**1.1 MQTT协议的特点**

MQTT（Message Queuing Telemetry Transport，消息队列遥测传输）是IBM开发的一个即时通讯协议，有可能成为物联网的重要组成部分。该协议支持所有平台，几乎可以把所有联网物品和外部连接起来，被用来当做传感器和制动器（比如通过Twitter让房屋联网）的通信协议。

MQTT协议是为大量计算能力有限，且工作在低带宽、不可靠的网络的远程传感器和控制设备通讯而设计的协议，它具有以下主要的几项特性：

1、使用发布/订阅消息模式，提供一对多的消息发布，解除应用程序耦合；

2、对负载内容屏蔽的消息传输；

3、使用 TCP/IP 提供网络连接；

4、有三种消息发布服务质量：

（1）“至多一次”，消息发布完全依赖底层 TCP/IP 网络。会发生消息丢失或重复。这一级别可用于如下情况，环境传感器数据，丢失一次读记录无所谓，因为不久后还会有第二次发送。

（2）“至少一次”，确保消息到达，但消息重复可能会发生。

（3）“只有一次”，确保消息到达一次。这一级别可用于如下情况，在计费系统中，消息重复或丢失会导致不正确的结果。

5、小型传输，开销很小（固定长度的头部是 2 字节），协议交换最小化，以降低网络流量；

6、使用 Last Will 和 Testament 特性通知有关各方客户端异常中断的机制；

**1.2 MQTT协议实现方式**

实现MQTT协议需要客户端和服务器端通讯完成，在通讯过程中，MQTT协议中有三种身份：发布者（Publish）、代理（Broker）（服务器）、订阅者（Subscribe）。其中，消息的发布者和订阅者都是客户端，消息代理是服务器，消息发布者可以同时是订阅者。

MQTT传输的消息分为：主题（Topic）和负载（payload）两部分：

（1）Topic，可以理解为消息的类型，订阅者订阅（Subscribe）后，就会收到该主题的消息内容（payload）；

（2）payload，可以理解为消息的内容，是指订阅者具体要使用的内容。

**1.3 网络传输与应用消息**

MQTT会构建底层网络传输：它将建立客户端到服务器的连接，提供两者之间的一个有序的、无损的、基于字节流的双向传输。

当应用数据通过MQTT网络发送时，MQTT会把与之相关的服务质量（QoS）和主题名（Topic）相关连。

**1.4MQTT客户端**

一个使用MQTT协议的应用程序或者设备，它总是建立到服务器的网络连接。客户端可以：

（1）发布其他客户端可能会订阅的信息；

（2）订阅其它客户端发布的消息；

（3）退订或删除应用程序的消息；

（4）断开与服务器连接。

**1.5MQTT服务器**

MQTT服务器以称为“消息代理”（Broker），可以是一个应用程序或一台设备。它是位于消息发布者和订阅者之间，它可以：

（1）接受来自客户的网络连接；

（2）接受客户发布的应用信息；

（3）处理来自客户端的订阅和退订请求；

（4）向订阅的客户转发应用程序消息。

**1.6 MQTT协议中的订阅、主题、会话**

**一、订阅（Subscription）**

订阅包含主题筛选器（Topic Filter）和最大服务质量（QoS）。订阅会与一个会话（Session）关联。一个会话可以包含多个订阅。每一个会话中的每个订阅都有一个不同的主题筛选器。

**二、会话（Session）**

每个客户端与服务器建立连接后就是一个会话，客户端和服务器之间有状态交互。会话存在于一个网络之间，也可能在客户端和服务器之间跨越多个连续的网络连接。

**三、主题名（Topic Name）**

连接到一个应用程序消息的标签，该标签与服务器的订阅相匹配。服务器会将消息发送给订阅所匹配标签的每个客户端。

**四、主题筛选器（Topic Filter）**

一个对主题名通配符筛选器，在订阅表达式中使用，表示订阅所匹配到的多个主题。

**五、负载（Payload）**

消息订阅者所具体接收的内容。

**1.7 MQTT协议中的方法**

MQTT协议中定义了一些方法（也被称为动作），来于表示对确定资源所进行操作。这个资源可以代表预先存在的数据或动态生成数据，这取决于服务器的实现。通常来说，资源指服务器上的文件或输出。主要方法有：

