定时器任务之Saturn

笔记本：	java		
创建时间：	2018\10\15 星期一 17:08	**更新时间：**	2018\10\18 星期四 14:55
作者：	804790605@qq.com		
URL：	https://vipshop.github.io/Saturn/#/zh-cn/3.0/quickstart		

定时器任务之快速使用Saturn--快速入门

一、简介

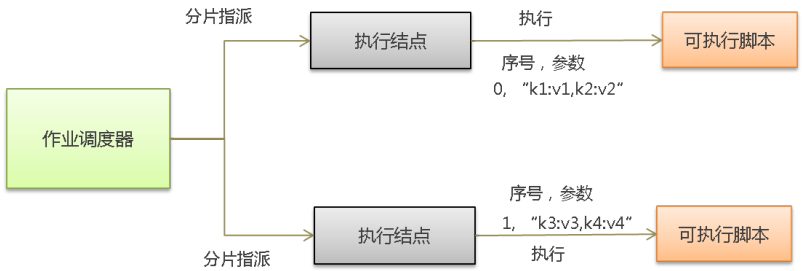
Saturn (定时任务调度系统)是唯品会自主研发的分布式的定时任务的调度平台，目标是取代传统的Linux Cron/Spring Batch Job/Quartz的方式，做到全域统一配置，统一监控，任务高可用以及分片。目前该平台已平稳运行1年，承载着唯品会核心系统的全部定时任务的调度，监控，配置，经受住了生产环境的各种考验。 开源版本系唯品会生产使用的saturn核心，去除了唯品会的认证，监控，告警系统等依赖，可独立部署安装使用。
基于当当的elastic-job 版本1开发，并且可以很好的部署到docker容器上。
Saturn包括两大部分，Saturn Console和Saturn Executor。
1、Saturn Console是一个GUI，用于作业/Executor管理，统计报表展现，系统配置等功能。它同时也是整个调度系统的大脑：将作业任务分配到各Executor。
2、Saturn Executor是执行任务的Worker：按照作业配置的要求去执行部署于Executor所在容器或物理机当中的作业脚本和代码。

二、常见术语

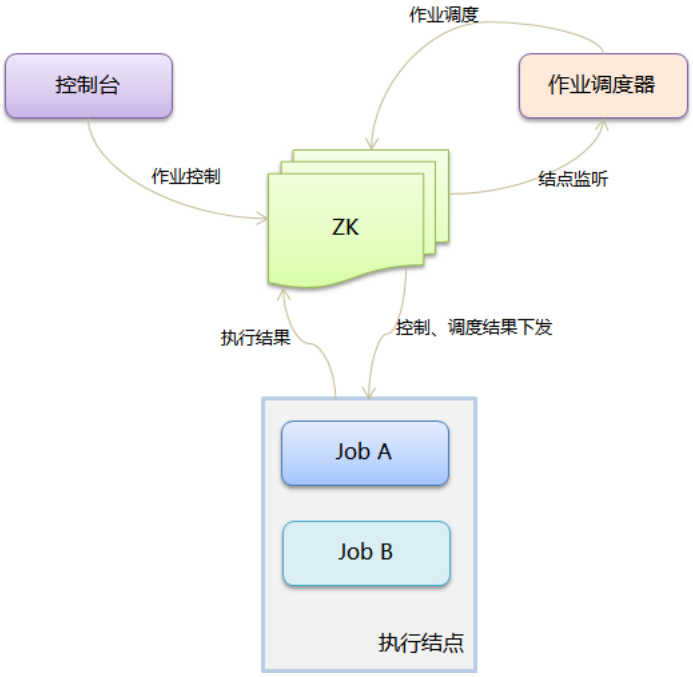
术语	解释
作业(Job)和作业分片	作业(Job)是可以独立运行的脚本(shell作业)或者具备某项功能的函数实现(java、消息作业)。作业可并发执行在多个执行节点(Executor)上，作业分片定义了作业并发执行的数量以及执行编号。比如作业分片为2，则表示作业最多可以并发执行在2个执行结点上，执行编号为0和1。作业分片调度指将作业分片指派给执行结点。比如A作业有5个分片（分片0，1，2，3，4），一共有两个执行结点E1和E2，将0，1，2指派给E1；3，4指派给E2。
Namespace	域（saturn称之为Namespace)代表一组特定的执行结点和作业。作业必须而且只能属于某一个特定的域。一个域下通常有若干个执行结点（每个执行结点都运行在不同的独立的机器上），通过启动参数指定执行结点的-namespace将执行结点加入到某个域。执行结点必须而且只能属于某一个特定的域。域下的全部执行结点功能是对等的，域下的任何一下执行结点均有能力执行域下的全部作业，换言之，作业可以在域下任何一个执行结点执行。
组织名	每个namespace可以属于一个组织
Saturn	定时任务调度系统
执行结点(Executor)	执行结点(Executor)是调用并执行作业的程序。它通过定时(quartz)驱动来触发调用事件，并最终调用作业的执行入口(shell脚本或者函数实现)。执行结点只会处理指派给它的作业分片。
控制台(Console)	统一配置界面，可以使用控制台来查看作业状态，执行结点状态和执行日志，添加、删除作业，修改作业属性。

