Shattuckite

设计文档

SHADOC-003,SDD, 第五组

版本 v0.0-SDD-5-g364f8f6

表 1： **分工说明**

小组名称	lemon	
学号	姓名	本文档中主要承担的工作内容
16231275	刘瀚骋	
16061053	孟巧岚	
16061069	许文广	
16061044	张起铭	
16061136	邓健	

表 2： **版本变更历史**

版本	提交日期	主要编制人	审核人	版本说明
v0.0-SDD-5-g364f8f6	2019-04-24 11:18	CNLHC <2463765697@qq.com>	CNLHC	提交架构设计文档首个版本
v0.0-SDD	2019-04-14 22:32	CNLHC <2463765697@qq.com>	CNLHC	提交临时版本

Contents:

1 范围

1.1 项目概述

shattuckite 项目首先作为 2019 年《软件工程》课程的课程设计，用于帮助开发团队获取这门课程的学分。

本项目旨在面向家庭及小型民用建筑物，提供一个可以通过手机及计算机远程访问环境数据及控制机电设备的计算机系统.

本系统将允许用户将若干 设备 连接至一个 嵌入式终端，并通过手机或计算机获取 信息或 控制设备行为。

嵌入式终端是一个拥有嵌入式 CPU 的硬件实体。它拥有一定的计算能力，并可以通过以太网和云端服务器进行数据交互。

典型的 设备包括物理量传感器和执行器。物理量传感器包括温度/湿度/空气质量等能够产生离散时间信号的设备，执行器包括继电器/线性导轨等能改变物理世界状态的设备。

设备到 嵌入式终端的 连接指的是设备通过有线或无线信道，通过一定的中继装置，建立与嵌入式终端的数据交互通路。有线信道包括运行 USART 协议的 RS232 总线，无线信道包括 LoRa ，Wifi。中继装置包括 Lora 网关或 Wifi 网卡。

信息至少包括

1. 传感器产生的数据
2. 执行器的状态
3. 当前系统事件

控制设备指改变配置或改变状态。改变配置指改变传感器或执行器的行为，例如数据持久化的策略/数据更新的速率/事件产生的条件等。改变状态特指改变执行器的状态。

1.2 文档概述

待处理： 插入文档概述

待处理

待处理： 插入文档概述

(原始记录 见 source/1.range/2.docintro.rstinc，第 4 行。)

待处理： 插入交叉引用

(原始记录 见 source/3.architecture/1.overall/1.soft/2.cloud.rstinc, 第 34 行。)

待处理: partial case: 传感器数据持久化时序图

(原始记录 见 source/3.architecture/1.overall/1.soft/2.cloud.rstinc, 第 36 行。)

待处理: partial case: 命令数据处理回显

(原始记录 见 source/3.architecture/1.overall/1.soft/2.cloud.rstinc, 第 38 行。)

待处理: 插入交叉引用

(原始记录 见 source/3.architecture/1.overall/1.soft/2.cloud.rstinc, 第 43 行。)

待处理: content: 加入动机交叉引用

(原始记录 见 source/3.architecture/1.overall/1.soft/2.cloud.rstinc, 第 51 行。)

待处理: partial case: 用户访问历史数据

(原始记录 见 source/3.architecture/1.overall/1.soft/2.cloud.rstinc, 第 52 行。)

待处理: partial case: 用户控制执行器

(原始记录 见 source/3.architecture/1.overall/1.soft/2.cloud.rstinc, 第 53 行。)

待处理: partial case: 用户改变传感器配置

(原始记录 见 source/3.architecture/1.overall/1.soft/2.cloud.rstinc, 第 54 行。)

待处理: partial case: 用户访问历史数据

(原始记录 见 source/3.architecture/1.overall/1.soft/3.user.rstinc, 第 36 行。)

待处理: partial case: 用户访问实时数据

(原始记录 见 source/3.architecture/1.overall/1.soft/3.user.rstinc, 第 37 行。)

待处理： partial case：用户访问控制数据

(原始记录 见 source/3.architecture/1.overall/1.soft/3.user.rstinc，第 38 行。)

待处理： 定义技术体系结构

(原始记录 见 source/3.architecture/1.overall/3.tech.rstinc，第 4 行。)

待处理： 绘制系统部署图

(原始记录 见 source/3.architecture/1.overall/4.supply.rstinc，第 7 行。)

待处理： 插入 shattuckite-rpc 交叉引用

(原始记录 见 source/4.interface/embedded/redis.rstinc，第 14 行。)

待处理： 插入 shattuckite-event 交叉引用

(原始记录 见 source/4.interface/embedded/redis.rstinc，第 28 行。)

待处理： 插入数据包定义的交叉引用

(原始记录 见 source/4.interface/embedded/redis.rstinc，第 42 行。)

待处理： 添加 fifo bibetex

(原始记录 见 source/4.interface/embedded/pipe.rstinc，第 11 行。)

待处理： 插入 IPC-Pipe 实现参考

(原始记录 见 source/4.interface/embedded/pipe.rstinc，第 21 行。)

待处理： 插入 Rest 接口文档

(原始记录 见 source/4.interface/server/rest.rstinc，第 10 行。)

待处理： 完善接口详细设计

(原始记录 见 /var/lib/jenkins/workspace/shattuckite-doc_master/architectureDesign/source/6.artifacts/index.rst，第 13 行。)

待处理： 添加运行开发环境

(原始记录 见 /var/lib/jenkins/workspace/shattuckite-doc_master/architectureDesign/source/7.enviroment/index.rst，第 4 行。)

待处理： 定义需求可追踪性

(原始记录 见 /var/lib/jenkins/workspace/shattuckite-doc_master/architectureDesign/source/8.mapping/index.rst，第 4 行。)