（1）Connect。等待与服务器建立连接。

（2）Disconnect。等待MQTT客户端完成所做的工作，并与服务器断开TCP/IP会话。

（3）Subscribe。等待完成订阅。

（4）UnSubscribe。等待服务器取消客户端的一个或多个topics订阅。

（5）Publish。MQTT客户端发送消息请求，发送完成后返回应用程序线程。

**2.1 MQTT协议实际应用**

了解了MQTT的一些基本概念，剩下就是实际的应用。

公司采用的网关接口信息见《物联网云平台》网关通信说明（MQTT）。

上面的信息有些已经发生改变，阅读时要加以判断。

下面开始说具体的连接网关，并获得网关发送的topic信息的demo（Java实现）。

运行时要使用jar包，也可使用maven，但是使用maven时要注意版本。

具体的jar包和maven依赖在网址

<https://gitee.com/iots/mqtt-client>

依赖为：

<dependency>

<groupId>org.fusesource.mqtt-client</groupId>

<artifactId>mqtt-client</artifactId>

<version>1.12</version>

</dependency>

下面开始编写demo

首先先要配置MQTT的一些配置，配置比较多，也很繁琐。

主要是配置主机号和端口号，根据自己的配置编写代码，在配置其他的一些细节配置，主要是和连接有关的。

代码如下：

// MQTT设置说明

// 设置主机号

mqtt.setHost("tcp://10.168.5.208:1883");

// 用于设置客户端会话的ID。在setCleanSession(false);被调用时，MQTT服务器利用该ID获得相应的会话。此ID应少于23个字符，默认根据本机地址、端口和时间自动生成

mqtt.setClientId("876543210");

// 若设为false，MQTT服务器将持久化客户端会话的主体订阅和ACK位置，默认为true

mqtt.setCleanSession(**false**);

// 定义客户端传来消息的最大时间间隔秒数，服务器可以据此判断与客户端的连接是否已经断开，从而避免TCP/IP超时的长时间等待

mqtt.setKeepAlive((**short**) 60);

// 服务器认证用户名

mqtt.setUserName("admin");

// 服务器认证密码

mqtt.setPassword("admin");

// 设置“遗嘱”消息的话题，若客户端与服务器之间的连接意外中断，服务器将发布客户端的“遗嘱”消息

mqtt.setWillTopic("willTopic");

// 设置“遗嘱”消息的内容，默认是长度为零的消息

mqtt.setWillMessage("willMessage");

// 设置“遗嘱”消息的QoS，默认为QoS.ATMOSTONCE

mqtt.setWillQos(QoS.***AT\_LEAST\_ONCE***);

// 若想要在发布“遗嘱”消息时拥有retain选项，则为true

mqtt.setWillRetain(**true**);

// 设置版本

mqtt.setVersion("3.1.1");

// 失败重连接设置说明

// 客户端首次连接到服务器时，连接的最大重试次数，超出该次数客户端将返回错误。-1意为无重试上限，默认为-1

mqtt.setConnectAttemptsMax(10L);

// 客户端已经连接到服务器，但因某种原因连接断开时的最大重试次数，超出该次数客户端将返回错误。-1意为无重试上限，默认为-1

mqtt.setReconnectAttemptsMax(3L);

// 首次重连接间隔毫秒数，默认为10ms

mqtt.setReconnectDelay(10L);

// 重连接间隔毫秒数，默认为30000ms

mqtt.setReconnectDelayMax(30000L);

// 设置重连接指数回归。设置为1则停用指数回归，默认为2

mqtt.setReconnectBackOffMultiplier(2);

// Socket设置说明

// 设置socket接收缓冲区大小，默认为65536（64k）

mqtt.setReceiveBufferSize(65536);

// 设置socket发送缓冲区大小，默认为65536（64k）

mqtt.setSendBufferSize(65536);

// 设置发送数据包头的流量类型或服务类型字段，默认为8，意为吞吐量最大化传输

mqtt.setTrafficClass(8);

// 带宽限制设置说明

// 设置连接的最大接收速率，单位为bytes/s。默认为0，即无限制

mqtt.setMaxReadRate(0);

// 设置连接的最大发送速率，单位为bytes/s。默认为0，即无限制

mqtt.setMaxWriteRate(0);

// 选择消息分发队列

// 若没有调用方法setDispatchQueue，客户端将为连接新建一个队列。如果想实现多个连接使用公用的队列，显式地指定队列是一个非常方便的实现方法

mqtt.setDispatchQueue(Dispatch.*createQueue*("foo"));

上面都是一些配置的问题，具体情况自己决定配置。具体的配置也可以参考下面的网址，下面的网址也有详细的描述。

https://gitee.com/iots/mqtt-client

下面开始讲讲连接和订阅和发送主题

fusesource提供三种mqtt client api，分别为阻塞API，基于Futur的API和回调API。

**其中，**阻塞API是在MQTT.connectBlocking方法建立连接和提供阻断API的连接。

基于Futur的API则是：在MQTT.connectFuture方法建立连接，为您提供了一个与结合Futur的连接。所有操作的连接是无阻塞的，并且经由返回的结果。

**回调API是最复杂的也是性能最好的，另外两种均是对回调API的封装。**

**因为回调API有些复杂，现在只是介绍回调API的封装。就是前两个，前两个的区别是第一个为阻塞的，第二个不是阻塞。下面开始代码演示。**

**第一个**阻塞API。代码如下：

//接收消息

Message message = connection.receive();

