Shattuckite

设计文档

SHADOC-003,SDP, 第五组

版本

表 1: **版本变更历史**

版本	提交日期	主要编制人	审核人	版本说明
	2019-04-15 15:33	CNLHC <2463765697@qq.com>	CNLHC	提交临时版本

Contents:

1 范围 **3**
- 1.1 项目概述 . . . 3
- 1.2 引用 . . . 4
- 1.3 术语及缩略词 . . . 4

2 需求概述 **6**
- 2.1 META-需求概述 . . . 6
- 2.2 用例图 . . . 6
- 2.3 用例简述 . . . 6

3 体系结构设计 **9**
- 3.1 META-体系结构设计 . . . 9
- 3.2 总体结构 . . . 9
- 3.3 关键问题及解决方案 . . . 13

4 接口设计 **15**
- 4.1 人机交互接口 . . . 15
- 4.2 系统接口说明 . . . 15

5 数据库设计 **15**
- 5.1 META-数据库设计 . . . 15
- 5.2 数据表设计 . . . 15
- 5.3 ER 图建模 . . . 15

6 详细设计 **15**
- 6.1 META-详细设计 . . . 15

7 运行与开发环境 **15**

8 需求可追踪性说明 **15**

References **15**

---

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 引用

1.3 术语及缩略词

缩写及相应全称

表 2: 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

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]

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. 用户终端软件

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 订阅控制与配置信息。

服务端软件提供

1. `Broker Pattern` 中的 `Broker` , 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, 用于从上述 `Broker` 中订阅 `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 ‘‘用户终端软件 ‘‘, 并处理用户与 ‘‘嵌入式终端的交互 ‘‘。

上述讨论的三个部分，每一个部分都由若干更小的构件组成。当将它们视为一个 “大构件” 时, 我们可以获得一个 UML 构件图，如下图所示。

#TODO 此处应该给出总览 UML 构件图

此外，我们还将继续在软件体系结构 一节中讨论上述三个 “大构件” 的内部组成, 在此先提前给出考虑了细节的 UML 构件图。

#TODO 此处应该给出完整版 UML 构件图

嵌入式终端软件架构

嵌入式终端软件由以下构件组成

1. Lora 设备驱动 (shattuckite-driver-lora)
2. Wifi 设备驱动 (shattuckite-driver-wifi)
3. 嵌入式终端核心 (shattuckite-core)
4. 本地数据缓存 (redis)

设备驱动将与硬件 (例如 `Lora` 网关或 `Wifi` 网卡) , 将传感器数据和执行器的状态映射为 `Linux` 文件。终端核心需要读取传感器数据时, 具体表现为读一个文件; 终端核心需要写执行器状态时，具体表现为写一个文件。即终端核心与设备驱动通过 Linux 系统调用 (`FileIO`) 交互数据。

在系统开机时, 嵌入式终端核心将会首先完成 初始化操作。初始化时，系统完成的工作包括

1. 检查服务器的可用性，清空本地缓存。
2. 从服务器获取传感器配置信息，并载入本地 Redis 缓存
3. 通过设备驱动，检查本地传感器列表与远程列表是否匹配, 如果不匹配，执行更新操作。
4. 以发布者的身份, 向服务器 Broker 订阅一个 Topic1, 用于发布数据.
5. 以订阅者的身份, 向服务器 Broker 订阅另一个 Topic2, 用于接收并处理命令

该过程用 UML 时序图表示如下图

..TODO 此处需要添加 UML 时序图

在核心初始化完成后, 核心进入 工作状态, 并行的完成下列工作

1. 根据本地缓存中的配置数据, 定时将传感器的数据发布到 Topic1.
2. 监听 Topic2，执行接收到的命令, 并将命令执行结果发布到 Topic1.

嵌入式终端软件的构件图如下图所示

..TODO 添加构件图

服务器端软件架构

服务端软件由以下构件组成

1. MQTT 协议代理 (shattuckite-mqtt-broker)
2. 数据管道 (shattuckite-data-pipe)
3. RESTful 服务器 (shattuckite-restful)
4. 推送服务器 (shattuckite-server-push)
5. 关系型数据库

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 (shattuckite-data-pipe) 一节中详细讨论.

TODO:partial case: 传感器数据持久化时序图

TODO: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 (shattuckite-restful) 一节。

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。

TODO:content: 加入动机交叉引用

TODO:partial case: 用户访问历史数据

TODO:partial case: 用户控制执行器

TODO:partial case: 用户改变传感器配置

关于服务端关系型数据库的使用, 请参考:ref:‘数据库设计 ‘一节, 此处不过多赘述。

TODO:uml: 加入该构件的详细构件图

用户端软件架构

用户端软件由以下构件组成

1. 视图模型 (shattuckite-gui-viewmodel)
2. 浏览器视图 (shattuckite-gui-browserview)
3. 移动端视图 (shattuckite-gui-nativeview)

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. 服务端软件架构中的关系型数据库
2. 嵌入式硬件的状态.

视图模型 (shattuckite-gui-viewmodel) 将作为 `MVVM` 中的 `Viewmodel` 角色。该构件应该能够从 `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时, 需要借助一个消息队列, 以 `Broker Pattern` 的方式异步的获取数据, 即服务器端软件架构 中提到的 `Defered` 接口。

浏览器视图 (shattuckite-gui-browserview) 和 移动端视图 (shattuckite-gui-nativeview) 将通过两种 `Data-Binding` 与 `ViewModel` 交互数据, 具体的讲

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。

TODO: 构件图

TODO:partial case: 用户访问历史数据 TODO:partial case: 用户访问实时数据 TODO: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。

TODO: 插入 Graphviz 图片

技术体系结构

支撑体系结构

系统部署方案

TODO: 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

4.2 系统接口说明

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

5.3 ER 图建模

6 详细设计

6.1 META-详细设计

本节主要阐述构件的详细设计。

7 运行与开发环境

8 需求可追踪性说明

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.

[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.