

**QuePython**

**电表软件设计文档**

**LTE系列**

版本：QuecPython电表软件设计文档\_V1.0

日期：2022-10-26

 状态：临时文件

www.quectel.com

|  |
| --- |
|  |
|  |

上海移远通信技术股份有限公司始终以为客户提供最及时、最全面的服务为宗旨。如需任何帮助，请随时联系我司上海总部，联系方式如下：

上海移远通信技术股份有限公司

上海市闵行区田林路1016号科技绿洲3期（B区）5号楼 邮编：200233

电话：+86 21 51086236 邮箱：[info@quectel.com](mailto:info@quectel.com)

或联系我司当地办事处，详情请登录：

<http://quectel.com/cn/support/sales.htm>

如需技术支持或反馈我司技术文档中的问题，可随时登陆如下网址：

<http://quectel.com/cn/support/technical.htm>

或发送邮件至：[support@quectel.com](mailto:support@quectel.com)

**前言**

上海移远通信技术股份有限公司提供该文档内容用以支持其客户的产品设计。客户须按照文档中提供的规范、参数来设计其产品。由于客户操作不当而造成的人身伤害或财产损失，本公司不承担任何责任。在未声明前，上海移远通信技术股份有限公司有权对该文档进行更新。

**版权申明**

本文档版权属于上海移远通信技术股份有限公司，任何人未经我司允许而复制转载该文档将承担法律责任。

版权所有 ©上海移远通信技术股份有限公司 2019，保留一切权利。

***Copyright ©*** ***Quectel Wireless Solutions Co., Ltd. 2019.***

# 文档历史

**修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **日期** | **作者** | **变更表述** |
| 1.0 | 2022-10-26 | Pawn | 初始版本 |

# 目录

[文档历史 3](#_Toc121572098)

[目录 4](#_Toc121572099)

[1 前言 5](#_Toc121572100)

[2 系统框架 6](#_Toc121572101)

[2.1. 硬件系统框架 6](#_Toc121572104)

[2.2. 软件系统框架 6](#_Toc121572105)

[3 关键组件 8](#_Toc121572106)

[3.1. EventMesh 8](#_Toc121572107)

[3.2. RFC1662ProtocolResolver 10](#_Toc121572108)

[3.3. ConfigStoreManager 12](#_Toc121572109)

[3.4. NetStateManager 13](#_Toc121572110)

[3.5. TcpModeManager 13](#_Toc121572111)

[3.6. MainUartManager 14](#_Toc121572112)

[3.7. DeviceInfoManager 16](#_Toc121572113)

[3.8. FtpOtaManager 16](#_Toc121572114)

[4 系统初始化流程 18](#_Toc121572115)

[5 业务流程 19](#_Toc121572116)

[6 功能示例 20](#_Toc121572117)

# 

1. 前言

此文档用于描述电表软件设计框架,包含核心组件功能描述，系统初始化流程的介绍等，方便更快理解本框架。

1. 系统框架
3. 1. 硬件系统框架

系统框架如下：

1. Module侧支持SIM，GPIO，UART等功能。
2. Module通过UART与MCU通信完成数据交互。

图表, 瀑布图

中度可信度描述已自动生成

* 1. 软件系统框架

软件系统框架如下:

1. APP层实现处理电表核心业务，解析上下行数据
2. EventMesh为事件处理器，通过支持事件订阅发布的机制来完成功能流转
3. Module侧接收外部事件或数据通过EventMesh驱动来处理执行

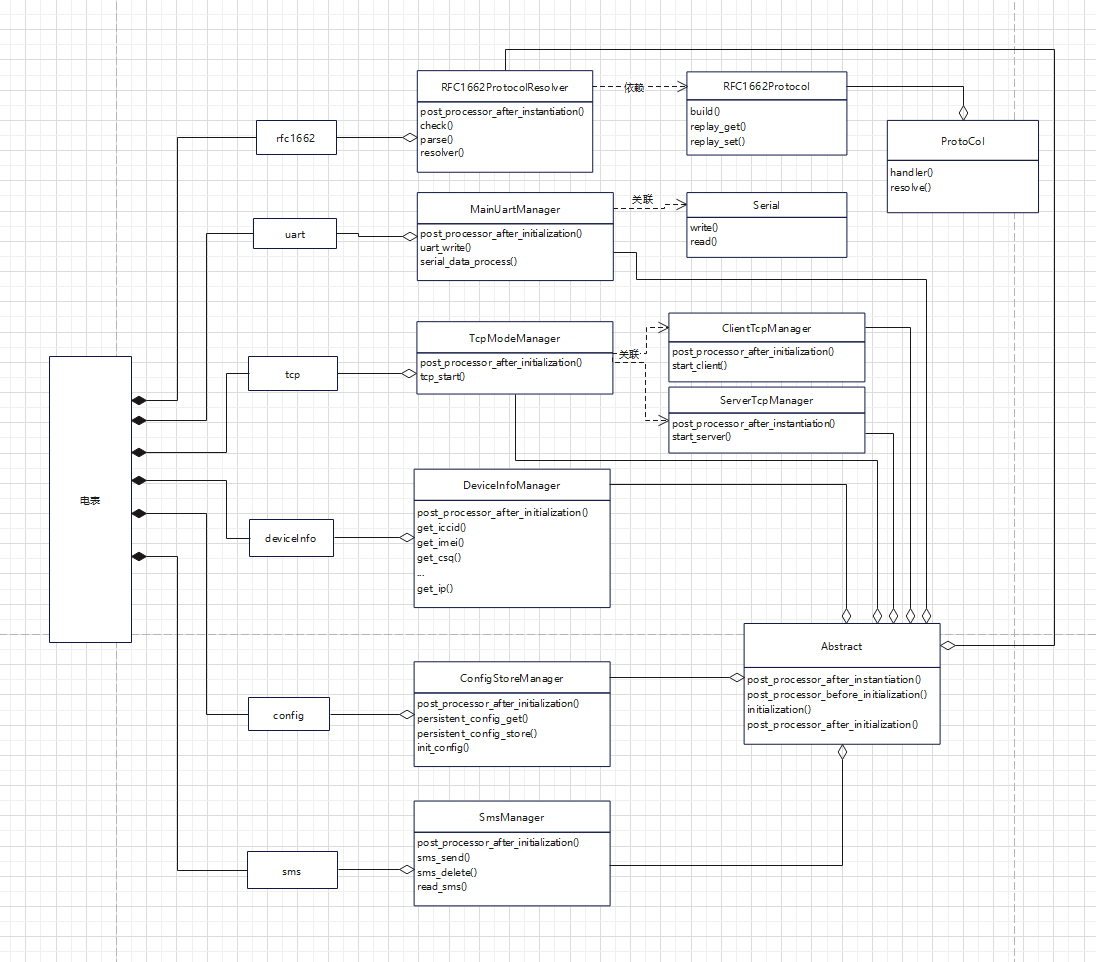
图表

描述已自动生成

1. 关键组件

组件对象关系描述:

下图用于描绘项目软件代码中各组件对象之间的依赖关系与继承关系，图中以电表作为总对象，将电表所依赖的功能对象联系到一起，核心的对象拥有哪些方法，依赖哪些功能类，具体如下图所示：



* 1. EventMesh

1. 功能描述：

EventMesh是一种动态基础中间件, 在事件驱动架构语境中,事件指的是系统中的变更、操作或观察,他们会生成通知, 然后响应到各个对事件做出响应的处理器函数中

1. 实现原理：

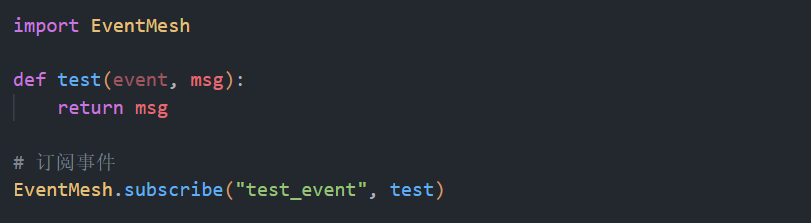
详细说明请参照EventMesh中间件设计文档，以下而为简介。

图示, 示意图

描述已自动生成

注解: 服务和驱动通过EventMesh进行数据通信, 所有的Event都是从EventStore里面过来的, 这里注意EventStore只是一种概念, 而不是真正意义上对外展示的东西