三、总体设计说明

Saturn的基本原理是将作业在逻辑上划分为若干个作业分片，通过作业分片调度器将作业分片指派给特定的执行结点。 执行结点通过quartz触发执行作业的具体实现（以shell为例，则为shell脚本），在执行的时候，会将分片序号和参数作为参数传入(见图1)。 作业的实现逻辑需分析分片序号和分片参数，并以此为依据来调用具体的实现（比如一个批量处理数据库的作业，可以划分0号分片处理1-10号数据库，1号分片处理11-20号数据库）。
基本原理图如下：



系统逻辑架构图如下：



解释

执行结点：负责作业的触发（定时），作业执行，结果上报，日志上报，告警上报，监控日志写入等功能。可独立运行在业务服务器，也可与业务代码运行在同一个JVM。 使用java开发，提供jar包和可运行的工程两种方式供业务方使用，是业务作业接入saturn最主要的组件。

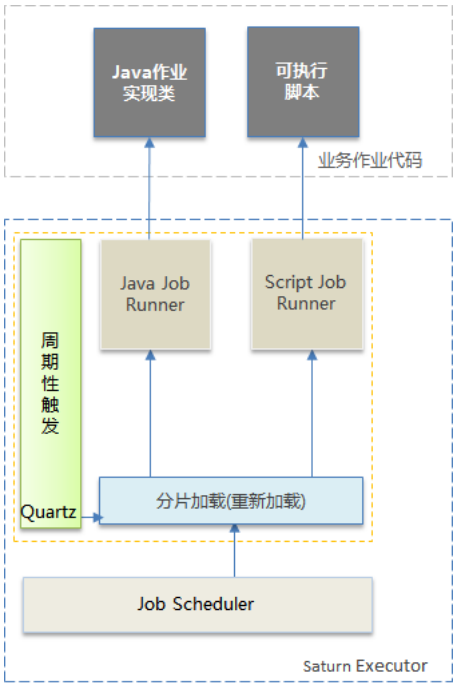
控制台：负责作业的统一配置，包括作业添加、删除，作业属性配置，作业状态查看，执行日志查看，执行结点监控等功能。 控制台单独部署，提供WEB应用给全域共用，业务接入方根据申请的权限控制对应的业务作业。

作业分片调度器：Saturn的”大脑“，其基本功能是将作业分片指派到执行结点。通过调整分配算法和分配策略，可以将作业合理地安排到合适的执行结点，从而实现HA，负载均衡，动态扩容，作业隔离，资源隔离等治理功能。 作业分片调度器为后台程序，单独部署；它是公共资源，所有域共用同一套作业分片调度器。 接入作业后，会自动接受作业分片调度器的调度。

四、模块划分

序号子系统	名称	功能描述
1	saturn-core	公共模块，定义公共类，方法及实现
2	saturn-console-core	saturn控制台公共模块
3	saturn-console	saturn控制台
4	saturn-executor	saturn执行结点
5	saturn-job-sharding	saturn作业分片调度器
6	saturn-job-embed	嵌入方式使用saturn的转换模块（把saturn嵌入其它系统中运行，比如tomcat)
7	saturn-plugin	saturn maven插件
8	saturn-it	saturn 集成测试

