**物联网中间件**

**功能需求文档**

|  |  |
| --- | --- |
| **文档拟制人** | 张静斌 |
| **日期** | 2017-4-15 |

**文档修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **更新时间** | **变更内容** | **变更理由** | **修改人** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1. 概述 5](#_Toc481695250)

[2. 相关技术 6](#_Toc481695251)

[2.1 SENSIP模型 6](#_Toc481695252)

[2.2 SensorAct模型 8](#_Toc481695253)

[2.3 CEP复杂事件处理（Esper） 8](#_Toc481695254)

[2.4 REST接口规范 10](#_Toc481695255)

[2.5 MQTT协议 11](#_Toc481695256)

[3. 整体框架设计 13](#_Toc481695257)

[3.1基于事件的按需作业部署中间件设计 13](#_Toc481695258)

[3.2设备节点层 14](#_Toc481695259)

[3.3服务端中间件层 15](#_Toc481695260)

[3.4应用层 16](#_Toc481695261)

[4. 主要功能设计 16](#_Toc481695262)

[4.1数据收集 16](#_Toc481695263)

[4.1.1模块描述 16](#_Toc481695264)

[4.1.2功能需求 17](#_Toc481695265)

[4.1.3模块概要设计 17](#_Toc481695266)

[4.2数据处理 19](#_Toc481695267)

[4.2.1模块描述 19](#_Toc481695268)

[4.2.2功能需求 19](#_Toc481695269)

[4.2.3模块概要设计 21](#_Toc481695270)

[4.3复杂事件检测及处理 22](#_Toc481695271)

[4.3.1模块描述 22](#_Toc481695272)

[4.3.2功能需求 22](#_Toc481695273)

[4.3.3模块概要设计 23](#_Toc481695274)

[4.4安全功能 26](#_Toc481695275)

[4.4.1模块描述 26](#_Toc481695276)

[4.4.2功能需求 26](#_Toc481695277)

[4.4.3模块概要设计 27](#_Toc481695278)

[4.5上层接口 27](#_Toc481695279)

[4.5.1模块描述 27](#_Toc481695280)

[4.5.2功能需求 28](#_Toc481695281)

[4.5.3模块概要设计 28](#_Toc481695282)

[4.6脚本扩展 29](#_Toc481695283)

[4.6.1模块描述 29](#_Toc481695284)

[4.6.2功能需求 29](#_Toc481695285)

[4.6.3模块概要设计 29](#_Toc481695286)

# 概述

1. **文档目的：**

本文档主要对北京大学智能计算与感知实验室开发的Mid-ICS物联网中间件的功能设计和具体实现思路进行介绍，方便日后开发人员进行参考和交流。

1. **项目简介：**

Mid-ICS是在宾夕法尼亚州立大学SENSIP中间件及开源中间件SensorAct基础上，结合复杂事件处理（CEP）、发布/订阅及功能自适应配置技术实现的用实时数据流检测和分析的物联网中间件。其主要分为服务端与设备端两个模块组成，该文档主要介绍设备端的具体功能需求和实现。

1. **创新点：**

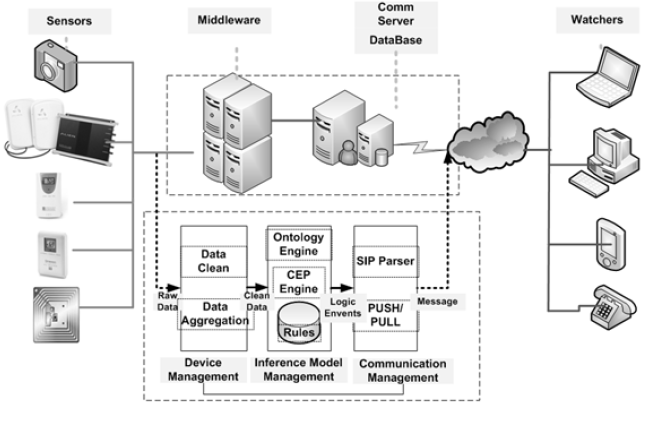
* 允许用户以自定义脚本形式进行功能扩展，极大增强了中间件的可扩展性。
* 对内部功能逻辑运行划分用户运行态和系统运行态，利于开发人员的使用和系统安全。
* 允许功能的自适应迁移，例如，当网络条件良好或者设备端不支持该功能脚本运行时，我们允许物端将采集的实时数据流直接传输到服务端，以利用服务端高效的处理性能；当网络条件较差时或者设备端处理性能满足既定要求，我们可以将脚本迁移至物端运行，以节约带宽。
* 提供高效的功能脚本调度，提高运行效率。//这不是创新点

# 相关技术

## 2.1 SENSIP模型

本工程的系统模型很大程度上学习了SENSIP 中间件的模型，并在此基础上做了机构性改进。SENSIP主要实现捕获传感器网络原始数据，对数据进行组织和智能化处理，并实现同SIP网络的互联互通。

系统包括传感器（Sensors）、中间件（Middleware）、通讯和数据库服务器（Comm Server/Database）和客户端（Watchers）。



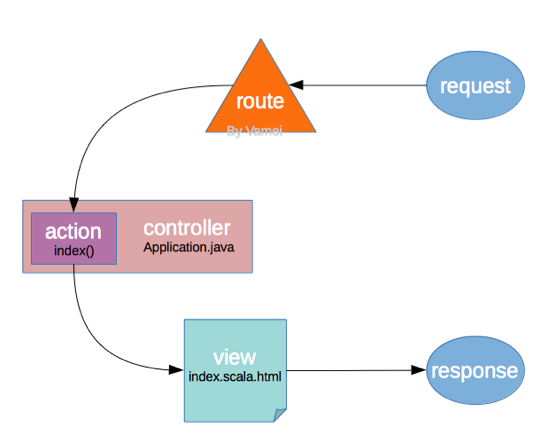
**图 1 SENSIP系统**

SENSIP系统的层次结构如**图 2**所示。四个层次自下而上分别为：传感器层（Sensor Layer）、抽象层（Abstract Layer）、融合层（Fusion Layer）、应用层（Application Layer）。其中中间件位于抽象层。