1.3 引用

1.4 术语及缩略词

缩写及相应全称

表 3: 缩写及相应全称

缩写	全称
Lora	Long Range
IOT	Internet Of Things
NB-IOT	Narror-Band IOT
REST	Representational State Transfer
RPC	Remote Procedure Call
RS232	Recommended Standard 232
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
M2M	Machine to Machine
C/S	Client/Server Pattern
MQTT	Message Queuing Telemetry Transport
RESTful	Representational State Transfer
MVVM	Model-View-ViewModel
QOS	Quality of service
UML	UNIFIED MODELING LANGUAGE

术语消歧

本节列出本文档中使用到，并可能出现歧义术语，对这些术语做简要的解释并列出参考资料，以期消除歧义。

Publish-Subscribe Pattern

一种软件架构模型，中文名可能是 *代理模式*或是 *订阅者-发布者模型*。

本文档中亦可能将这种模型称为 Broker Pattern

与该模型相关的术语包括 Broker, Publisher(*发布者*), Subscriber(*订阅者*) ,‘‘Topic‘‘

更多信息请参考 [BJ87]。

Pipe-filter Pattern

一种软件架构模型，中文名可能是 *管道或过滤器模型*。

与该模型相关的术语包括 Source (*数据源*), Sink(*数据汇*), filter(*数据过滤器*)

本文档中，为了兼容:ref:akka-stream 规范 Filter 与 Flow 同义

更多信息请参考 [Sha95]

Client-Server Pattern

一种软件架构模型，中文名可能是 *服务器/客户端模型*。

与该模型相关的术语 Client(*客户端*) Server(*服务器*).

更多信息请参考 [BCT04]

RESTful

一种软件架构模型，定义了一系列创建 Web 服务时的规范。

更多信息请参考 [PWA14]

Model-View-ViewModel Pattern

一种软件架构模型。可能没有中译名，国内常使用其简写 MVVM

和 MVVM` *有关的术语包括*``Data Binding , View , ViewModel, 及 Model。本文档中将使用两种具体的 Data Binding, 包括 Properiies(*属性*) 和 ‘‘ Event Callbacks(*事件回调*)‘‘

本项目将使用 MVVM 架构构建用户界面。

更多信息请参考 [And12]

UML

统一建模语言，本文档将以 UML 2.5.1 为标准。

更多信息请参考 [BCR+17]

2 需求概述

2.1 META-需求概述

本节对项目需求进行简述，列出了用例名称以及用例简述。本节的数据来自《shattuckite-需求文档》。

2.2 用例图

图 1： 系统用例图

2.3 用例简述

用例： 登陆控制程序

用户位于任何位置，使用 手机或 计算机访问控制程序。程序弹出登陆页面，并检查用户是否曾经选择保存账号或密码，如果保存，则直接使用保存的账号和密码进行鉴权；否则，提示用户输入账号和密码后进行鉴权。登陆成功后，用户应该看到程序跳转到了控制程序主页面。

用例： 概览系统情况

用户位于控制程序主页面。用户浏览主页面获取当前系统的运行状况，信息包括

1. 系统的工作状态
2. 当前未处理报警

主页拥有导航至其他页面按钮，其他页面包括

1. 监控数据
2. 历史事件
3. 管理设备
4. 操作执行器
5. 用户详情

用例：处理未处理事件

用户处理主页面中的未处理事件。用户可将事件设置为已处理，事件处理后将不再主页面中显示。控制程序记录处理人账号和处理时间。用户可以点击 历史事件按钮，查看已处理事件。

关于事件的数据类型，可参见 事件

用例：查看传感器数据

查看传感器数据操作存在两种类型：

1. 查看实时监控数据
2. 查看历史监控数据

用户可以选择需要查看的数据类型。页面拥有返回至主页面的按钮。

关于传感器数据的定义，参见 传感器数据

用例：实时监控数据

实时监控数据表现形式是折线图，用户添加传感器使传感器数据表现在折线图中，用户移除传感器使传感器数据不在折线图中，实时监控数据页面拥有返回至 [查看监控数据] 的按钮。

用例：历史监控数据

历史控数据默认表现形式是表格，表格的一组数据包含一个时间点所有传感器数据，用户可以通过操作使历史数据以折线图呈现，用户可以删除历史监控数据历史监控数据页面拥有返回至 [查看监控数据] 的按钮。

用例：查看历史事件

用户位于任何位置，使用 手机或 计算机访问控制程序并已经完成登陆进入历史事件页面。用户希望从历史事件页面获取系统的历史事件，用户希望获得的历史事件信息包括

1. 时间
2. 描述
3. 处理用户

用户希望能够删除特定历史事件。历史事件页面拥有返回至主页面的按钮。

用例：管理设备

管理设备包含的操作：

1. 管理传感器

用户可以在本页面选择操作类型管理设备页面拥有返回至主页面的按钮。

用例：添加传感器

用户希望向家庭监控系统内添加新的传感器。用户需要按照用户手册，完成传感器的硬件连接。系统应该具有自动发现新硬件的能力。用户完成传感器的硬件连接后，系统应该自动识别新增硬件，并在 管理传感器相关的 GUI 页面中列出新添加的传感器。

用例：管理传感器

页面应该列出当前连接在系统上的传感器，并具有过滤传感器的功能。用户可以选择某个传感器，对其进行进一步设置。管理传感器页面拥有返回至 管理设备的按钮。

用例：设置传感器报警逻辑

用户进行 ‘‘ 管理传感器 ‘‘操作。用户可以通过 GUI，设置传感器报警的逻辑。用户可以设置两种报警逻辑，分别是当传感器的数据：

1. 发生变化时
2. 位于某个范围时

系统自动产生一个警报事件并通知用户。用户可以结合使用场景，为不同的传感器设置不同的报警逻辑以满足不同场景的需求。例如，用户可以将一个安装在防盗门上的干簧管传感器设置为 数据发生变化``时报警，以实现监测是否有人闯入屋内；可以将安装在屋内的温度传感器设置为``位于某个范围时报警，以实现远程监控屋内的气温是否过高并据此做出是否打开空调的决策。

