一种以完全的数据驱动的工作流设计

当前商业化的流程控制设计，一般是，将服务组合进行设计，形成协议文件资源，比如BPMN2.0、DMN等标准协议文件，通过流程引擎解析当前的流程设计文件，启动流程，控制流程的跳转，由引擎来托管整个流程的流转，并负责持久化流程状态，外部应用通过流程引擎API完成与流程的通信和控制。

流程节点的执行由流程引擎进行控制和通知。

基于MQ消息流驱动的工作流设计，相对更轻量，也只是为了解决特定领域的问题，一定程度上来说，不是以业界流程设计标准协议来驱动的。

主要的区别是，所有的流程节点都抽象为服务，相比BPMN2.0标准协议，它没有Gateway网关、Timer定时器等标准的控件。

1、所有的节点都是服务；

2、工作流以服务组合，并以一定的协议设计的形式来描述，；

3、每个服务通过订阅自己感兴趣的topic来进行服务，完成工作之后将自己的工作状态写进消息里，然后投递给消息broker topic router，router完成路由解析之后投递给broker；

4、broker完成状态的解析和产生新的消息，然后通过将消息投递给service topic router，router通过service topic集群的消息投递到对应的topic，有对应的topic服务器完成消费，再回到流程3，继续循环。

5、broker使用parser解析状态，如果发现协议描述工作流已经完成和所有的消息都工作完成，此时将不再将消息投递给router，直接将消息投递给terminal节点，完成所有的工作流。

2017年6月19日09:44:28

**FlowProtocol协议设计要点**

1、工作流id，以区分唯一的工作流实例；

2、当前工作流的完整描述（以某种组合的形式，或以二进制格式描述）；

3、工作流启动时间，完成时间，以BI后续挖掘使用；

4、当前工作流完成的每个服务节点的情况；

5、每个服务节点开始时间，完成时间；

6、每个节点输入、输出数据的资源描述（可以放在消息本身，也可以抽象为一个资源地址，比如一个restful资源地址，一个id，一个redis的key等等）；

7、服务节点之间的数据依赖，global域，scope域（参考activiti变量设计，以key-val形式来设置，读取）

8、提供每个服务节点的服务重试次数配置，支持某些服务可能需要重试的场景；

其他要点：

**Exception Node**

从运维和系统可用性的角度考虑，引入Exception Node的概念，将一些异常的流程（比如超过重试次数，数据依赖不正确等），统一由Exception Node来处理，落地，监控，报警，数据修复，重启等等。

**Validation**

工作流描述协议的验证

**Status Persistent and Query**

考虑到流程状态的查询是高频的使用场景，在Start Node，Terminal Node，Exception Node这些处理节点中，需要落地至数据库和NoSQL(如redis)，不同于一般的工作流引擎的设计思路，因为工作流协议信息本身就包含整个工作流状态，因此可以考虑异步落地数据库和缓存，提升性能。所有过来查询状态的请求，先从redis查询，找不到则从数据库拉取，再查不到，则报异常。

**Service Nodes & Broker Nodes & XX Router Load Balance**

统一以接口的形式设计:

Service接口(考虑同步异步的接口设计，AsyncService，参考Executer接口的设计)

Broker接口

Router Load Balance Policy接口设计(路由策略由外部注入，可以是Ribbon或任何的提供软件负载策略的设计)

**Task Parse**

使用64位长整型，描述工作流配置情况：

比如0111表示工作流的配置情况，很明显有3个节点；

比如0100表示已完成了一个几点的当前工作流情况；

使用0111 & 0100 = 0100表示已完成的工作流情况，如果0111 & xxxx = 0111自身，表示工作已经执行完毕，后续将状态flush至数据库或NoSQL缓存；

将0111 | 0100 = 0011，表示工作流已经完成至0011最后两个节点；

将0011与当前节点0010做 0011 & 0010 = 0010等于自己0010，即此时表示当前节点可以执行；