* 传感器层从各种传感器获取原始数据。
* 抽象层的主要功能有三个：一是对数据的预处理，包括对数据的平滑、聚合等以获得高质量的数据。二是对数据进行语义化、智能化处理，即将原始数据转换成可读的信息。三是完成数据向高一层进行发布的必要处理和格式化。
* 融合层实现所有通讯实体通过标准SIP接口进行通讯。
* 应用层包括SIP客户端软件、语音网关或手持设备等。

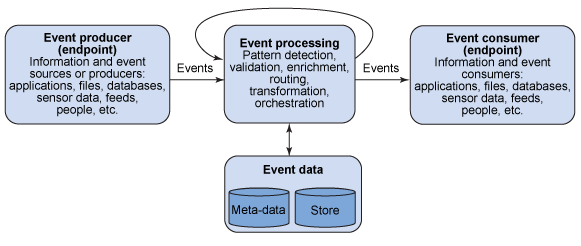
## 2.2 SensorAct模型

SensorAct利用Java语言和Play！框架进行开发，优点是代码库丰富且代码逻辑清晰，过程的自动化构建能力很强，对此我们进行了借鉴和学习，很大提高了效率。



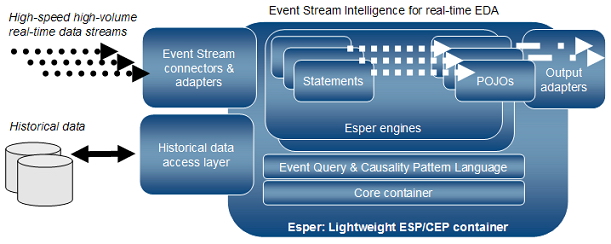
## 2.3 CEP复杂事件处理（Esper）

CEP将数据看做一种数据流，基于规则引擎对业务过程中持续产生的各种事件进行复杂的处理，能够实现对连续数据的快速分析处理。



CEP在逻辑上应该包括：

1. 事件发生器： 通过应用系统、文件系统、数据库、互联网、人工、以及传感器产生事件
2. 事件处理器： 模式的匹配、验证和改进，路由，转换以及编排
3. 事件消费者： 与事件发生器类似，也可以是应用系统、文件系统、数据库、互联网、人工界面等

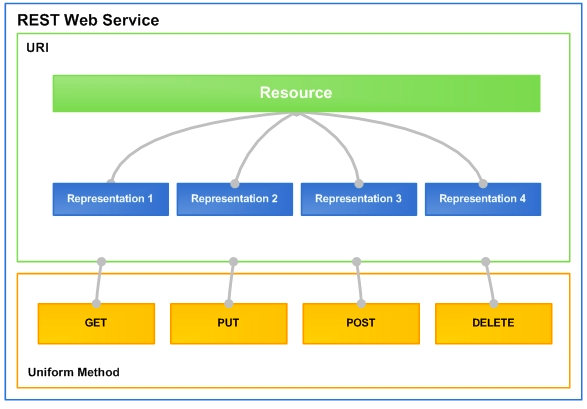


Mid-ICS中间件采用了Esper复杂事件处理引擎。如上图所示，Esper是一个面向Java语言的对实时数据流进行事件流处理和复杂事件处理的框架。该引擎可应用于网络入侵探测，SLA监测，RFID读取，航空运输调控，金融方面(风险管理，欺诈探测)等领域。它的特点是能够快速开发出复杂的实时计算策略，并且有着高吞吐量以及低延迟的特点，特别适合大量数据的实时计算。

## 2.4 REST接口规范

REST即表述性状态传递（英文：Representational State Transfer，简称REST）是Roy Fielding博士在2000年他的博士论文中提出来的一种[软件架构](http://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E6%9E%B6%E6%9E%84)风格。它是一种针对[网络应用](http://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%BA%94%E7%94%A8)的设计和开发方式，可以降低开发的复杂性，提高系统的可伸缩性。

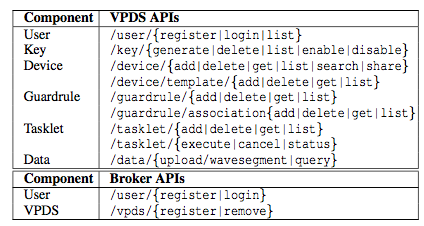
RESTful service是一种架构模式，近几年比较流行了，它的轻量级web服务，发挥HTTP协议的原生的GET，PUT，POST，DELETE。 REST模式的Web服务与复杂的SOAP和XML-RPC对比来讲明显的更加简洁，越来越多的web服务开始采用REST风格设计和实现。



REST服务关键原则：

* 1. 给一切物体一个ID
  2. 连接物体在一起
  3. 使用标准方法
  4. 资源多重表述
  5. 无状态通信

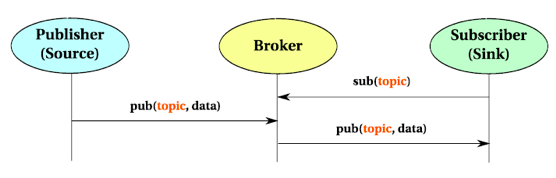
基于REST的使用，我们上层以下形式进行通信，使得编程开发和用户使用都会变得简答易懂。