用例：触发报警事件

由于被监控物理量发生变化，传感器产生了某个事件，且该事件的级别大于报警级别。服务器向用户推送信息，告知用户报警事件的发生。服务器通过‘‘邮件 ‘‘(通用平台) 和‘‘事件中心 ‘‘(Android 平台) 向用户推送信息。

用例：添加执行器

用户希望向家庭监控系统内添加新的执行器。用户需要按照用户手册，完成执行器的硬件连接。系统应该具有自动发现新执行器的能力。用户完成执行器的硬件连接后，系统应该自动识别新增硬件，并在 操作执行器相关的 GUI 页面中列出新添加的执行器。

用例：操作执行器

页面应该列出系统中当前可用的执行器，并具有过滤执行器列表的功能.

用户可以选择某个特定执行器，用户选择执行器后，可以对其进行操作。用户选择相关操作，可远程控制执行器做出相应动作。

操作执行器页面拥有返回至 [概览系统情况] 的按钮。

用例：查看用户信息

用户获取并编辑当前系统的用户信息，包括

1. 获取并编辑当前账号的用户信息，信息指账号/密码，编辑指修改密码/注销账户
2. 获取当前系统中其他用户的信息，仅指账号
3. 注销登录

用户信息页面拥有返回至主页面的按钮

用例：修改账号密码

用户希望修改自己账号的密码，修改密码需要输入现有密码。修改密码成功则返回登陆界面，拥有返回至用户信息页面的按钮。

3 体系结构设计

3.1 META-体系结构设计

本节描述 shattuckite 项目的体系结构设计与设计动机。

第一部分总体结构 将分别从：

1. 硬件体系结构
2. 软件体系结构
3. 技术体系结构
4. 项目支撑结构

四个方面讨论本项目的体系结构设计； 第二部分关键问题及解决方案 将从实际问题的角度，讨论上述设计中部分设计的动机。

3.2 总体结构

软件体系结构

总览

shattuckite 软件分为三个部分：

1. 嵌入式终端软件
2. 服务端软件
3. 用户端软件

显然，此处是根据运行时环境对软件进行分类的。嵌入式终端软件运行于嵌入式处理器上；服务端软件运行于服务器上；用户终端软件运行于个人计算设备上（例如智能手机和个人电脑）。

嵌入式终端软件负责驱动边缘网络硬件，从传感器中收集数据，并作为一个发布者，将收集到的数据发布到服务器 broker；同时作为一个订阅者向 broker 订阅控制信息，实现用户远程控制设备.

服务端软件提供

1. Broker Pattern 中的 Broker 组件，用于接收其他发布者的数据并向订阅者推送；
2. 基于 Pipe-filter Pattern 设计的构件，用于订阅 Broker 中的特定 Topic 作为 Source ，数据通过一个用于处理数据持久化逻辑的 Flow，最终将数据推至由关系型数据库构成的 ‘‘Sink‘‘。
3. Client-Server 模型中的 Server ，并通过 ‘‘RESTful ‘‘ 接口，将关系型数据库中的数据暴露给用户端软件，同时处理用户与嵌入式终端的交互。

用户端软件提供用户与系统交互的人机接口.

上述讨论的三个部分，每一部分都由若干更小的构件组成。UML 组件图如下图所示.

图 2: 系统 UML 组件图

嵌入式终端软件架构

嵌入式终端软件由以下组件组成

- LORA 驱动 (shattuckite-lora-driver)
- WIFI 驱动 (shattuckite-driver-wifi)
- 本地 Redis 服务器 (redis)
- MQTT 协议代理 (shattuckite-mqtt-broker)
- 数据管道 (shattuckite-data-pipe)
- RESTful 服务器 (shattuckite-restful)
- 推送服务器 (shattuckite-server-push)
- 关系型数据库 (mysql)
- 视图模型 (shattuckite-gui-viewmodel)
- 浏览器视图 (shattuckite-gui-browserview)
- 移动端视图 (shattuckite-gui-nativeview)
- 数据源 (dataSource)
- 数据分发处理 (dataHandle)
- 订阅消息中转 (subBridge)
- 事件处理 (evnetHandle)
- RPC 执行器 (shattuckiteRPC)

该部分 UML 组件图如下图所示

图 3: 嵌入式终端 UML 组件图

设备驱动将与硬件（例如 Lora 网关或 Wifi 网卡）通信，并把传感器数据和执行器的状态映射为 Linux 文件。

rpc 执行器可通过文件读操作获取传感器数据与执行器的状态；终端核心可通过写操作设置执行器的状态。关于通过文件 IPC 的更多详情请参考系统接口说明

dataSource 将通过由驱动提供的 IPC 接口获取传感器数据和执行器状态，并将他们 Publish 到 redis 提供的数据 Pub 接口中.

服务器端软件架构

服务端软件由以下组件组成

- LORA 驱动（shattuckite-lora-driver）
- WIFI 驱动（shattuckite-driver-wifi）
- 本地 Redis 服务器（redis）
- MQTT 协议代理（shattuckite-mqtt-broker）
- 数据管道（shattuckite-data-pipe）
- RESTful 服务器（shattuckite-restful）
- 推送服务器（shattuckite-server-push）
- 关系型数据库（mysql）
- 视图模型（shattuckite-gui-viewmodel）
- 浏览器视图（shattuckite-gui-browserview）
- 移动端视图（shattuckite-gui-nativeview）
- 数据源（dataSource）
- 数据分发处理（dataHandle）
- 订阅消息中转（subBridge）
- 事件处理（evnetHandle）
- RPC 执行器（shattuckiteRPC）