1. 订阅事件



1. 发布事件

图形用户界面, 文本, 聊天或短信

描述已自动生成

* 1. RFC1662ProtocolResolver

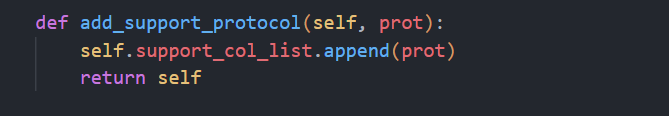
1. 功能描述：

RFC1662协议数据的解析器，用于解析处理业务中传输的RFC1662协议数据，对该类数据进行解包组包的功能。

1. 实现原理：
2. 实例化创建RFC1662协议解析器后将协议指令事件添加到List数组中

文本

描述已自动生成



1. 该类函数在初始化的时候将添加进来的协议指令事件MODE与对应的handler处理函数注册到EventMesh组件中去，如设备接收到MODE指令然后通过EventMesh去发布执行。

电脑屏幕的照片上有文字

描述已自动生成

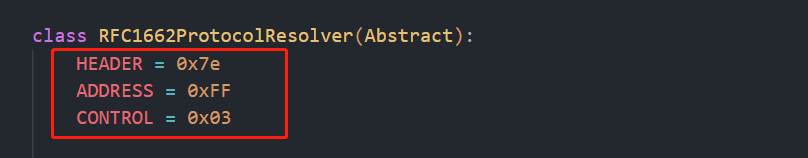
MODE指令函数示例：

文本

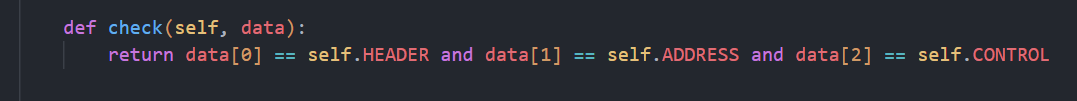
描述已自动生成

1. 收到表计发送给模组的数据后检查匹配协议帧格式

RFC1662协议帧固定格式：



判断协议帧是否满足格式：



1. 判断数据是否能解析，返回解析状态和解析到的截止下标。

文本

描述已自动生成

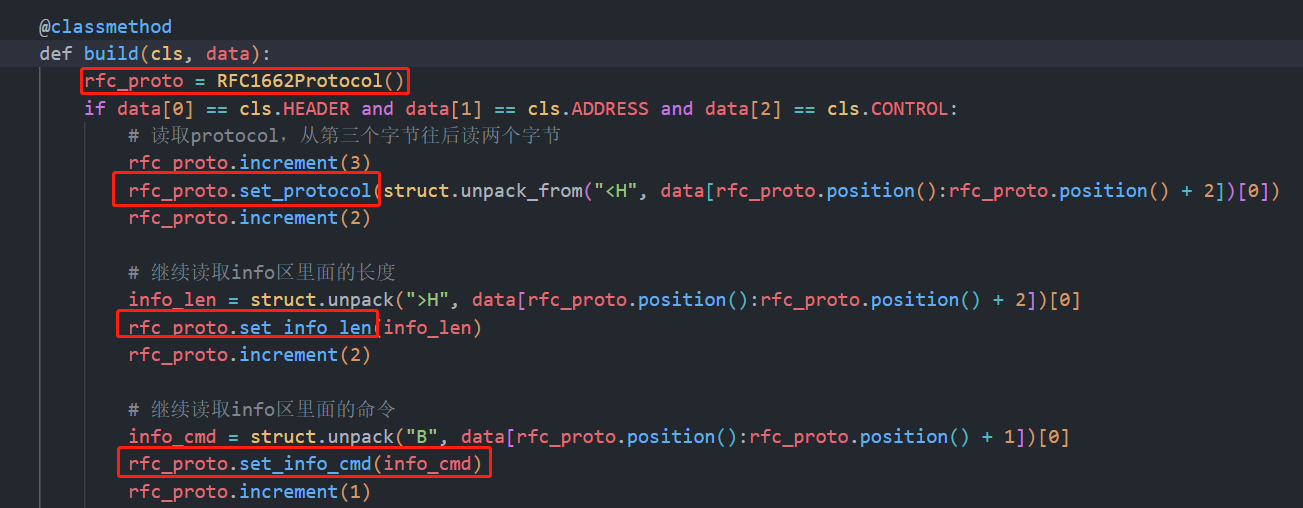
1. 数据解析是由RFC1662Protocol类方法中的build方法执行，该方法按照协议帧格式使用struct依次读取数据，并将解析出的数据分别保存在RFC1662Protocol类属性中，解析完成后将RFC1662Protocol类作为对象返回。

