# 一种以数据驱动的工作流设计要点

当前商业化的流程控制设计，一般是，通过提供的工作流设计工具，提供标准的控件，进行任务的控制或组合完成工作流编排设计，形成协议文件资源，比如基于BPMN2.0、DMN等标准协议文件，类似的产品有Activiti，jbpm等开源软件。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。

目前的开源流程引擎主要有以下特点：

1、本质是一个基于数据库存储信息的有限状态机；

2、流程节点的执行由流程引擎进行控制和通知，工作流由流程引擎托管；

3、基于标准的BPMN2.0规范，支持丰富的应用场景；

4、提供友好的设计工具，丰富的标准控件；

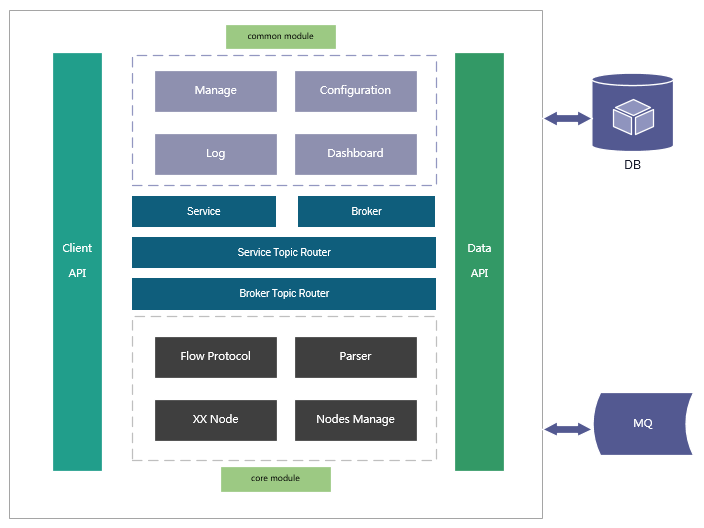
对于高并发，分布式的环境：

1、基于数据库状态的流程引擎，关系型数据库可能会是瓶颈之一；

2、难以分布式部署，水平扩展；

3、基于流程引擎的工作流驱动，业务吞吐量依赖流程引擎数量，不易提高吞吐量；

## 产品架构



#### Client API

以一定形式封装提供对外使用的API，可能是JAR，RPC或Restful形式的，建议设计为Restful形式的API，对外提供服务：

1. Start流程启动
2. Query流程状态查询
3. Callback服务节点回调
4. Cancel 流程终止
5. 其他服务等

#### Common Module

通用模块，主要是以产品的运维和使用的角度出发的，包括：

1. Manage 应该提供包括服务节点，代理节点的管理，参数管理
2. Configuration 参数配置，地址配置，topic配置等等
3. Log 日志，异常流程处理等等
4. Dashboard 统计和分析，供BI数据挖掘，包括某个流程某个节点使用了多长时间等等

#### Service

这里的Service泛指在工作流里的节点，是完成工作流服务的实体资源，可以是封装外部服务的节点，也可以直接是外部直接引用我们提供JAR包开发服务的节点。

Service节点在生产上支持分布式集群部署。

#### Broker

由于所有的service节点只关心完成自己的工作，不需要关心服务的投递和流程状态的解析，因此由于工作流状态和服务信息的路由投递工作将由Broker来完成，流程信息解析完成之后落地，需要将状态异步写数据库，异步写NoSQL缓存，写数据库主要是为了数据恢复和统计分析，写NoSQL主要是为了流程状态的查询。

所有broker的功能都是一致的，但是需要多节点部署，提高吞吐量和高可用。

支持分布式集群部署。

#### Service Topic Router

服务节点topic路由器，由于每个服务节点只关心自己提取消息、解析消息和消费消息，它不应该关心它将自己完成服务之后消息的路由投递，消息的路由投递，首先从broker对消息进行进一步解析，解析完成之后，交给service topic router，由它去完成消息的准确投递，有了service topic router，就可以天然的支持分布式服务节点的扩展和部署。