该部分 UML 组件图如下图所示

图 4: 服务端 UML 组件图

本项目使用 MQTT 协议完成 M2M 端到端通信。

shattuckite-mqtt-broker 构件负责提供 Broker。为了实现业务需求，服务端将同时维护两个 Broker。其中一个 ‘Broker‘负责处理 嵌入式终端和 服务器端之间的通信 Topic，称之为 Broker1；另一个负责处理 用户端和 服务器端之间的 Topic 以实现服务器推送的功能，称之为 Broker2 。

shattuckite-data-pipe 负责提供将 Subscriber 抽象为 Source 的中间件，并完成若干 Flow 和 Sink 以实现业务逻辑。本构件中实现三种 Flow 。分别用于

1. 从数据流中筛选出传感器数据
2. 从数据流中筛选出命令回显数据
3. 从传感器数据中筛选出需要被持久化的数据

此外本构件中实现两种 Sink，分别用于

1. 将数据持久化到关系型数据库
2. 作为 Broker2 的发布者，将命令回显数据发布到相关 Topic 以实现 Server Push

关于数据流的扇入，扇出，多路复用等细节，将在数据管道一节中详细讨论.

待处理: 插入交叉引用

待处理: partial case: 传感器数据持久化时序图

待处理: partial case: 命令数据处理回显

---

shattuckite-RESTful 主要提供 用户端与 服务器端交互的接口。接口工作在 Http(s) 上。本构件将提供 Immutable(不可变) 和 mutable(可变) 两种类型的接口。关于更多细节请参考 RESTful 服务器一节。

待处理: 插入交叉引用

当用户端调用不可变接口时（例如查询传感器历史数据），该构件将通过持有的数据库连接器，执行相应的查询指令从关系型数据库中获得数据，并将查询结果序列化后返回给客户端。

当用户端调用可变接口时，该构件将非阻塞的，通过 Http 协议返回响应。上述响应并不包含用户端真正想要获得的信息，它类似于异步编程模型中 Future 对象，我们称其为 partial response。partial response 中包含了关于如何获得真正响应的 metainfo(元信息)，用户将根据元信息中的内容，订阅 Broker2 中特定的 Topic 并最终从消息队列获取真正的回显。

待处理: content: 加入动机交叉引用

待处理: partial case: 用户访问历史数据

待处理: partial case: 用户控制执行器

待处理: partial case: 用户改变传感器配置

关于服务端关系型数据库的设计，请参考数据库设计 一节，此处不过多赘述。

用户端软件架构

用户端软件由以下构件组成

- LORA 驱动 (shattuckite-lora-driver)
- WIFI 驱动 (shattuckite-driver-wifi)
- 本地 Redis 服务器 (redis)
- MQTT 协议代理 (shattuckite-mqtt-broker)
- 数据管道 (shattuckite-data-pipe)
- RESTful 服务器 (shattuckite-restful)
- 推送服务器 (shattuckite-server-push)
- 关系型数据库 (mysql)
- 视图模型 (shattuckite-gui-viewmodel)
- 浏览器视图 (shattuckite-gui-browserview)
- 移动端视图 (shattuckite-gui-nativeview)
- 数据源 (dataSource)
- 数据分发处理 (dataHandle)
- 订阅消息中转 (subBridge)
- 事件处理 (evnetHandle)
- RPC 执行器 (shattuckiteRPC)

图 5: 用户端 UML 组件图

本项目使用 MVVM 架构模式实现人机交互接口。

MVVM 中的 model 由两部分组成:

1. 服务端中的关系型数据库

2. 嵌入式硬件的状态

视图模型组件作为 MVVM 架构中的 Viewmodel 角色。该组件应该能够从 Model 获取数据并通过 Data-Binding 处理与 View 的交互。

当 ViewModel 需要对关系型数据库中的数据进行操作时，将通过传统意义上的，基于 Http(s) 协议的 RESTful 接口；当 ViewModel 需要直接对嵌入式硬件的状态进行操作或是获取终端实时数据时，需要借助一个消息队列，以 Broker Pattern 的方式异步的获取数据，

浏览器视图和移动端视图作为 MVVM 架构中的 View 角色，将通过两种 Data-Binding 与 View-Model 交互数据，包括

1. ViewModel 在内部维护一个状态树，并通过 Properties 的形式，将数据提供给 View。
2. View 通过 Event-Callback 更新 ViewModel 中的状态。

总之，界面由状态树唯一决定，页面发生变化的唯一方法是更新状态树。

待处理: partial case: 用户访问历史数据

待处理: partial case: 用户访问实时数据

待处理: partial case: 用户访问控制数据

硬件体系结构

系统要正常工作，应该包含以下硬件设施

1. 终端设备

包括传感器和执行器。例如温度传感器和继电器。

2. 边缘网络网关

用于支撑嵌入式终端和终端设备进行交互的硬件。例如 LORA 发射器或 WIFI 节点.

3. 嵌入式终端

负责与服务器交互，实现数据上行聚合以及远程状态控制。一般是一个以嵌入式芯片为计算核心，并附加有内存/磁盘等外围设备的小型计算机系统。本项目将优先支持 CPU 架构为 ARM Cortex A9 的嵌入式终端。

4. 服务器集群

负责处理核心业务逻辑。服务器集群是 N 台 (N≥1) 位于专业 IDC 机房的计算实例组成。

5. 客户端

负责处理用户与系统的交互。一般是某种个人计算设备，例如手机或是个人 PC。

技术体系结构

待处理： 定义技术体系结构

支撑体系结构

系统部署方案

待处理： 绘制系统部署图

3.3 关键问题及解决方案

数据上行聚合

本项目将会有 **大量**嵌入式终端，将 **小规模**的数据持续聚合到服务器端。

在上述场景下，使用 Http 协议进行数据聚合将面临以下问题：

1. 处理高并发请求

常见的 Http 服务器采用多线程模型来处理入站请求。对于常规的 Web 应用，一次访问常常伴随着执行 SQL 语句，渲染前端页面等一系列操作，为每一次请求分配一个线程进行处理是合情合理的。但是物联网项目中，每次请求要求的计算资源极小，而请求的频率很高；当并发数量上升时，线程上下文切换将浪费系统资源，导致性能下降