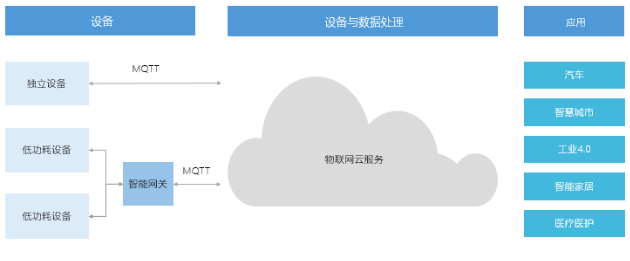
## 2.5 MQTT协议

MQTT (Message Queuing Telemetry Transport) 是由IBM研发的构建在TCP/IP之上的简单轻量的消息协议，目标使用场景为受限制环境，如低带宽、高延迟、不可靠网络，很适用于M2M和IoT中。

MQTT协议最早于1999年发布第一版并被广泛使用在各个行业中，在2013年发布MQTT 3.1版本。MQTT-S(MQTT for Sensor Networks)是专门为WSN（无线传感器网络，Wireless Sensor Network）设计的协议，是MQTT的修改版本，1.2版本于2014年推出，可用于zigbee网络。 MQTT的通信模型:



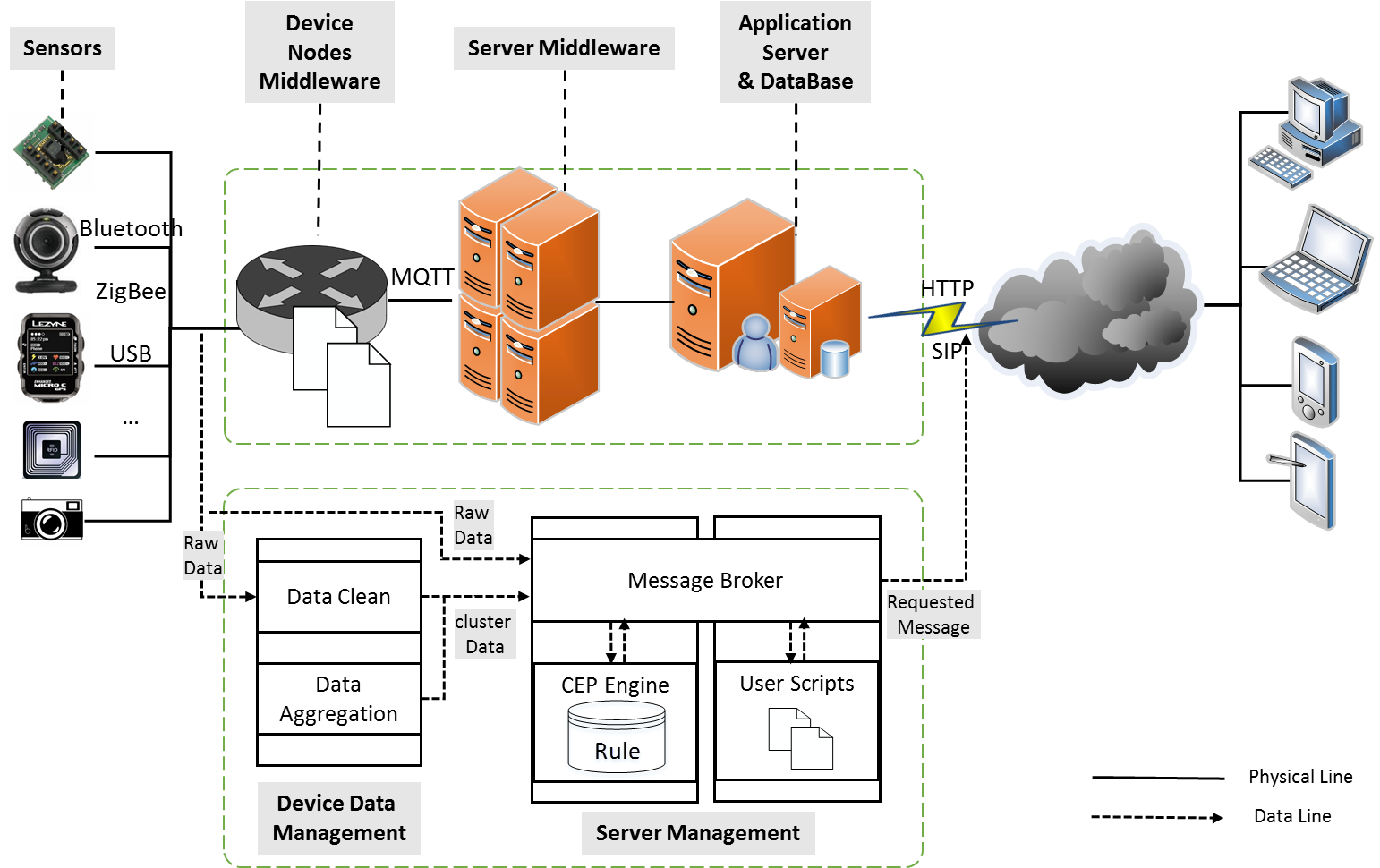
基于MQTT我们可以设计多种使用场景，如下图所示：



# 整体框架设计

## 基于事件的按需作业部署中间件设计

MID-ICS中间件是在SenSip及SensorAct两个开源中间件的基础上，引入张颖老师在《网络环境中应用的按需部署技术研究》中提出的应用功能的按需部署实现，以及Pogo和SenSocial两个移动感知中间件所采用的数据处理流程框架，而设计出的面向大规模异构智能设备的基于事件流的实时数据处理中间件模型。相较于现有中间件，其具有环境自适应、高可扩展性、负载均衡等特点。具体的架构设计如下图所示，其中实线部分表示系统的物理链路模型，虚线部分表示系统的数据流向模型。