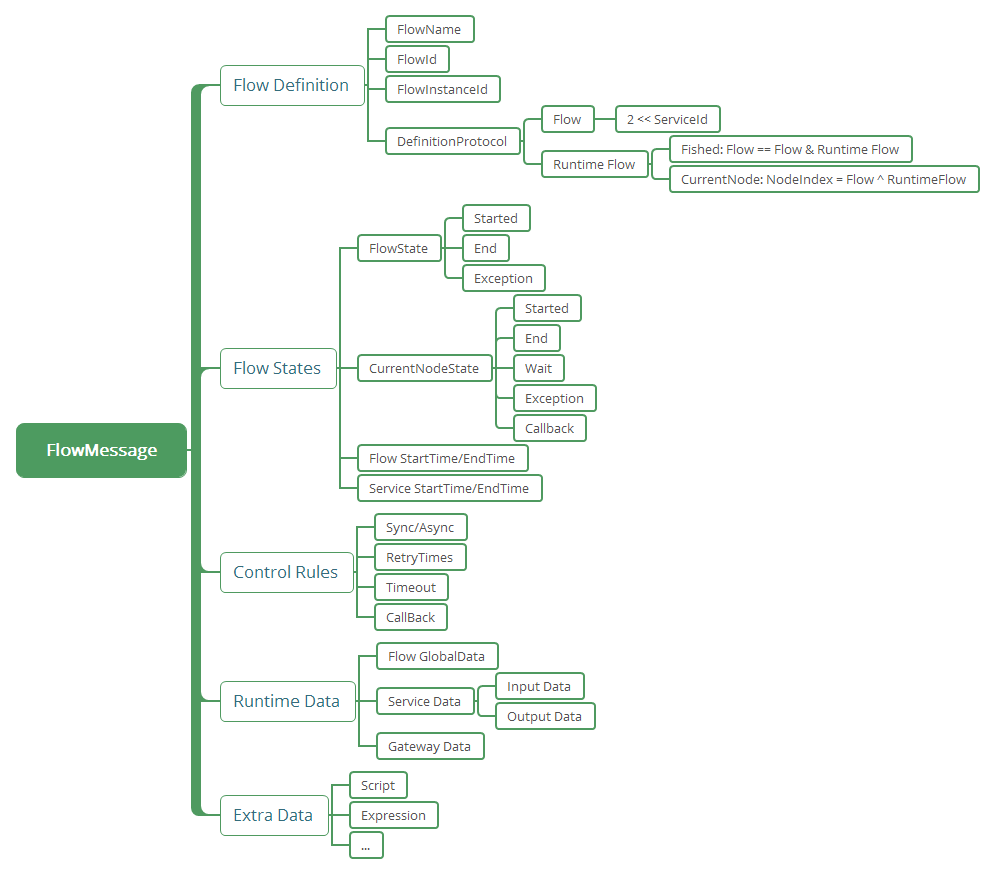
#### Broker Topic Router

原理跟service topic router一样，都是完成对broker的topic路由，这里的broker，功能都是一样，解析消息和完成流程的状态落地（异步写数据库、写NoSQL缓存），这里需要broker topic router主要是为了集群部署，提高集群的吞吐量和高可用。