2. 过多的冗余信息

一个合法的 Http 协议请求，将一系列 Key-Value 以字符串的形式封装在包中作为请求 Header。本项目中一次请求的数据量极小，可能会出现 Header 中 Value 数据占用的空间还没有 Key 占用的空间大这样的情况。使用 Http 协议传输数据会将大量的带宽浪费在传输冗余的数据上。

为了避免上述问题发生，本项目使用 MQTT 协议进行数据上行聚合。MQTT 能较好的解决上述两个问题

1. 常见的 MQTT Broker 并不通过多线程，而是通过异步的方式处理并发请求，因此能避免线程上下文切换带来的额外开销。
2. MQTT 协议的数据包是以二进制形式存储数据的，可以最大化带宽利用率。

服务器推送

本项目在两个场景下需要引入服务器推送的支持。

1. 服务端向用户端的推送

某些情况下（例如用例：触发报警事件），服务端会将数据主动推送到用户端。

2. 服务端向嵌入式终端的推送

某些情况下（例如用例：操作执行器，用例：设置传感器报警逻辑），服务端会将数据主动推送到嵌入式终端。

Http 协议作为典型的 C/S 架构协议，无力处理这种客户端不发送请求但需要获取数据的情形。此外 NAT 技术的广泛使用也让服务端基本不可能获得客户端的真实地址。为了实现服务端推送，一种常见的做法是摒弃高层协议，转而使用最基本的 TCP Socket 来维护一个长连接，以实现服务器推送。

这种通过长连接，使用自定义协议实现服务器推送的方法，最大的缺点在于它会极大的增加代码的复杂性。在实现时需要考虑断线重连，虚连接等等一系列非常底层的问题。当我们的服务对于 QOS 有要求时，复杂度会进一步上升。

在 Broker Pattern 中没有 C/S 架构中“请求-回复”的模式，取而代之的是“发布-订阅”的模式。这种模式最显著的特点是

1. 任何节点都可以是逻辑上数据的产生者，也可以逻辑上数据的获取者
2. 数据的产生者和数据的消费者不必知道对方的存在

综上两点原因，我们在设计阶段引入 Broker Pattern 来实现服务器推送。在数据上行聚合 中也提到了我们将会用 MQTT 协议实现数据的聚合，而 MQTT 协议天生就带有 Broker Pattern 的属性。

控制嵌入式终端状态

本项目允许用户通过某些接口更改嵌入式终端的状态（例如用例：操作执行器，用例：设置传感器报警逻辑）。实现这项功能的难点分别为

1. 用户端与嵌入式端并不总是能够（事实上，几乎总不能够）相互访问，必须要通过服务端中转。
2. 更改嵌入式终端的状态是一项非常耗时的操作。即便是拨动一个继电器，硬件也需要至少 100 毫秒的动作时间。100 毫秒的等待时间对于服务器和客户端都是非常高昂的。

为了解决上述两个问题，引入了 Defered RESTful 的概念，向传统的 RESTful 服务器中引入消息队列。

4 接口设计

4.1 人机交互接口

图 6: 页面-alterpassword

图 7: 页面-alterpassword_error

图 8: 页面-historyevent

图 9: 页面-historyevent_detail


返回
主页
系统运行状态：
开 启
待处理报警：
报警名称
时间
报警名称
时间
报警名称
时间
监控数据
历史事件
设备管理
设备操控
用户信息

图 10： 页面-home

图 11： 页面-login

图 12: 页面-login_passworderror

图 13： 页面-login_servererror

图 14： 页面-manage

图 15： 页面-managedevice

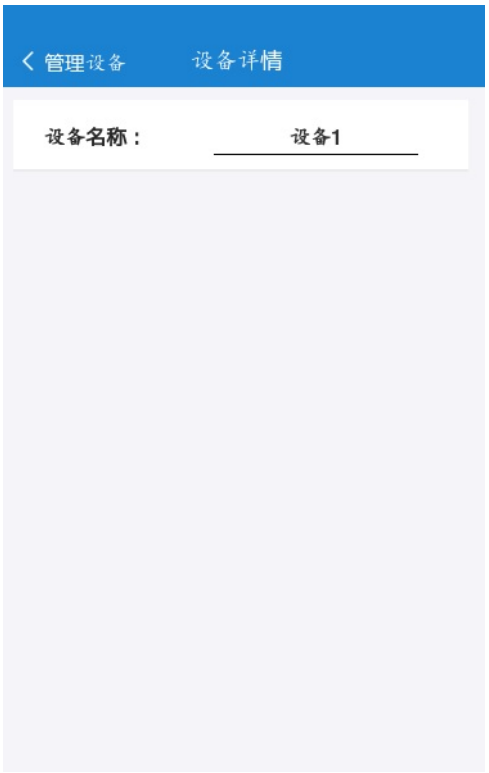

图 16： 页面-managedevice_detail

图 17: 页面-manageevent

图 18: 页面-manageevent_detail

图 19: 页面-monitordata

图 20: 页面-monitordata_history

图 21： 页面-monitordata_present

图 22： 页面-operate

图 23： 页面-operatedevice_detail

图 24： 页面-todo

图 25: 页面-userinfo

4.2 系统接口说明

系统接口目录

本节列出系统的接口目录

嵌入式端接口

- (1)-IPC-Pipe2
- (2)-IPC-Pipe1
- (4)-Pub
- (5)-Sub
- (6)-Socket
- (7)-Http
- (8)-Websocket
- (12)-Props
- (15)-数据 Pub
- (16)-数据 Sub
- (17)-命令 Pub
- (18)-命令 Sub