图一：

图形用户界面, 文本, 网站

描述已自动生成

图二：



rfs对象作为build函数的结果返回，数据解析完成后rfs对象中包含所有数据信息，在通过EventMesh去发布该条指令，找到对应的处理函数执行。

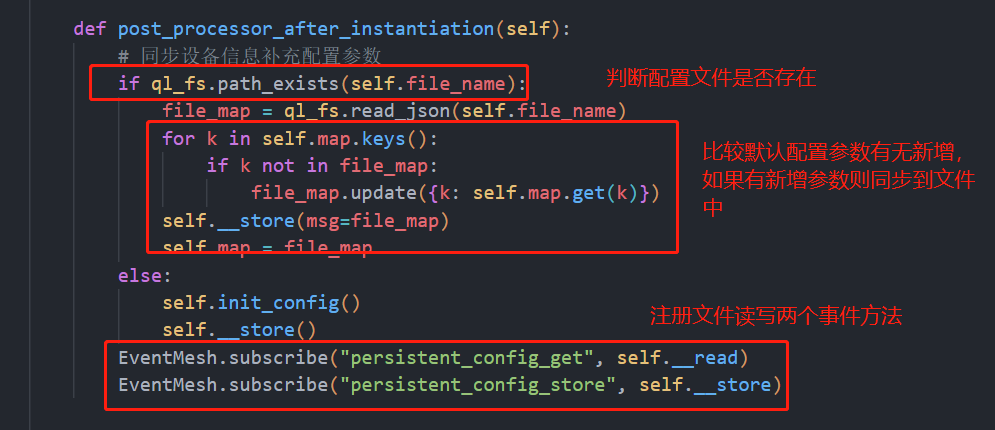
EventMesh.publish(rfs.protocol(), rfs) = EventMesh.publish(指令MODE, rfs),该指令在类函数在初始化的时候已经被注册，详见步骤2中描述。

* 1. ConfigStoreManager

1. 功能描述：

该功能主要用于处理设备参数持久化保存，提供参数读取于参数更新的方法，通过该方法与模组文件系统交互。

1. 实现原理：
2. 初始化类方法时会判断配置文件是否存在，不存在则创建一个JSON文件，文件创建成功后将默认参数写入文件中。
3. 若文件已存在，则会比对默认参数有无新增，若有新增同步更新到配置文件
4. 该类方法对外通过EventMesh注册读取和写入两个事件



* 1. NetStateManager

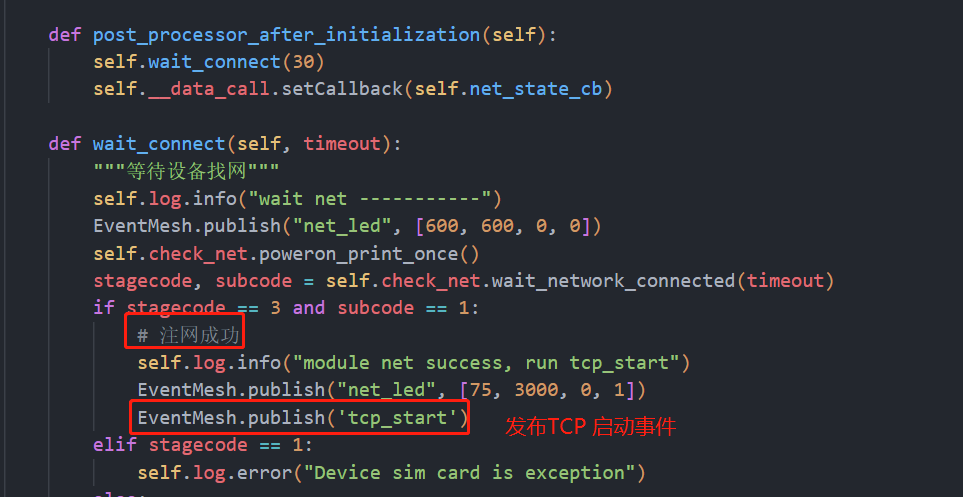
1. 功能描述：

该功能主要用于模组网络初始化以及网络状态管理。

2. 实现原理：

1. 该类方法初始化时会等待设备注网完成，且注册网络变化回调函数，通过checkNet API返回值判断设备找网状态，找网成功后会通过事件发布的方式启动TCP连接，若找网失败则会尝试重新找网。