Saturn执行节点



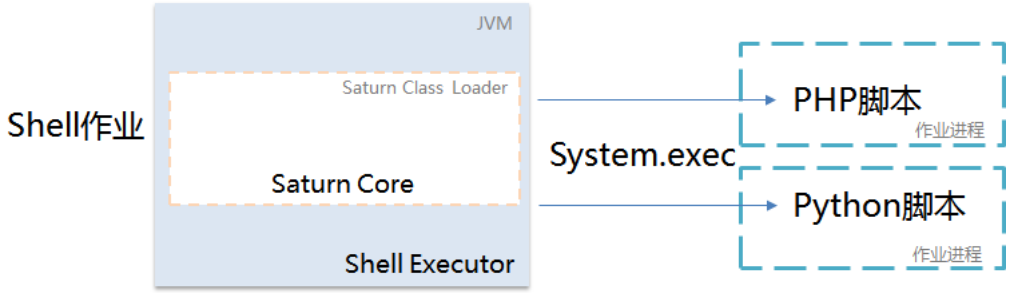
解释：

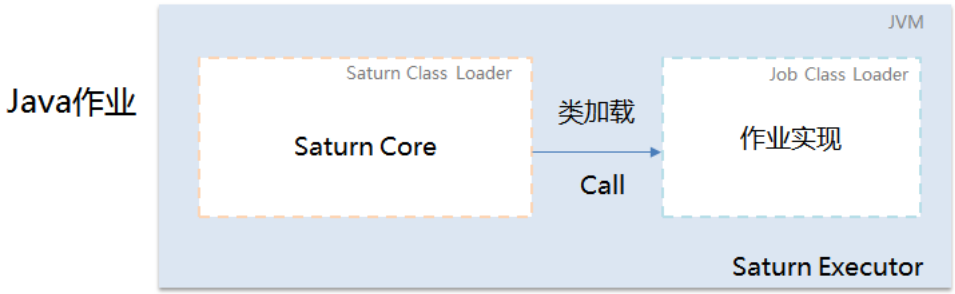
1、Job Scheduler：作业处理基础类，负责从ZK中读取作业配置信息（比如作业类型【shell作业/java作业/msg作业】，作业cron表达式，分片参数等），根据作业类型启动不同的处理逻辑。 shell作业和java作业使用cron表达式开启quartz scheduler，并周期性触发；msg作业启动分片监听，并根据获取到的分片启动消息订阅。

2、分片加载（重新加载）：检查有没有收到作业分片调度器发出的重新加载指令(通过ZK结点)，如果有，则全部执行结点都须等待全部分片执行完成，然后由其中一个执行结点从任务调度器的调结果中读取本作业的分片指派结果， 并将结果广播（使用ZK的监听机制）给本作业的全部执行结点。