- (19)-事件 Pub
- (20)-事件 Sub

服务端接口

- (1)-IPC-Pipe2
- (2)-IPC-Pipe1
- (4)-Pub
- (5)-Sub
- (6)-Socket
- (7)-Http
- (8)-Websocket
- (12)-Props
- (15)-数据 Pub
- (16)-数据 Sub
- (17)-命令 Pub
- (18)-命令 Sub
- (19)-事件 Pub
- (20)-事件 Sub

客户端接口

- (1)-IPC-Pipe2
- (2)-IPC-Pipe1
- (4)-Pub
- (5)-Sub
- (6)-Socket
- (7)-Http
- (8)-Websocket
- (12)-Props
- (15)-数据 Pub
- (16)-数据 Sub
- (17)-命令 Pub
- (18)-命令 Sub
- (19)-事件 Pub
- (20)-事件 Sub

嵌入式终端接口设计

命令相关接口

接口名称 命令 Pub

接口 ID 17

组件名称 redis

被以下组件调用

- subBridge
- evnetHandle

接口名称 命令 Sub

接口 ID 18

组件名称 redis

被以下组件调用

- shattuckiteRPC

上述两个接口提供中转命令功能。

这两个接口使用 Redis Message 传输字符串。字符串使用 json 格式进行序列化.

关于命令的格式请参考

待处理: 插入 shattuckite-rpc 交叉引用

事件相关接口

接口名称 事件 Pub

接口 ID 19

组件名称 redis

被以下组件调用

- dataHandle

接口名称 事件 Sub

接口 ID 20

组件名称 redis

被以下组件调用

- evnetHandle

上述两个接口提供中专事件功能。

这两个接口使用 Redis Message 传输字符串。字符串使用 json 格式进行序列化.

关于事件数据的格式请参考

待处理: 插入 shattuckite-event 交叉引用

数据相关接口

接口名称 数据 Pub

接口 ID 15

组件名称 redis

被以下组件调用

- dataSource
- shattuckiteRPC

接口名称 数据 Sub

接口 ID 16

组件名称 redis

被以下组件调用

- dataHandle

上述两个接口提供中转数据功能。

这两个接口使用 Redis Message 传输字符串。字符串使用 json 格式进行序列化.

关于系统数据的格式请参考

待处理: 插入数据包定义的交叉引用

命名管道 IPC 接口

接口名称 IPC-Pipe2

接口 ID 1

组件名称 shattuckite-lora-driver

被以下组件调用

- shattuckiteRPC

接口名称 IPC-Pipe1

接口 ID 2

组件名称 shattuckite-driver-wifi

被以下组件调用

- shattuckiteRPC

上述两个接口由驱动程序提供，用于向其他组件提供访问硬件的支持。

待处理： 添加 fifo bibetex

接口使用由 Linux 系统提供的 fifo 进行数据通信。每一个 IPC-Pipe 接口会建立两个命名管道，其中一个管道用于驱动下发硬件数据，另一个管道用于驱动接收操作硬件的指令.

命名管道传输二进制数据流。信道使用行标记进行帧同步，数据包之间用字节 \r\n 分割。数据包使用简单的键值对进行序列化。

关于驱动 IPC-Pipe 接口的更多详细信息，请参考

待处理： 插入 IPC-Pipe 实现参考

服务端接口设计

mqtt 代理接口

接口名称 Pub

接口 ID 4

组件名称 shattuckite-mqtt-broker

被以下组件调用

- shattuckite-restful
- dataHandle

接口名称 Sub

接口 ID 5

组件名称 shattuckite-mqtt-broker

被以下组件调用

- shattuckite-data-pipe
- shattuckite-server-push
- subBridge

上述两个接口实现 mqtt 数据中转。

接口通过 mqtt 协议传输二进制数据流。数据载荷使用 json 格式进行序列化.

具体的数据包格式可参考数据相关接口

Rest 服务接口

接口名称 Http

接口 ID 7

组件名称 shattuckite-restful

被以下组件调用

- shattuckite-gui-viewmodel

上述接口暴露给 shattuckite-gui-viewmodel 。

接口使用 Http 协议进行通信. 数据载荷使用 json 格式序列化。

待处理: 插入 Rest 接口文档

推送服务器接口

接口名称 Websocket

接口 ID 8

组件名称 shattuckite-server-push

被以下组件调用

- shattuckite-gui-viewmodel

上述接口暴露给 shattuckite-gui-viewmodel , 用于服务器主动向客户端推送数据。

接口使用 Websocket 协议进行通信, 传输二进制数据流, 数据载荷使用 json 格式序列化。

更多详细信息请参考接口组件。

数据库接口

接口名称 Socket

接口 ID 6

组件名称 mysql

被以下组件调用

- shattuckite-data-pipe

- shattuckite-restful

数据库接口由 Mysql 组件提供。通过该接口实现对数据库的数据的操作。

shattuckite-data-pipe 通过 alpakka 提供的 Slick 组件与数据库连接并执行 SQL 语句。

shattuckite-restful 通过 django-orm 提供的组件与数据库连接并执行 SQL 语句。

客户端接口设计

视图模型接口

接口名称 Props

接口 ID 12

组件名称 shattuckite-gui-viewmodel

被以下组件调用

- shattuckite-gui-browserview
- shattuckite-gui-nativeview

5 数据库设计

5.1 META-数据库设计

本节主要阐述项目中关系型数据库结构的设计。

5.2 数据表设计

数据表: 传感器数据 (SensorData)

(ef8da) 基本信息

表 4: 数据表:SensorData 字段信息

字段名	字段类型	字段描述
dataIndex	Integer	数据表主键
Sensor	Integer	指向传感器数据表的外键
value	Integer	数据
time	Datetime	数据产生的时间