2. checkNet API可在wiki中心查找到详细描述：<https://python.quectel.com/wiki>



* 1. TcpModeManager

1. 功能描述：

该功能用于管理设备支持的两种TCP连接模式，客户端和服务端，根据对应的模式在注网成功后启动TCP连接。

2. 实现原理：

1. 该类方法对外提供一个TCP启动的事件，在设备找网成功后通过发布该事件来启动TCP连接。

图片包含 网站

描述已自动生成

2. TCP start方法会根据配置文件中的模式来选择启动客户端还是服务端



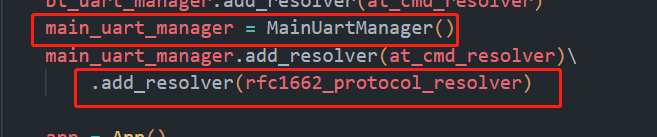
* 1. MainUartManager

1. 功能描述：

该功能主要为对设备串口的初始化，串口消息数据的收发处理。

2. 实现原理：

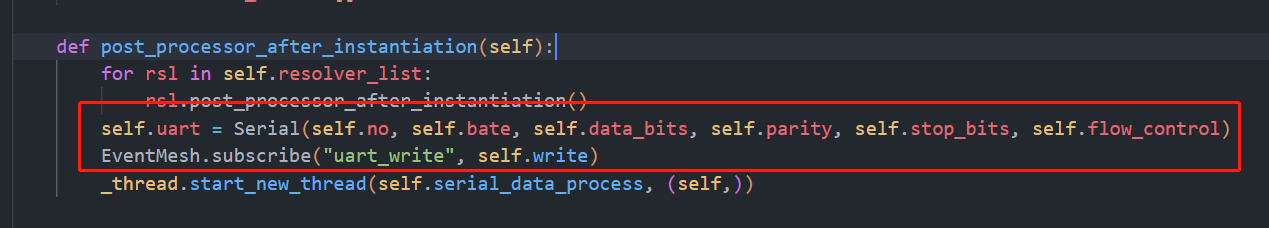
1. 因模组和表计通信是通过串口使用RFC1662协议通信，故需要给串口对象装载协议解析器用于数据解析。



