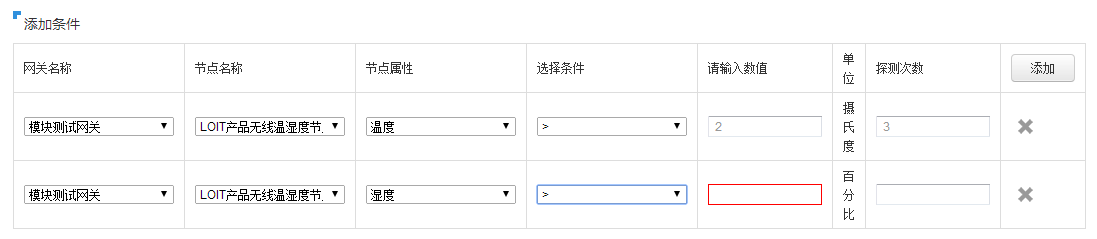
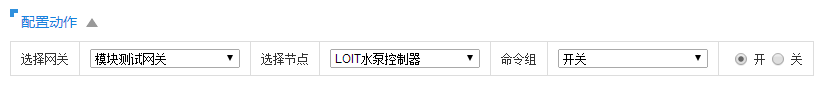
**Ifttt服务器端初步设计方案**

## 业务

IF



THEN



从后端数据角度看，一个ifttt可以分解为三大部分：（1）if部分的组成“项”，每项就是对应一个feedId、stream\_id对或场景等；（2）if部分各项之间的关系（与、或、非）；（3）then部分的组成“项”，每项就是对应一个feedId、stream\_id对或发送短信、邮件等。

## 架构

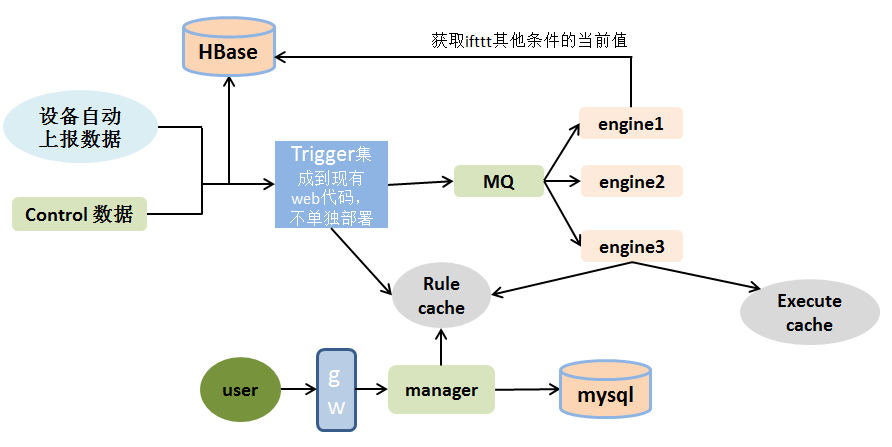


图1

**触发流程**

如图1所示，设备自动上报的数据或控制命令数据在存储到hbase后，再由trigger模块做一次ifttt检查此次数据是否涉及到ifttt，如果是，则封装任务消息并发送到MQ，MQ将任务消息实时推送到engine部分，engine再根据ifttt的相关条件判断是否要触发此次ifttt。