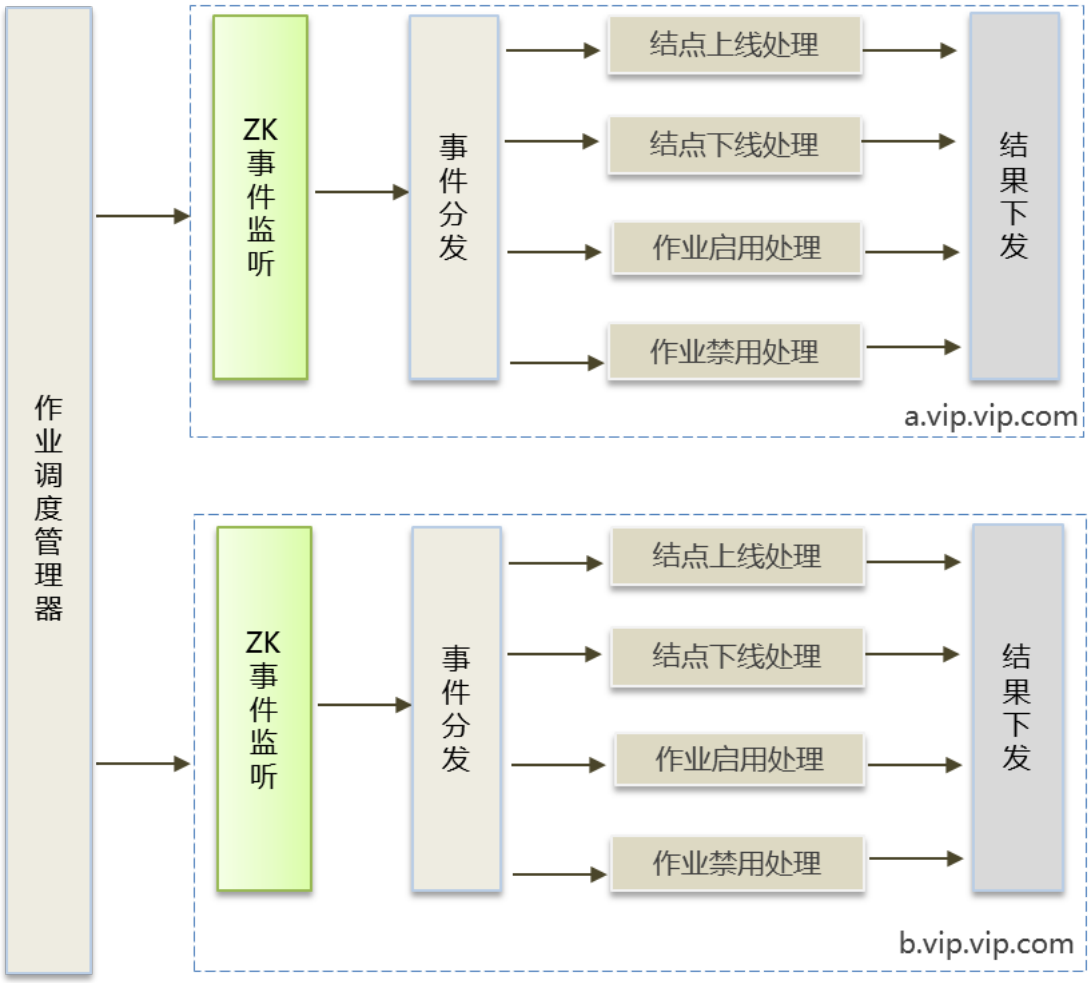
3、Shell/Java作业触发：Shell作业和Java作业根据cron表达式定义的时间规则进行周期性触发，每次触发周期到达时，先进行分片加载（重新加载），之后根据作业类型调用Java Job Runner或Script Job Runner。 Job Runner取出分片序列，解析分片参数（分片参数可通过控制台配置）； 如果是java作业，启动线程池，将分片序列和分片参数作为调用参数构造作业执行线程，将作业执行线程提交到线程池执行； 如果是script作业，将分片序列和分片参数作为调用参数exec调用可执行脚本，并获取输出流。 Job Runner启动作业之后，如果作业配置了超时时间，则需进行超时检测，一旦检测到超时，需要做超时处理（作业中止，状态上报）。