对于物理实体部分，从左向右，分别包括设备节点层、服务端中间件层以及应用层。设备节点层主要负责底层异构传感器的数据采集，数据清洗及聚合，以及数据到服务端的传输；其中，数据的传输采用基于发布订阅模式的实时通讯协议MQTT。相较于http的持续性轮询方式，MQTT始终保持双方的长连接，以一种push based方式来维持消息的传输，相比较而言有着较低的传输能耗；另外，对比同为实时通讯协议的XMPP，由于并不需要保持XML的消息格式， MQTT更为轻量，更加适合物联网设备资源受限的特性。综合考虑，我们采用MQTT作为设备节点同服务端的交互协议。服务端中间件层主要负责复杂事件的检测及处理、按需的功能调度与部署、系统及用户自定义逻辑的运行时环境管理等。应用层负责将中间件层提供的部分功能封装成应用可以直接调用的接口，例如，与用户自定义逻辑脚本相关的功能函数接口，以及逻辑脚本上传的接口等。另外，应用层向上提供两种应用层协议（HTTP、SIP）供不同场景下的异构应用进行调用。

对于数据流向部分，从图中可以看出，系统可以根据环境需求动态地调整底层数据处理逻辑的位置，例如：当网络条件较差且底层设备处理性能足够满足数据处理需求时，服务端中间件会将相关的数据处理逻辑置于设备节点执行；相反的，当网络条件良好时，用户可能更加需要的是服务端更高的处理性能，因此在底层设备电量等资源足够支持的情况下，系统会将底层的处理逻辑回迁会服务端执行。通过这样，系统充分利用各部分资源，达到负载平衡的需求。后续的，在系统运行时环境下，服务端会对数据流进行相关的复杂事件检测及处理；在用户自定义运行时环境下，服务端会综合利用配置管理、历史数据以及具体功能的组件实现在数据及管理层面上用户自定义功能的实现等。

## 设备节点层

设备节点层主要负责传**感器数据的采集**、**传感器管理**以及**数据处理逻辑的运行时管理**三个模块。

其中，数据采集模块是指从具体的异构传感器读取数据的模块，为了迎合物联网的异构环境，需要尽可能多地支持底层协议，现阶段需兼容Bluetooth、USB、ZigBee等最普遍的协议。

传感器管理负责上述提过的数据采集模块，同时还需要负责具体某个传感器控制参数的调节以及某个制动器运转模式的切换等相应运行时的控制功能。因此引入消息代理结构，传感器管理器可以订阅相应的控制参数，同时发布原始的感知数据，实现控制与采集，控制逻辑与底层实现的解耦，用以提高工作效率。

至于逻辑脚本的运行时管理，涉及具体数据处理功能的实现，需要满足如下几个需求：

1. 支持多脚本高效的基于优先级的并发调度与运行， 例如：一个设备节点可以同时对温度与湿度两种传感器的采集的数据进行相应的预处理。
2. 脚本的运行时动态更新、安装、执行、暂停以及卸载，例如：服务端可以随时对设备端的某个脚本进行更新，而不会影响其他功能脚本的正常运行。
3. 支持脚本之间在数据及功能层面上的依赖关系，即某个脚本可以调用另一个脚本定义的功能或者调用另一个脚本的某些执行结果。

逻辑脚本也通过消息代理来订阅感兴趣的消息，同时发布相应的计算结果，这样可以实现需求三中数据层面的脚本依赖。同时，消息代理还负责将最终的处理结果转发到服务器端。一个可能的场景如下图所示：



另外，由于需要支持数据处理逻辑的按需执行，因此有必要在消息代理中引入一个Filter，对数据流的流向进行相应的配置。Filter的修改与逻辑脚本的按需迁移执行同步，均由服务端中间件中的脚本管理进行控制。具体的Filter可以采用XML的形式，描述如下问题：

1. 我最终向服务端提交什么样的数据；
2. 针对某一个传感器，那些脚本需要执行，也就是描述具体每个脚本的运行状态；

注意：Filter描述的脚本信息要符合脚本的依赖关系，比如script1依赖script2，script2依赖script3，那么就不能出现script3暂停运行，其他两个脚本运行这种状态。

由于考虑到底层设备有限的存储能力，在设备节点不提供消息的永久存储。

## 服务端中间件层

相较于设备节点，服务器中间件层主要分为**系统运行时环境**和**用户运行时环境**。



系统运行时环境的功能包括**复杂事件检测及处理**，**事件及规则的永久存储**以及**可供应用调用的功能库。**其中，复杂事件的检测及处理采用了当前流行的复杂事件处理引擎Esper；事件及规则的存储使用的是非SQL的MongoDB数据库；可供应用调用的功能库，则是对一些通用的控制功能以及基于事件流的数据处理功能（例如，通过机器学习等方法训练好的数据分类器）进行封装，使得应用可以直接通过上层API调用这些功能，或者被用户自定义的逻辑脚本调用实现更高维的逻辑功能。

用户运行时环境的功能包括**自定义逻辑脚本的管理**，**底层数据处理逻辑的按需迁移**以及**逻辑脚本的永久存储**。自定义逻辑脚本的管理，主要需求是实现脚本的高效调度与执行，采用Quartz（作业调度框架）以及Ehino（JS脚本的Java运行时框架）来实现基于脚本优先级的最优化任务调度与执行；底层数据处理逻辑的按需迁移，综合考虑网络环境，底层设备资源、处理能力以及服务端处理能力，均衡资源利用、网络负载及数据处理的时效性。根据环境变化，实现底层数据处理逻辑服务器端与设备节点的按需迁移；逻辑脚本的永久存储，也是采用的MongoDB作为数据库。