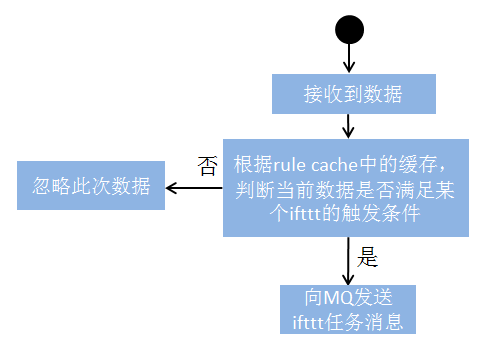
**模块详解**

**Trigger：**

对设备上报的数据与控制命令数据进行ifttt判断过滤，如果满足某个ifttt条件设置，向MQ发送ifttt任务通知；

此模块不单独部署，集成到现有设备上报数据与控制命令数据存储web代码中。

内部流程图：



**MQ：**

接收trigger的ifttt任务消息并向engine模块分发任务；

引入MQ的原因是达到设备数据上传、控制指令与ifttt执行的异步化，防止ifttt子系统出现问题而阻塞正常的设备控制、上传数据等基本功能的使用。

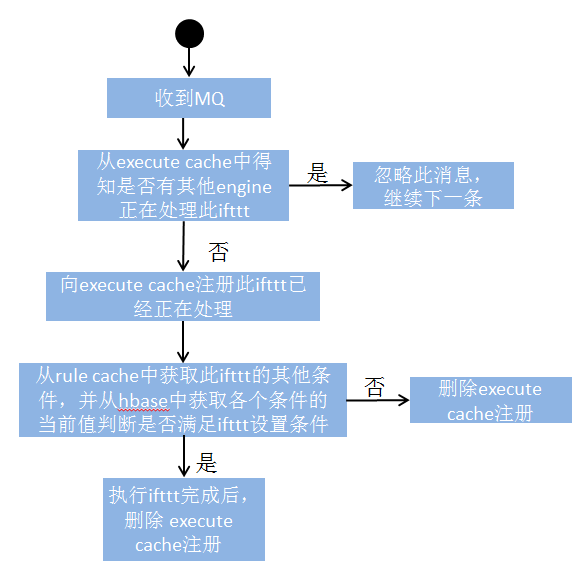
准备使用公司的MQ服务，目前订单系统也在使用，可靠性、并发性等都能满足我们的需求。

**Engine：**

接收MQ分发的ifttt任务消息，并对任务的其他条件进行判断，是否需要执行ifttt，如果是，进行命令执行。

此模块无状态，可以分布式部署、横向线性扩展来提高系统处理能力；

内部流程图：



**Manage：**

接收用户的ifttt设置，并进行ifttt的判断（比如是否会造成死循环、是否不满足一些要求等判断），如果正确，则存储到mysql与rule cache中。

**Rule cache：**

存储ifttt的设置，满足trigger、engine的查询；

准备使用公司的分布式缓存，引用缓存可以提高系统的处理速度与最大并发数，减小trigger、engine等模块的设计复杂度。

**Execute cache：**

由engine模块使用，当ifttt的if条件是多条件时，由于MQ可能将同一个ifttt的不同条件分发到不同的engine上去，为了防止一次条件的满足造成多次控制命令的发送，execute cache相当于一个简单的分布式锁，减小这种情况发生。

**各个模块压力分析**

设备总数：千万级

平均每个设备上传数据间隔时间：100s及其以上（2min）

Trigger对rule cache的访问tps会达到：10万级别

MQ：由于满足ifttt条件的情况较少，因此估算 千/万级别

Engine对rule cache访问tps：千/万级别

Engine对execute cache访问tps：千/万级别

Engine对hbase访问tps：千/万级别

Hbase 性能：单节点读tps 3000，写tps 20000（目前，我们服务预分9个节点）；

Cache：纯内存式，单分片get/set tps 10万；且根据缓存云人员反馈，其服务的可用性可以达到mysql的可用性。

**优点：**

1. 实时性高，表现在两方面：用户设置后马上生效；收到满足if条件的数据后马上可以触发ifttt。
2. 由if条件触发，避免所有ifttt检测。

**风险点：**

1、Cache挂掉后，初期trigger与engine可以直接访问mysql，后期设备访问量增大后（trigger、engine的访问tps超过mysql的负载）：每次控制命令的数据都从mysql中进行ifttt判断（原因：这种并发量一般不会很大），设备直接上传数据只做部分从mysql中进行ifttt判断，部分不做ifttt检查（原因：这种访问量很大，且即使这次不进行判断，隔一定时间间隔后，数据又会到达服务器就可能进行ifttt检查并触发，因此只是延迟了ifttt的触发时间）。

需要一个额外恢复cache的代码，可以整合到后台，不用单独部署。

1. execute cache只是一个简单的相当于分布式锁功能，有可能出现多条件if

情况下，两个及其以上if部分的条件几乎同时满足并发送到不同engine上去，而执行两次及其以上的控制命令。

但是这种概率极低，因为如果多个条件是同一个设备上报（同一设备一般一次会将自己所涉及的所有stream上报），此时MQ中只会产生这个ifttt的一个任务消息，相当于单条件；如果多条件是多个设备上报数据，由于每个engine拿到任务后会第一时间在execute cache中注册（整个过程时间不会超过100ms），所以只有不同设备上报数据的时间差恰好在100ms以内才有可能造成。

如果业务上必须要保证100%不会发生，就要采用分布式锁进行，但这样会降低系统的处理能力。