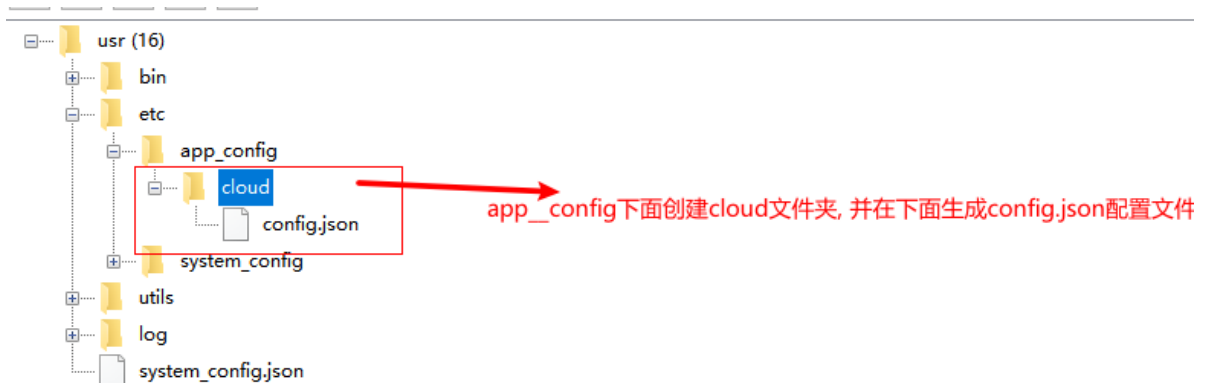
基本使用

- 提供日志上传至移远云
- OTA单文件升级

```
交互 文件 下载 设置
1 >>> guard_context.refresh()
2 [ OK ] create sys monitor net service
3 [ OK ] create sys monitor log service
4 [ OK ] create app monitor media service
5 [ OK ] create app monitor exception service
6 [ FAILED ] load cloud monitor error reason:[cloud service load error]
7 [ OK ] create app monitor pm service
8 >>>
```

每次刷新的时候都会失败, 是由于服务配置的问题

创建cloud服务的config.json cloud服务是在app_config下面的



cloud服务config.json的内容

```
{
  "level": 3,
  "params": {
    "uid": "96",
    "module_type": "EC600S-CNLB",
    "pk": "1c8723dec360166f33a633f1e2bf0142"
  }
}
```

可选择写或者不写, 等级为3的时候, 是需要用户调容器guard_context.reload()重载的时候才能使用

uid云端获取

模组型号, 云端获取

项目秘钥, 云端获取

由于当时设置的level是3即使这里创建成功也是不能使用的, 所以我们需要, reload服务才会开启自动升级服务, 设计的初衷是让一些服务的启动等级变低, 客户去控制器启动顺序

```
交互 文件 下载 设置
1 >>> guard_context.refresh()
2
3 [ OK ] create sys monitor net service
4 [ OK ] create sys monitor log service
5 [ OK ] create app monitor media service
6 [ OK ] create app monitor exception service
7 [ OK ] create app monitor cloud service
8 [ OK ] create app monitor pm service
9 >>>
```

```
1 >>> ccloud_ser = guard_context.get_server("cloud")
2 >>> guard_context.reload()
3 >>> INFO:ota:uri = http://220.180.239.212:8274/v1/oauth/token?imei=869537059549171&secret=a6b5fcbaba366678dcf6ba7c396d0a
4
5 INFO:ota:get_token = {'code': 200, 'msg': 'OK', 'data': {'expire_in': 85236, 'access_token':
6 'eyJhbGciOiJIUzI1NiIsInR5cGU6IjY2Z29wZXMiOjIjcmVhdGU1LCJpc3M1OjIjcmVhdGVjIiwiaWF0IjoxNjMwMDQzNTE1LCJleHA1OjE2MzAxMjk5MTV9.MorukKakUBE6uMrKq1LnuhEEV5dMBEbcra5mHEPG2
7 r8'}}
8 INFO:ota:the device app currently version is V1.0.0
9 INFO:ota:module_type == EC6005-CNLB
10 INFO:ota:version
11 INFO:ota:uri = http://220.180.239.212:8274/v2/oauth/token?imei=869537059549171&secret=a6b5fcbaba366678dcf6ba7c396d0a
12 INFO:ota:no upgrade plan for this device
13
14 >>> ccloud_ser.get_app_code()
15 '3002'
16 >>>
```

- 提交日志

```
>>> ccloud_ser.commit_log("23545")
INFO:ota:uri = http://220.180.239.212:8274/v2/oauth/token?imei=869537059549171&secret=a6b5fcbaba366678dcf6ba7c396d0a
INFO:ota:get_v2_token = {'code': 200, 'msg': 'OK', 'data': {'expire_in': 85079, 'access_token':
'eyJhbGciOiJIUzI1NiIsInR5cGU6IjY2Z29wZXMiOjIjcmVhdGU1LCJpc3M1OjIjcmVhdGVjIiwiaWF0IjoxNjMwMDQzNTE1LCJleHA1OjE2MzAxMjk5MTV9.MorukKakUBE6uMrKq1LnuhEEV5dMBEbcra5mHEPG2
r8'}}
>>> |
```

其他功能

请查看API文档

异常服务

基本使用

- 订阅异常服务
- 发布异常

```
>>> exc_ser = guard_context.get_server("exception")
>>> def func(*args, **kwargs):
...     print("args = {} kwargs = {}".format(args, kwargs))
...
...
>>> exc_ser.subscribe(func)
>>> exc_ser.publish("error")
error
args = ('anonymous',) kwargs = {'message': {'message_id': 2, 'sender': 'anonymous', 'message': 'error', 'from_event': 'EXCEPTION', 'msg_type': 255}}
>>> |
```

三方组件

主要有gpio, 中断, 定时器, 看门狗的封装, 还有安全组件, 断传的封装

请查看相关API文档

sys_bus使用

- 订阅topic的函数
- 发布数据

```
>>> def func(topic, msg):
...     print("func topic = {} msg = {}".format(topic, msg))
...
...
>>> sys_bus.subscribe("topic2", func)
>>> sys_bus.publish("topic2", "this is a topic2 msg")
>>> func topic = topic2 msg = this is a topic2 msg
```

裁剪说明

- 圈出来的, 带_service下坠的都是可以裁剪的, 如果不想用此服务, 删除即可