#### Core Module

**Flow Protocol 工作协议设计**

主要数据和字段如下所示：



**Parse**

流程状态解析，根据流程消息协议，解析当前流程的状态，修改状态信息

**XX Nodes**

这部分节点主要是为了协助流程完成特定的工作，职能有所区分

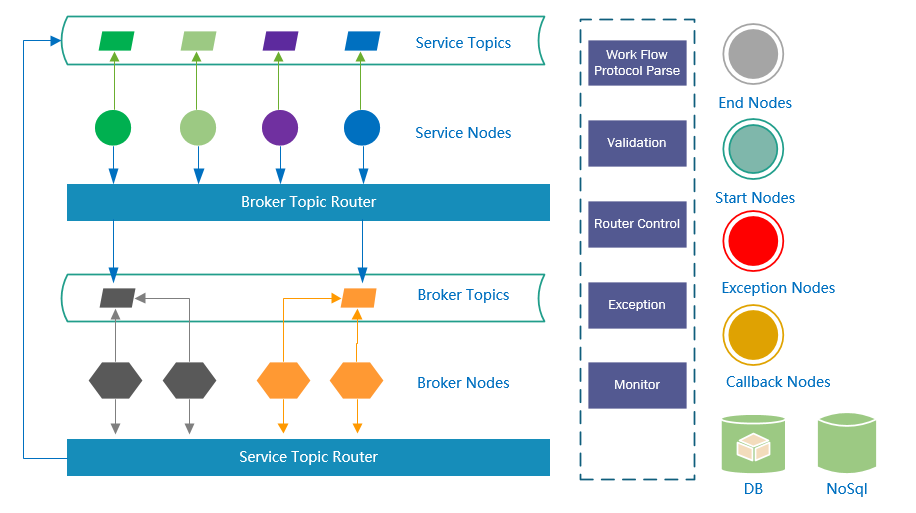
主要有这些类型的Nodes：

1. Start Nodes 开始节点，流程启动的入口
2. End Nodes 流程正常结束节点，所有完成的流程作业统一由End Nodes完成状态的落地
3. Exception Nodes 异常节点，service出现异常，状态将flush至异常节点，并适当报警以及在监控平台实时展示
4. Callback Nodes 回调节点，部分service需要外部的回调来完成工作流，回调节点将统一接受外部的回调和状态的解析

**Nodes Manage**

这里主要是指service节点和broker节点的管理，包括节点本身的注册、注销，对应监听topic的管理等等。因为这部分信息主要是供router进行Load Balance负载均衡使用的，因此节点或者topic信息的管理建议基于zookeeper来做注册和发现机制。

## 工作原理简述



基于数据驱动的工作流设计，相对更轻量，但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、所有的节点都抽象为服务；

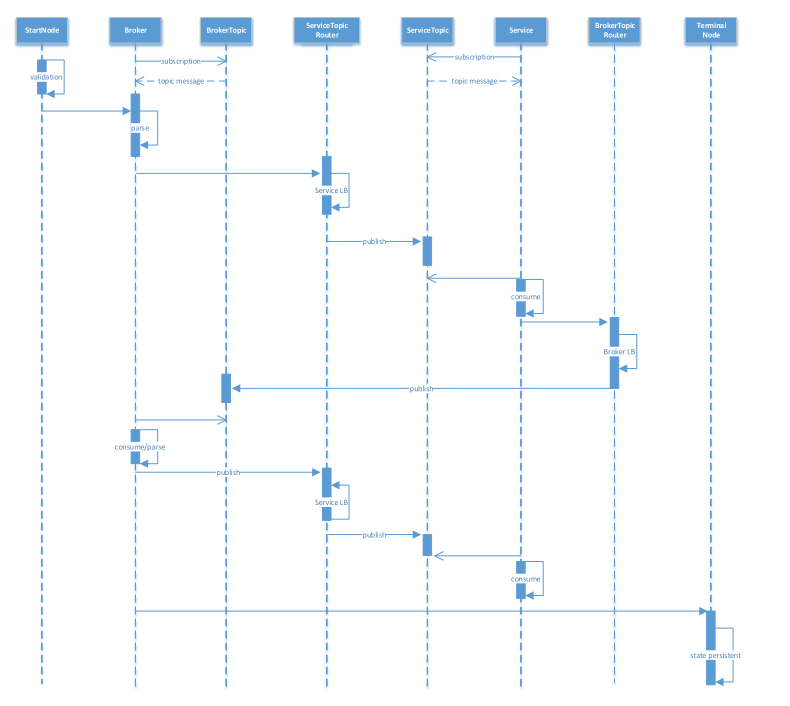
2、工作流以服务组合，并以一定的协议设计的形式来描述，；

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，对应的topic服务器完成消费，再回到流程3，继续循环，直至完成所有定义的节点；

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；

#### 核心流程时序图：



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；

9、timeout超时机制；

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；