uncode-schedule

oschina:http://git.oschina.net/uncode/uncode-schedule

Zookeeper目录结构

```
1.默认根路径：/uncode/schedule

2.服务器节点：/uncode/schedule/server 记录当前存活的服务器节点
节点名称格式：ip$uuid$序列化id
示例：
    [zk: localhost:2181(CONNECTED) 7] ls /uncode/schedule/server
    [10.242.128.90$7BDA7A9C4B6D4F829BDA56A1E109E073$0000000008]
节点内容：
    {
        "uuid": "10.242.128.90$7BDA7A9C4B6D4F829BDA56A1E109E073$0000000008",//ip$uuid$序列化id的格式
        "ip": "10.242.128.90",
        "hostName": "wangshenweideMacBook-Pro.local",
        "registerTime": "2016-01-13 21:15:51",//客户端向ZK注册的时间
        "heartBeatTime": "2016-01-13 21:20:30",//客户端向ZK发送心跳的最后时间
        "dealInfoDesc": "Zookeeper connecting ......127.0.0.1:2181",//描述信息
        "version": 139,
        "isRegister": true
    }

3.任务列表节点： /uncode/schedule/task 记录需要执行的定时任务的列表
节点名称格式：beanName$methodName
这里的beanName就是spring-task定义的定时任务的bean的名称,methodName就是方法名
对于haitao里面的JobServiceImpl来说beanName就是jobService，methodName对于的是方法名称，如果有多个任务该节点就有多个子节点，这里就列了一个。
示例：
    [zk: localhost:2181(CONNECTED) 12] ls /uncode/schedule/task
    [jobService#doSchedule]

4.任务节点：/uncode/schedule/task/任务节点 记录当前执行该任务的机器的UUID
节点名称格式：ip$uuid$序列化id（即服务器节点的名称）
示例：
    [zk: localhost:2181(CONNECTED) 15] ls /uncode/schedule/task/jobService#doSchedule
    [10.242.128.90$7BDA7A9C4B6D4F829BDA56A1E109E073$0000000008]
    说明对于jobService的doSchedule这个任务当前由10.242.128.90这台机器上的某个进程执行。uuid是用来支持一台机器开启多个JVM的模式。
```

运行机制

```
1.初始化
    JVM启动时客户端向ZK注册客户端信息，在服务器节点（/uncode/schedule/server）下创建PERSISTENT_SEQUENTIAL类型的当前服务器节点，初始化注册时间，心跳时间等信息。
2.客户端心跳与任务的重新分配
    客户端初始化完成之后向ZK定时发送心跳信息（默认2秒），并跟新当前的心跳时间。
    每次心跳都会触发一次任务的重新分配，但是并不是每次重新分配都会导致任务的在服务器之间的迁移，后面会做说明。
    任务重新分配的过程：
    1.首先会检查服务器的存活情况，删除那些心跳超时的服务器节点
    2.在存活的服务器节点中选取一台服务器作为leader进行任务分配。（leader选取策略：序列化ID最小的即为leader）
    3.任务分配策略：
        *在任务列表节点找到所有的任务，逐一进行分配。
        *如果任务没有分配给任何服务器，则在存活的机器中随机选取一台分配。
        *如果任务已经分配了节点 并且该节点还存活就不进行重新分配，任务继续由该节点执行。
        *如果任务已经分配了节点，但是改节点已经心跳超时，则在存活的机器中随机选取一台分配。
3.任务执行
    当定时任务第一次执行是客户端会自动向ZK注册任务，再任务节点（/uncode/schedule/task）下生成一个节点，但是并不会立即执行，会等待任务的分配。
    每次定时任务的执行客户端都会向ZK判断是否该任务属于当前自己这个节点，如果不是则不执行。
4.异常处理
    当ZK不可用或者与zk集群网络不通的情况下，客户端会继续执行之前已经分配的任务，客户端本地会有分配给自己的任务缓存。
```

优点与缺点

优点：
1.基本原理和TBSchedule差不多，相对于TBSchedule更加轻量，只依赖ZK，不需要引入数据库配置　2.代码改动小，有黑名单机制，预发布的机器可以排除在任务分配之外。。
3.任务的failover保证机器不可用或者进程退出的情况下任务的重新分配。　缺点：
1.相对于TBSchedule配置不够简单，需要手动改ZK配置。
2.只有在之前分配节点不可用的情况下才会进行任务的重新分配，任务的分配不均衡，可能导致一台服务器的任务过多。
3.对ZK依赖较强，需要客户端启动时ZK必须可用，否则起不来。

建议线上修改过程

一些重要新相对比较低的定时任务可以先修改成分布式执行，待稳定之后再迁移别的定时任务。