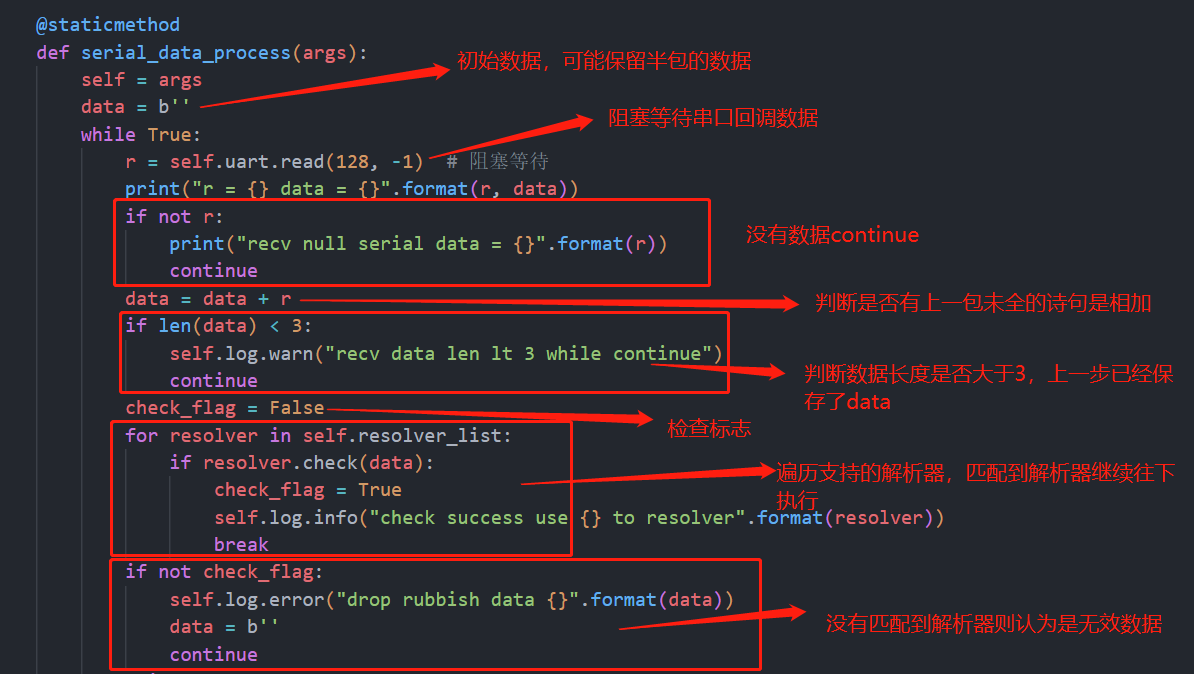
文本

中度可信度描述已自动生成

2. 类函数初始化时会创建一个串口对象，通过该对象可以对串口进行读写操作，且对外提供往该串口写数据的功能。



3. 启动一个串口数据监听的任务，当串口收到表计发来的数据后会匹配解析器来判断该条数据是否能被解析，如果能解析则通过解析器去将数据返回，若该条数据有回复则会在将结果打包好返回给表计。