(ef8da) 备注

该数据表用于存储传感器产生的数据。在逻辑上，传感器产生的数据与传感器实体拥有多对一的关系，即一个传感器可以产生若干组数据，故该数据表持有指向传感器数据表的外键。

数据表： 事件 (Event)

(4e1f4) 基本信息

表 5： 数据表:Event 字段信息

字段名	字段类型	字段描述
eventId	Integer	传感器事件数据表主键
Sensor	Integer	指向传感器数据表的外键
Actuator	Integer	指向执行器数据表的外键
type	String	事件类型。
level	Integer	事件等级。数字表示的事件优先级，数字越小表示优先级越高。
time	Datetime	事件产生的时间
verbose	String	事件描述。用自然语言简述事件背景，由 DSL 确定。

(4e1f4) 备注

该数据表用于存储系统运行过程中产生的事件。

数据表： 传感器 (Sensor)

(162cc) 基本信息

表 6： 数据表:Sensor 字段信息

字段名	字段类型	字段描述
sensorID	Integer	传感器数据表主键
House	Integer	指向房屋数据表的外键
updateRate	Integer	传感器更新速率，以 Hz 为单位
EventTriggerDSL	string	简短的 DSL 程序，描述传感器产生事件的逻辑
PersistenceStrategyDSL	string	简短的 DSL 程序，描述传感器数据持久化逻辑

(162cc) 备注

该数据表用于存储传感器的相关数据，其中每一条记录和一个物理上存在的传感器实体一一对应。

数据表： 房屋 (House)

(2526c) 基本信息

表 7： 数据表:House 字段信息

字段名	字段类型	字段描述
HouseId	Integer	房屋数据表主键
name	String	房屋名称
deviceType	DeviceType <: String	描述房屋终端嵌入式设备的类型

(2526c) 备注

该数据表用于在逻辑上表示由一个嵌入式终端及附加设备所覆盖的物理区域。

数据表： 执行器 (Actuator)

(7ceb7) 基本信息

表 8： 数据表:Actuator 字段信息

字段名	字段类型	字段描述
ActuatorId	Integer	执行器数据表主键
House	Integer	指向房屋数据表的外键
type	ActuatorType <: String	执行器类型
value	String	执行器当前状态
EventTrig-gerDSL	String	简短的 DSL 程序，描述执行器产生事件的逻辑

(7ceb7) 备注

该数据表用于记录执行器实体的相关信息。

数据表： 用户 (User)

(b512d) 基本信息

表 9: 数据表:User 字段信息

字段名	字段类型	字段描述
UID	Integer	用户主键
House	Integer	指向房屋数据表的外键
Permission	Integer	指向房屋数据表的外键
username	String	用户名
password	String	用户密码哈希值

(b512d) 备注

记录系统用户及相关信息。

数据表: 权限 (Permission)

(229ef) 基本信息

表 10: 数据表:Permission 字段信息

字段名	字段类型	字段描述
permissionID	Integer	权限数据表主键
code	String	权限识别码
verbose	String	权限描述

(229ef) 备注

存储预定义的值，表示用户的操作权限。

5.3 ER 图建模

图 26: 关系型数据库 ER 图

6 详细设计

6.1 META-详细设计

本节主要阐述系统组件的详细设计。

待处理： 完善接口详细设计

6.2 嵌入式端组件详细设计

嵌入式终端 Redis

设计

该组件基本信息如下：

组件名称 本地 Redis 服务器

组件代号 redis

提供接口

- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

本系统通过使用 Redis 提供的订阅发布功能来实现嵌入式终端内其他组件的 IPC。在部署时，使用 Redis Server 为数据，事件及命令分别建立 Topic，程序中其他组件可以通过订阅 Topic 或向 Topic 中推送消息来实现相互通信。通过引入这种 IPC 模式，可以

1. 降低其他组件间的耦合性

以数据分发处理组件为例说明 Redis 是如何降低模块间的耦合性的。该组件用于处理嵌入式终端的数据，将数据推送到远程 Broker；在数据满足某些条件时，生成系统事件并将系统事件推送到远程服务器及本地的事件处理组件。如果没有 Redis，该模块至少需要三套接口，分别处理与数据源 MQTT 协议代理和事件处理之间的通信，这使得数据分发处理和这三个组件紧密的耦合在了一起，提高了维护的复杂程度。引入 Redis 后，显而易见的，该模块与事件处理和数据源的耦合不再存在了，它们三者彼此无法感知到对方的存在；具体实现时代码的复杂程度也得以降低，因为现在只需要处理和 Redis 之间的通信即可。

2. 提高组件内聚性

以 RPC 执行器为例说明 Redis 是如何提高组件内聚性的。该组件会执行命令。引入 Redis 后，获取命令这一操作简化为监听 Redis 服务器 Topic. 因此实现该组件的过程中无需过多考虑从哪里以及以何种方式获得待执行的命令，而专注于如何实现 执行命令这一核心逻辑。

3. 提高系统的扩展性

接上述关于 RPC 执行器的例子。事实上在当前的系统设计中，需要执行的命令主要来源于远程服务器和事件处理器。假如后期系统需要在本地接入一些交互组件（例如键盘）来实现用户对嵌入式终端的直接控制，那么我们无需对现有的组件进行任何改动，只需要新开发一个叫做 本地交互组件驱动的新组件，该组件只需要将用户的操作映射为预先规定的命令格式，并发布到由 Redis 提供的命令 Topic 中，就完成了系统的扩展。

4. 简化 IPC 实现

在没有 Redis 的情况下，组件间的通信可能会通过命名管道，本地 Socket 或是共享内存实现。尽管它们的性能有可能高于 Redis，但是使用它们将会带来极大的不便。命名管道对通信双方的读写时序有严格的要求；本地 Socket 只能传输字节流，为了实现通信需要引入额外的序列化与反序列化逻辑，此外还需要考虑 TCP 连接稳定性的问题；使用共享内存则需要通过同步原语构造复杂的逻辑来保证数据访问不会冲突。引入 Redis 将大大的简化 IPC 的实现。Redis 本身支持丰富的数据类型，因此可以不用考虑序列化/反序列化的问题；订阅发布的模式则让实现无需考虑同步的问题。