## 应用层

应用层的主要功能就是将中间件层所设计的逻辑脚本上传、封装好的功能库以及一些控制操作等具体功能打包形成相应的Restful风格的云服务，通过诸如HTTP等应用层协议供其他应用远程使用。目前，设计兼容的协议包括HTTP (吸取SensorAct的经验，采用PLAY框架， 已经实现) 以及SIP (至于SIP具体如何去兼容Restful Web Servic，就交给何硕了= =）。

# 主要功能设计

## 数据收集

### 模块描述

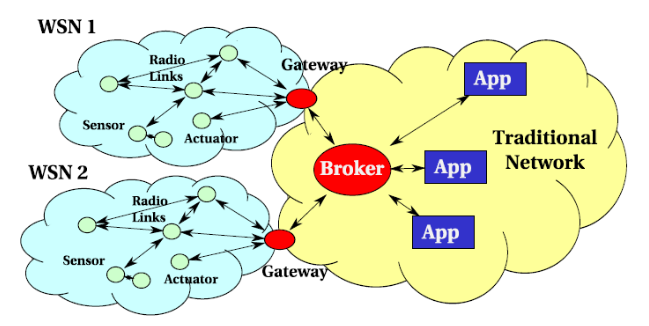
数据收集模块是指从物端采集数据，是中间件的下层接触物端的模块，这里采用MQTT这一被广泛使用的通信协议进行数据采集，由于物端的数据具有很强的异构性，数据采集时对于物端信息传输进行初步的格式化要求，必要情况下对于物端进行脚本设置。

### 功能需求

1. 根据传感器数据特点实现数据的单次请求或者长连接
2. 以发布订阅的形式进行数据采集，能够对于多种形式进行兼容
3. 对于异构数据具有兼容性
4. 保证数据的有效性和低延时

### 模块概要设计

* **发布订阅模式：**



* **实现发布订阅模式**

[MQTT](http://mqtt.org/)是IBM开发的一个即时通讯协议。MQTT是面向M2M和物联网的连接协议，采用轻量级发布和订阅消息传输机制。

[Mosquitto](http://mosquitto.org/)是一款实现了 MQTT v3.1 协议的开源消息代理软件，提供轻量级的，支持发布/订阅的的消息推送模式，使设备对设备之间的短消息通信简单易用。

1. **安装Mosquito**

此处以mosquitto-1.4.5为例

# 下载源代码包

wget <http://mosquitto.org/files/source/mosquitto-1.4.5.tar.gz>

# 解压

tar zxfv mosquitto-1.4.5.tar.gz

# 进入目录

cd mosquitto-1.4.5

# 编译

make

# 安装

sudo make install

1. **测试**

2.1 启动代理服务

mosquitto –v

【-v】打印更多的调试信息

2.2 订阅主题

mosquitto\_sub -v -t sensor

【-t】指定主题，此处为sensor

【-v】打印更多的调试信息

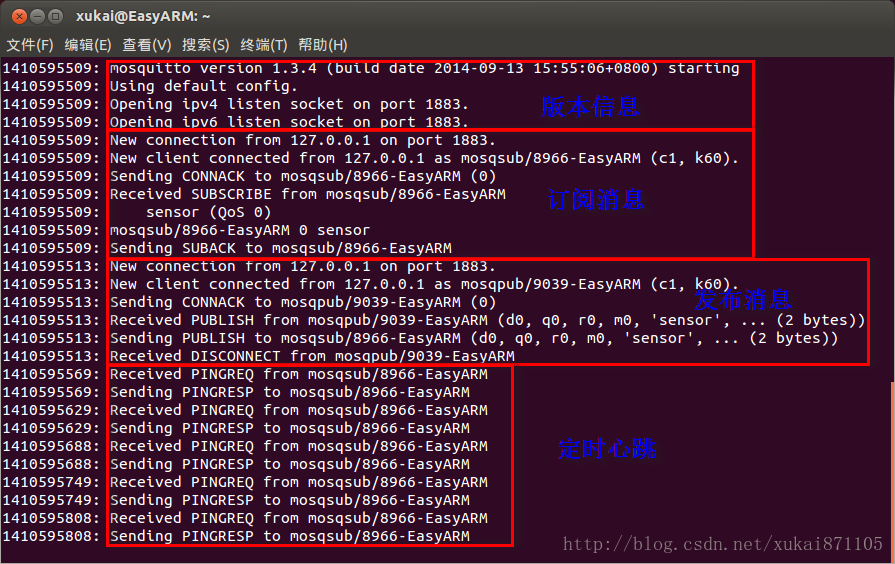
2.3 发布内容

mosquitto\_pub -t sensor  -m 12

    【-t】指定主题

    【-m】指定消息内容

1. **运行结果**



## 数据处理

### 模块描述

数据处理模块由数据过滤模块和数据实时存储模块两大部分构成。智能感知中间件系统应该提供必要的数据过滤功能，再将经过处理的数据转发给复杂事件处理模块。这不但可以提高计算和存储资源的利用率，同时也将提高数据处理效率。

相比基于传统关系数据库的数据处理，传感数据流处理对实时性与高效性提出了更高的要求，使用实时内存数据库管理海量实时数据，对高效的数据处理提供支持是必要的功能。内存数据库是将数据直接放在内存中直接操作的数据库，相对于磁盘，内存的数据读取速度要高出几个数量级，将数据保存在内存中相比从磁盘上访问能够极大的提高数据存取性能。

同时传感数据处理面向极大数据量，传统关系数据库对于超大规模简单数据表不能提供良好的支持，数据的后台存储应提供基于分布式的数据存储模型。

### 功能需求

数据过滤功能可以提高计算和存储资源的利用率，同时也将提高数据处理效率。数据过滤模块需满足以下功能需求：

**（1）提供数据过滤功能，包括数据阈值过滤等；**

在智能感知中间件的实际应用环境中，中间件从分布在各处的传感器中采集大量数据，然而并非所有数据都是有用或有效的，数据存在误差和误读等错误，为了节省存储与计算资源。

**（2）提供数据聚合功能，包括冗余数据消除等；**

数据的冗余读取是常见的数据错误之一，冗余读取一方面是因为传感器之间数据的重复性，另一方面也是保证数据能够正确获取，冗余数据会造成计算资源以及存储资源的浪费，冗余数据消除技术是针对冗余读取的数据聚合功能。