日程表

低可信度描述已自动生成

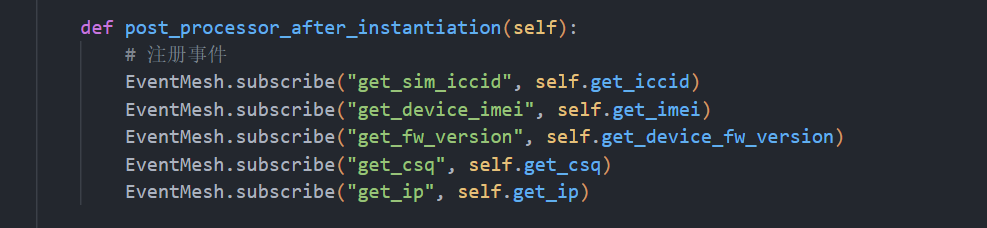
* 1. DeviceInfoManager

1. 功能描述：

该功能用于获取设备的一些基础信息，例如设备IMEI，SIM card的ICCID等。

2. 实现原理：

1. 类函数初始化时会将获取设备信息的函数注册成事件对外提供，可直接通过事件发布的方式使用。



2. 类似查询iccid

使用sim API 获取当前sim card的iccid

文本

描述已自动生成

3. 查询设备信息的API 使用详细描述可参考wiki文档：https://python.quectel.com/wiki

* 1. FtpOtaManager

1. 功能描述：

该功能用于对APP应用程序代码通过FTP远程升级更新

2. 实现原理：

1. FTP远程升级需要提供服务端地址信息以及user和password。

2. 类方法初始化时对外提供下载入口，通过事件发布的方式调用该方法。

文本

描述已自动生成

3. 与FTP服务端建立连接，连接建立成功后登陆到FTP服务器，获取到升级包路径

屏幕上的文字

描述已自动生成

4. 开始下载升级包到模组，下载完成后需要将升级包进行解压，完成解压后设置升级标志，设备重启后完成升级。

文本

描述已自动生成

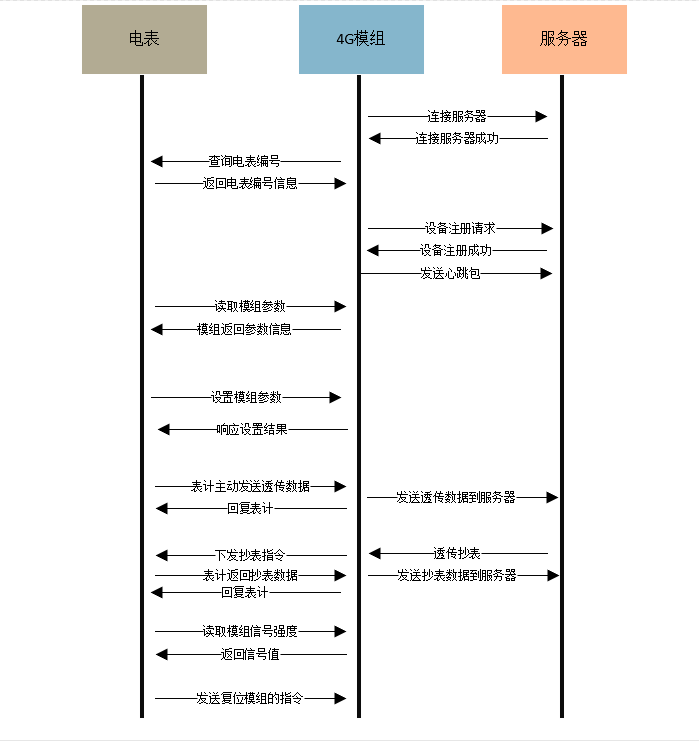
1. 系统初始化流程

图示

描述已自动生成

系统初始化流程说明：

1. 创建RFC1662解析器对象，将需要使用到该解析器的类方法都注册到解析器对象中，此处的类方法指的是我们定义的指令功能实现。
2. 创建串口对象，用于和表计通信，通过需要将解析器注册到串口对象中，因为串口会一直监听表计发送过来的数据，当有数据时使用解析器对象进行解析返回结果。
3. 所有的类方法都会约定必须有初始化前后或实例化前后要完成的事件注册或功能处理，所以我们通过一个APP类将所有的类方法在装载时和start时会将每个类方法的初始化前后动作执行完。
4. 业务流程



1. 功能示例

本章节介绍如何基于项目代码进行二次开发，例如新增功能，修改代码功能逻辑等，可以参照下面的示例帮助快速上手，熟悉代码流程。

在准备调整代码前需要先了解当前代码中功能是如何流转运行的，下面会做步骤描述：

1. 首先整个项目中实现的功能函数都采取的是使用EventMesg来进行事件驱动，通俗的讲就是我们现在实现了一个功能功函数，然后将这个函数定义一个事件名称，例如查询设备IMEI，我们将事件名称与对应的事件函数以键值对的方式做映射(”Get\_Imei”: Function),这种创建映射关系的动作理解成订阅事件。而EventMesh会保存这些订阅事件，当我们需要查询设备IMEI的时候只需要去发布Get\_Imei这个事件即可拿到返回结果。这样我们每个功能都可以独立开发，独立调试，达到解耦合的效果。
2. 新增一个功能类函数，我们先实现该功能函数，如图：



如上图中我们首先创建了一个功能类方法，这个类方法继承了一个公共基类Abstract，这个基类是用来约束开发者在功能实现上保持一致性；然后我们在类方法中重新实现了基类的post\_processor\_after\_instantiation()方法，为什么要实现该方法呢，因为我们新增的这个功能在实现完之后需要加入到APP中去注册启动，APP类会将每一个继承于Abstract类的功能类下初始化方法调用执行，也就是post\_processor\_after\_instantiation()方法，所以我们需要在该方法中将需要对外暴露的方法订阅到EventMesh中去。

Abstract基类：

文本

描述已自动生成

APP类：



3.将新增的类方法加入到main.py入口函数中：

1.导入新增的类方法：

这里的usr.msg指的是usr（用户）分区下面的mgr.py文件，因为我的类方法是是现在mgr.py中的

文本

描述已自动生成

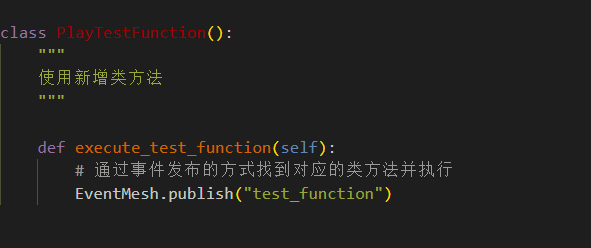
2.将类方法添加到App类中：

文本

描述已自动生成

1. 使用新增的类方法：

1.前几步中我们已经完成的类方法的新建，功能实现以及事件注册，添加到App中启动，那么接下来介绍怎么使用新增的类方法：



通过事件发布的方式找到已经订阅好的事件，执行对应的事件函数。