//将收到的消息封装为消息类

ReceiveRealValiedateVO realValiedateVO= JSON.*parseObject*(message.getPayloadBuffer().toByteArray(), ReceiveRealValiedateVO.**class**);

System.***out***.println(realValiedateVO.getG\_id());

**if**(realValiedateVO.getS\_id().equals("1") && realValiedateVO.getS\_id().equals("1")) {

//新建json

JSONObject jsonParam = **new** JSONObject();

//封装json

jsonParam.put("s\_id",realValiedateVO.getS\_id());

jsonParam.put("g\_id",realValiedateVO.getG\_id());

jsonParam.put("seq",realValiedateVO.getSeq());

jsonParam.put("type",realValiedateVO.getType());

jsonParam.put("valid","true");

String s=jsonParam.toString();

//发送特定主题的消息

connection.publish("datasources/1/1/req\_ack", s.getBytes(), QoS.***AT\_LEAST\_ONCE***, **false**);

}

//打印主题

System.***out***.println(message.getTopic());

//byte[] payload = message.getPayload();

System.***out***.println(String.*valueOf*(message.getPayloadBuffer()));

// process the message then:

message.ack();

//连接断开

//connection.disconnect();

具体的解释都在代码里了，剩下就没有什么了。

要注意的点就是连接中断的处理，和对于服务器的处理。

第二种就是使用future连接,代码如下：

//使用future连接

FutureConnection connection = mqtt.futureConnection();

Future<Void> f1 = connection.connect();

f1.await();

//订阅消息

Future<**byte**[]> f2 = connection.subscribe(**new** Topic[]{**new** Topic("datasources/1/1", QoS.***AT\_LEAST\_ONCE***)});

//

**byte**[] qoses = f2.await();

// 发送身份验证消息.

//Future<Void> f3 = connection.publish("foo", "Hello".getBytes(), QoS.AT\_LEAST\_ONCE, false);

// 接收订阅消息..

Future<Message> receive = connection.receive();

// 打印消息.

Message message = receive.await();

System.***out***.println(String.*valueOf*(message.getPayloadBuffer()));

//回应

message.ack();

//

Future<Void> f4 = connection.disconnect();

f4.await();

下面就说说网关发送的消息和云平台发送的消息。

网关发送的消息有5种。分别为：身份验证消息，采集点描述信息，采集点数据上报消息，校时请求消息，云平台指令反馈消息。

云平台发送至网关的消息有5种。分别为：身份验证响应消息，校时响应消息，写入采集点消息，获取设备的采集日志信息消息。采集开启和关闭消息。

各个消息的格式见《物联网云平台》网关通信说明（MQTT）文档

要写的类

身份验证类

属性： s\_id(网关的厂商编号),g\_id（网关编号）,type（消息类型）,ver（网关程序版本号）,pro\_mo（网关产品型号）,seq(发送数据的标识,用于去重，防止MQTT发送重复数据),vaild(如果合法，返回true，否则返回false)。

采集点描述类

属性：type(消息类型),data\_ver(采集信息版本),meters（采集通道下的设备）

{channel\_id(采集通道的id)，channel\_desc(采集通道的描述)，plug\_name(?),meter\_id(采集通道下的设备的id),meter\_desc(采集通道下的设备的描述),tags（采集点）

{id(采集点的id),name(采集点的名称),cod(数据类型),dir(dir表示数据的方向)}}

数据上报类

属性：type(数据类型),time(数据的时间),seq(发送数据的标志),meters(采集通道)

{channel\_id(采集通道id),meter\_id(设备id),io\_status(设备的io状态),send\_pkg(发送的帧数),rev\_pkg(接收数据的帧数),tags(数据采集标签)

{id(采集点id),error(选填字段),value(采集的值),err\_msg(错误信息)

}}

校时类

属性：s\_id(网关的厂商编号),g\_id(网关编号),type(数据类型),time(服务器当前的时间)

云平台指令类**cloud**instruction

属性：type(指令类型),cmd\_seq(命令序号),meter\_id(写入的设备id),tag\_id(写入数据标签的id),cod(数据标签的数据类型),value(写入的值),success(是否写入成功),msg(写入信息的说明),limit(要返回的近多少次的完整采集日志),meter\_log(返回的日志数组),op(开启或关闭网关的采集)

网关和云平台的topic

网关发送的

datasources/req 身份验证消息 datasources/{s\_id}/{g\_id}/req 采集点描述信息

datasources/{s\_id}/{g\_id} 数据上报 datasources/req 校时datasources/{s\_id}/{g\_id}/cmd\_ack 云平台指令反馈

云平台发送的

datasources/req\_ack 身份验证消息回应datasources/{s\_id}/{g\_id}/cmd 云平台指令datasources/req\_ack 校时响应