**（3）时间数据平滑以及空间数据平滑功能；**

传感数据具有时间性和空间性，数据处理往往无需处理最低时间和空间维度数据，例如每5秒钟对于数据进行一次整合，最有效的技术是数据时间平滑以及数据空间平滑技术。通过数据时间平滑和空间平滑，能够将精简数据量，提高数据有效性等。

**（4）能够实现数据处理中的约束功能，提供伴随约束、路径约束、动态约束等描述方式；**

通过底层的平滑机制可以消除大部分的漏读事件，但是没有一种机制能够消除全部的错误，需要多种措施共同作用进行完善的数据过滤功能。在数据处理的过程中，应提供对数据约束描述以及约束方法的支持。约束方法可以基于不同的约束条件，包括对象的物理属性、伴随属性、运动路径、关联关系等。

传感数据流处理对实时性与高效性提出了更高的要求，构建基于分布式系统的后台存储架构以及基于内存的实时数据库系统，数据实时存储模块应满足以下功能需求：

**（1）提供基于分布式文件系统的海量数据存储模型；**

传感数据处理面向极大数据量，传统关系数据库对于超大规模简单数据表不能提供良好的支持，数据的存储应提供基于分布式的数据存储模型。

**（2）设计并实现基于内存的实时数据存储模型；**

实时内存数据库（RMDB，Real-time Memory Database）将数据常驻内存，相对于磁盘有更高的存储速度，缩短了响应时间提高了系统的性能。

**（3）数据库的创建、打开、关闭、删除、重命名等，以及对数据的检索和增删改操作；**

**（4）提供附加功能**

例如数据库状态信息、读取所在文件的信息、读取所在数据库环境的信息、清空数据库的内容、数据库的同步备份、版本升级、提示出错信息等等；

**（5）满足给定的数据存储和查询的性能要求**

### 模块概要设计

数据处理模块主要由数据过滤和数据实时存储模块两大部分构成。图1是数据处理模块的主要架构图，其中数据过滤部分主要包括冗余过滤器.、时间空间平滑处理器及数据约束功能模块，数据存储模块由实时数据存储和分布式存储构成，分别提供存储、索引、查询等功能。本模块从下层传感器获得原始传感数据，将传感数据进行处理和存储，为上层复杂事件处理模块提供经过处理好的原始事件数据。

## 复杂事件检测及处理

### 模块描述

复杂事件处理模块将数据视为事件的发生，通过检测系统中的异常行为或设定的行为模式，复杂事件处理能够从事件数据流中挖掘复杂模式。复杂事件处理模块主要的功能由事件检测、事件查询语言构成。

### 功能需求

复杂事件处理（Complex Event Processing）是以事件驱动（Event Driven）的观点处理信息系统中产生的海量数据，可用于检测系统中的异常行为或者感兴趣的行为模式，预测未来事件等等。事件可以是已经发生或者正在发生的任何事物。事件可以来自于真实世界或者虚拟世界，例如飞机降落、火车开车、RFID传感器收发数据、数据库访问、网络服务等等。事件分为基本事件（Primitive Event）和复杂事件（Complex Event）。复杂事件是由基本事件由一定的裸机关系组合或派生出的事件。SENSIP中间件将传感器收集到的数据视为事件，通过复杂事件处理模块检测用户定义的复杂事件类型。SENSIP中间件中的复杂事件处理模块主要应具有以下功能：

**（1）事件检测功能**

复杂事件处理模块需从未经处理的原始事件通过检测，聚集等操作之后复合成更为高级的复合事件（Composite Event）。

**（2）事件串流处理**

在复杂事件处理中，为了要达到高吞吐量(throughput)、高度利用性(availability)、以及低度延迟(latency)以及实时决策的需求，应实现使用事件串流处理(event stream processing)。

**（3）事件查询语言**

复杂事件处理模块应实现SQL-LIKE的事件处理语言，事件查询语言应有一定的逻辑表示功能，对事件串流提供复杂的逻辑处理支持。常用的SQL-LIKE逻辑应包括SELECT, FROM, WHERE, GROUP BY, HAVING 以及ORDER BY等语句。事件构造逻辑应包括+, -, \*, /, %等数学算符，sum、avg、count、min/max、stddev等基本统计功能，NOT、OR、AND等逻辑运算以及&、|、^等二进制运算符。

**（4）事件过滤功能**

复杂事件处理模块应实现一定的事件滑动窗口和聚集、分组窗口、输出率限制等事件过滤功能。

**（5）基于优先级的事件调度**

目前国内外开发的复杂事件处理系统对复杂事件的处理采用是FIFO算法，即先接收到的复杂事件先进行处理。但在实际应用中，尤其是面对千万量级事件的应用，某些事件会比其他事件更为紧急。SENSIP中间件应采用基于事件优先级的调度算法，将事件优先级的调度技术应用到事件查询，系统为每一个查询请求赋予一个优先级，更符合实际的应用需求。