数据源

该组件基本信息如下：

组件名称 数据源

组件代号 dataSource

提供接口

- 无

依赖接口

- IPC-Pipe2
- IPC-Pipe1

- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

数据源负责调用各种硬件驱动，包括 shattuckite-lora-driver 和 shattuckite-driver-wifi 来收集终端传感器数据和执行器状态，并订阅 redis 提供的数据 Pub ，将系统数据发布到本地 Redis 的数据 Topic 中。

Lora 驱动

该组件基本信息如下：

组件名称 LORA 驱动

组件代号 shattuckite-lora-driver

提供接口

- IPC-Pipe2

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub

- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

该组件用于将 Lora 节点的值映射为命名管道文件。

Wifi 驱动

该组件基本信息如下：

组件名称 WIFI 驱动

组件代号 shattuckite-driver-wifi

提供接口

- IPC-Pipe1

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

该组件用于将 wifi 节点的值映射为命名管道文件。

数据分发处理

该组件基本信息如下：

组件名称 数据分发处理

组件代号 dataHandle

提供接口

- 无

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

该组件从数据 Sub 订阅系统中的数据，并完成两项工作

1. 将系统数据发布到 Pub
2. 根据系统数据生成系统事件，并将事件数据发布到 Pub 和事件 Pub

参见数据 Sub ，该组件将从 Redis 服务器中，获得按照 json 格式序列化的数据。将一个数据包反序列化后，得到的数据包结构大概如下

```
{
type:"actuator"


}
```

数据分发处理

该组件基本信息如下：

组件名称 订阅消息中转

组件代号 subBridge

提供接口

- 无

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

asddas

事件处理

该组件基本信息如下：

组件名称 事件处理

组件代号 evnetHandle

提供接口

- 无

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http

- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

asddas

shattuckiteRPC

该组件基本信息如下：

组件名称 RPC 执行器

组件代号 shattuckiteRPC

提供接口

- 无

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

6.3 服务端组件详细设计

MQTT 协议代理

该组件基本信息如下：

组件名称 MQTT 协议代理

组件代号 shattuckite-mqtt-broker

提供接口

- Pub
- Sub

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

关系型数据库

该组件基本信息如下：

组件名称 关系型数据库

组件代号 mysql

提供接口

- Socket

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

数据管道

该组件基本信息如下：

组件名称 数据管道

组件代号 shattuckite-data-pipe

提供接口

- 无

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub

- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

推送服务器

该组件基本信息如下：

组件名称 推送服务器

组件代号 shattuckite-server-push

提供接口

- Websocket

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

RESTful 服务器

该组件基本信息如下：

组件名称 RESTful 服务器

组件代号 shattuckite-restful

提供接口

- Http

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

6.4 客户端组件

视图模型

该组件基本信息如下：

组件名称 视图模型

组件代号 shattuckite-gui-viewmodel

提供接口

- Props

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket

- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

浏览器视图

该组件基本信息如下：

组件名称 浏览器视图

组件代号 shattuckite-gui-browserview

提供接口

- 无

依赖接口

- IPC-Pipe2
- IPC-Pipe1
- Pub
- Sub
- Socket
- Http
- Websocket
- Props
- 数据 Pub
- 数据 Sub
- 命令 Pub
- 命令 Sub
- 事件 Pub
- 事件 Sub

asddas

移动端视图

该组件基本信息如下：

组件名称 移动端视图

组件代号 `shattuckite-gui-nativeview`

提供接口

- 无

依赖接口

- `IPC-Pipe2`
- `IPC-Pipe1`
- `Pub`
- `Sub`
- `Socket`
- `Http`
- `Websocket`
- `Props`
- 数据 `Pub`
- 数据 `Sub`
- 命令 `Pub`
- 命令 `Sub`
- 事件 `Pub`
- 事件 `Sub`

7 运行与开发环境

待处理： 添加运行开发环境

---

8 需求可追踪性说明

待处理： 定义需求可追踪性

References

[And12] Chris Anderson. The Model-View-ViewModel (MVVM) Design Pattern. Apress, Berkeley, CA, 2012. ISBN 978-1-4302-3501-9. URL: https://doi.org/10.1007/978-1-4302-3501-9_13, doi:10.1007/978-1-4302-3501-9_13.

[BCT04] B. Benatallah, F. Casati, and F. Toumani. Web service conversation modeling: a cornerstone for e-business automation. IEEE Internet Computing, 8(1):46–54, Jan 2004. doi:10.1109/MIC.2004.1260703.

[BJ87] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. SIGOPS Oper. Syst. Rev., 21(5):123–138, November 1987. URL: http://doi.acm.org/10.1145/37499.37515, doi:10.1145/37499.37515.

[BCR+17] Conrad Bock, Steve Cook, Pete Rivett, Tom Rutt, Ed Seidewitz, Bran Selic, and Doug Tolbert. Unified modeling language specification version 2.5.1. Technical Report, Object Management Group, 2017. URL: https://www.omg.org/spec/UML/2.5.1.

[HL19] Hancheng Liu. Shattuckite-需求文档. 2019. URL: https://github.com/sebuaa2019/Team105.

[PWA14] Cesare Pautasso, Erik Wilde, and Rosa Alarcon, editors. REST: Advanced Research Topics and Practical Applications. Springer New York, 2014. URL: https://doi.org/10.1007/978-1-4614-9299-3, doi:10.1007/978-1-4614-9299-3.

[Sha95] Mary Shaw. Patterns for Software Architectures. Addison-Wesley, Carnegie Mellon University, 1995.