**（6）复杂事件描述机制**

包括滑动窗口描述机制，事件聚合机制，因果关系以及关联机制等复杂事件检测描述机制，扩展描述语言的表达能力；

### 模块概要设计

复杂事件处理模块从基本事件中检测由一定的逻辑关系组合或派生出的复杂事件。SENSIP中间件中的复杂事件处理模块主要应具有以下功能：

（1）事件检测功能

（2）事件串流处理

（3）事件查询语言

（4）事件过滤功能

（5）基于优先级的事件调度

（6）复杂事件描述机制

复杂事件检测模块在SENSIP中间件系统中的结构如图2所示。复杂事件处理模块从下层数据过滤模块获取经过数据过滤的基本事件，经过复杂事件检测模块中的窗口处理，再进行用户自定义的模式检测，最终形成输出事件流。同时复杂事件处理模型还具有对于用户的推送功能，用户能够将自定义的请求进行订阅，模块在检测到事件后能够主动将消息推送给用户。

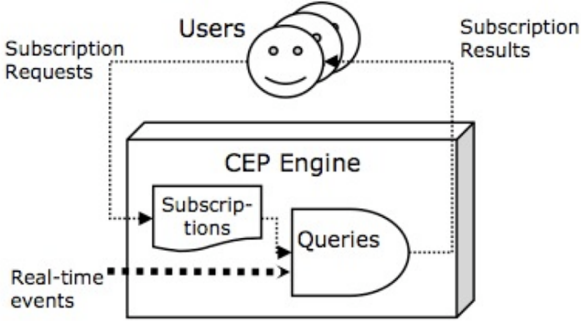
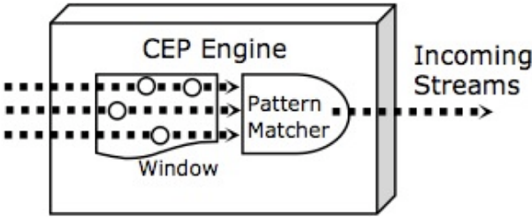


图2 复杂事件检测模块结构图

如图3所示，复杂事件检测模块对于每一个事件流构成一个时间窗口，模块使用滑动窗口机制对用户指定的窗口内的事件进行检测。事件聚合和因果关系等复杂逻辑检测也能在事件流中进行检测。

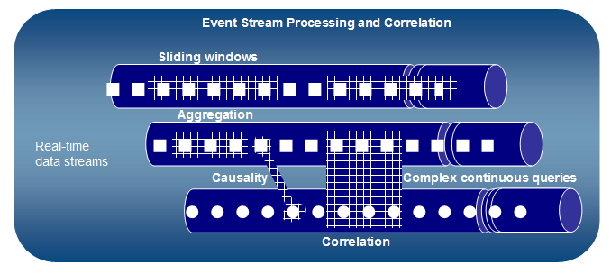


图3 事件流检测机制

复杂事件检测模块采用类似于SQL语言的事件处理语言（Event Processing Language ，EPL），该语言继承自SQL语言实现了SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY 等基础关键字，同时为了支持 事件流的窗口检测，提供滑动窗口关键字如win:length(t)，win:time(t sec)，win:time\_batch(t sec)，win:length\_batch(t)，滑动窗口关键字的事件流处理模式如图4-6所示。

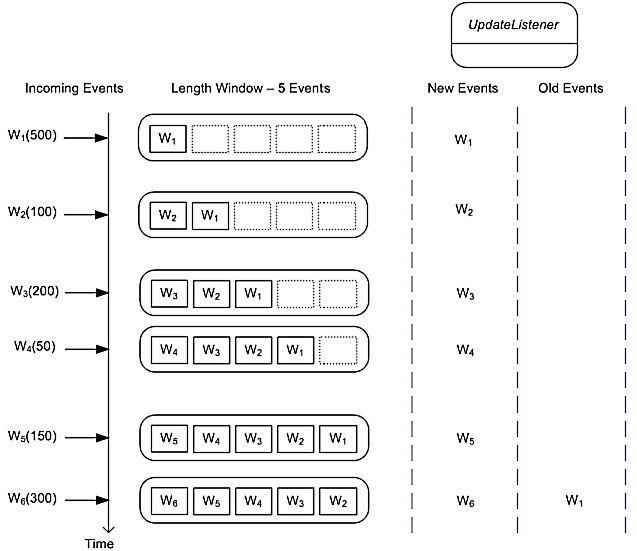


图4 事件连续窗口模式

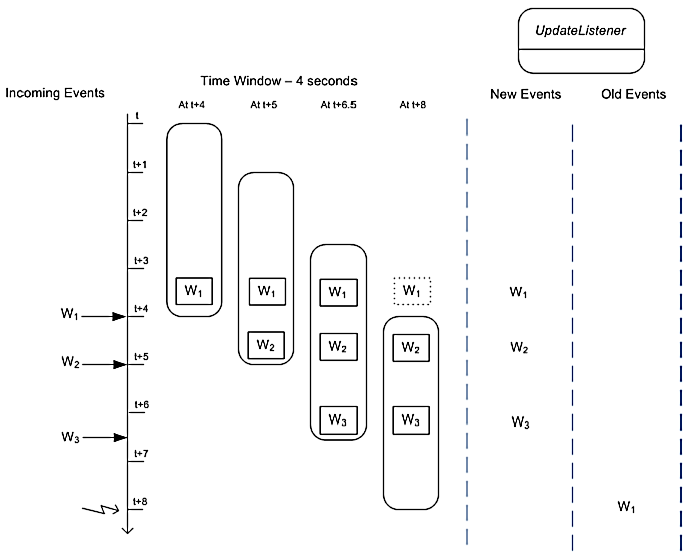


图5 时间连续窗口模式

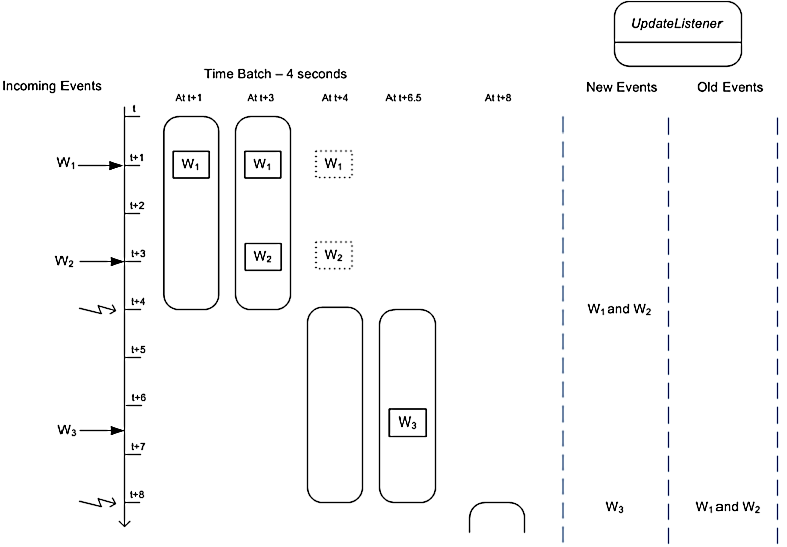


图6 时间分段窗口模式

## 安全功能

### 模块描述

安全功能模块是SENSIP中间件系统中的重要组成部分，安全功能应贯穿于中间件系统从底层传感层到上层数据处理层以及应用层，有效的安全功能有助保障系统功能。安全模块主要实现安全认证、安全数据聚合、数据加密以及隐私保护等重要功能。

### 功能需求

（1）安全认证

作为安全机制的核心与重要基础环节,如何在无线传感网的各种限制下,安全、高效、低能耗地实现认证,始终是无线传感网安全研究领域的热点。为了防止恶意节点注入非法信息或篡改数据等攻击,在传感器的身份认证和控制信息的中必须引入安全认证机制。

（2）安全数据聚合

存在安全问题或不可信的聚合数据会延误、误导甚至破坏整个网络的正常工作。不安全的数据聚合可以使得数据采集工作功亏一篑，SENSIP中间件需要在适当的数据处理环节上实现数据的安全聚合。

（3）数据加密

数据加密机制是防止数据窃听以及实现端到端数据隐私保护的有效方法，SENSIP中间件应实现高效的数据加密机制。

（4）隐私保护

数据与隐私是SENSIP系统面临的挑战之一。传感器需要收集人们的日常生活信息，如个人的旅游路线信息，购买习惯信息等，而这些信息一般属于个人的隐私信息。因此解决好应用过程中的隐私保护问题，是SENSIP中间件得到广泛应用的必要条件之一。隐私保护主要需要提供匿名化技术（传感器位置隐私保护、位置服务隐私保护、数据查询隐私保护）

### 模块概要设计

系统安全性对于物联网中间件来说显得尤为重要，开发之中既可以借助成熟的安全技术手段，又可以多角度进行安全防护，以保证系统的安全稳定。SENSIP中间件中的安全模块主要应具有以下功能：

1. 多重安全认证

安全认证使用常规认证技术能够达到要求，我们对于应用端和中间件配置中心都进行用户认证，能够提高系统安全性，对于设备终端涉及的无线传输，需要提高认证级别，避免信息截取。

1. 安全数据聚合
2. 数据加密：在应用端使用https传输提高信息的加密属性
3. 隐私保护：使用匿名化技术，避免敏感信息泄漏，特别是与用户位置、身份等相关的信息。

## 上层接口

### 模块描述

中间件屏蔽底层的复杂处理之后，最终需要为开发者和用户提供上层的API接口，方便开发，我们的中间件以REST的形式提供简单移动的接口便与开发。

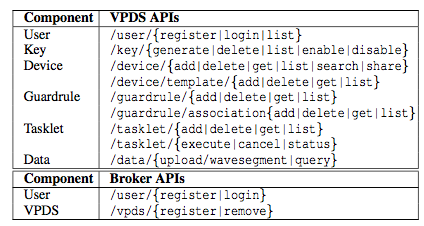
形如： POST http://www.ics.pku.edu.cn/user/login

### 功能需求

1. 上层开发接口应该使用被业界广泛接受的形式，以便被更多的开发者所接受，且有更为成熟的开发体系
2. 接口形式方便简洁，且安全可靠
3. 接口能够满足多种服务需求

### 模块概要设计

我们中间件以REST形式提供接口，其服务可以用以下形式进行访问和实现。



上层信息以JSON数据格式进行传递，形如：

{

"username": "samy1234",

"password": "mypassword",

"email": "pandarasamya@iiitd.ac.in"

}

JSON格式数据在不同开发语言之间具有很好的通用性，且有很明晰的结构对于开发者和机器都有很好的可读性。

## 脚本扩展

### 模块描述

中间件提供固定的开发API便于开发者进行调度使用，同时也为开发者提供了可以自由发挥的空间，允许开发者自行扩展js脚本，并进行多脚本的调度。

### 功能需求

1. 提供脚本规范
2. 划分用户开发的系统态和用户态
3. 允许自定义脚本，提供脚本对于中间件其他功能的调配功能
4. 提供多脚本的任务调度

### 模块概要设计

我们中间件以Java语言为基础进行开发，同时利用了Play！框架，这里我们选择JavaScript脚本语言进行扩展开发，并引入Rhino作为跨语言融合的包。

Rhino 是一种使用 Java 语言编写的 JavaScript 的开源实现，原先由Mozilla开发，现在被集成进入JDK 6.0。与其他很多语言一样，Rhino 是一种动态类型的、基于对象的脚本语言，它可以简单地访问各种 Java 类库。Rhino 从 JavaScript 中借用了很多语法，让程序员可以快速编写功能强大的程序。最为明显的区别是，Rhino 不再使用语句结束符( ; )，放宽了变量声明规则，并且极大地简化了修改和检索对象属性的语法（没有求助于调用存取方法）。