作业交互图如下：





作业分片调度器逻辑图如下：



解释：

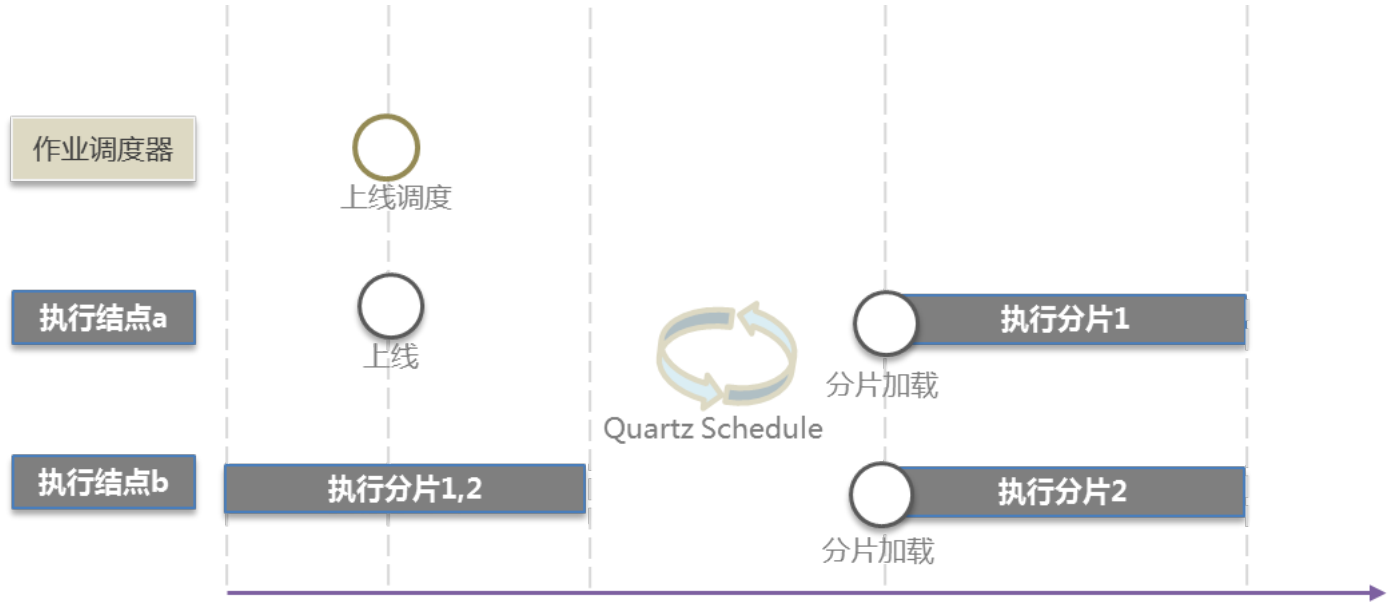
1、作业分片调度管理器：遍历全部域，为每个域都生成域作业分片调度器。域作业分片调度器的处理步骤为：ZK事件监听，事件分发和处理，结果下发。
2、ZK事件监听：执行结点上线后会在ZK注册一个临时结点，下线后ZK会话超时临时结点会被清除。 作业启用和禁用状态为持久化结点，由管理员通过控制台修改（从启用改为禁用，或者从禁用改为启用）。 Saturn作业分片调度器监听这些结点的变化，根据结点路径和结点的值判断出事件类型进行事件分发。
3、事件分发：Saturn作业分片调度器处理以下4类事件：结点上线，结点下线，作业启用，作业禁用。 分别对应结点上线处理类，结点下线处理类，作业启用处理类，作业禁用处理类四个模块。 事件处理类的作用是进行作业分片调度，将作业分片指派给执行结点。并将指派结果保存到ZK。
4、结果下发：将重新加载指令写入到域下全部作业的指定的ZK结点。

六、功能描述

1、多种作业类型(Java作业/Shell作业)：Saturn executor使用java语言开发，Java作业天然支持，接入作业只需实现特定的作业执行方法即可接受调度；对于Shell作业，使用apache common exec调用Shell执行脚本，并获取输出流，使用timeout系统命令在超时后将脚本进程退出。
2、动态扩容，HA，负载均衡：Saturn通过作业分片调度器和作业分片调度算法在资源改变时控制作业分片的指派； 资源改变包括：执行结点增加（扩容），执行结点减少（减容），作业上线（启用） 执行结点上线（增加、扩容）时，作业分片调度器会根据平衡算法将现有执行结点上的部分作业分片迁移到新增加的执行结点，作业上线（启用）时，作业分片调度器会根据平衡算法将新上线作业的全部分片分配给域下的执行结点，从而实现动态扩容和负载均衡。 执行结点下线（减少、减容）时，作业调度器会根据平衡算法将被减容（下线）的执行结点的全部作业分片迁移到其它存活的执行结点，从而实现HA；
3、失败转移：执行结点下线时，如果有作业正在执行且尚未结束，则正在执行的作业分片会被作业调度器指定到其它执行结点并且尝试立刻执行，如果无法立刻执行，则会等到当前执行结点分配到的本作业的其它分片执行完毕后立刻执行。 注意：saturn的失败转移机制并不保证分片迁移到新的执行结点后立刻开始执行，它需等待当前执行结点分配到的本作业的其它分片执行完毕后才会立刻执行。如果作业执行周期较长，就会存在一段相当长的时间内，此分片没有分配到执行结点。
4、Job Timeout处理：设置了timeout的作业开始执行后会启动超时检测，如果执行超时，则会停止当前作业的执行（JAVA作业停止线程，Script作业停止进程），并将超时事件上报。
5、资源隔离：有时候需要对执行结点进行功能划分，比如1，2号执行结点处理作业A，B；3，4号作业处理作业 C,D； 设置作业的优先列表，当优先列表结点存活时，作业的分片只会指派给这些结点；当优先结点全部下线时，作业的分片才会指派给其它结点。
6、作业隔离：某些作业不希望与其它作业共用同一个执行结点，与其它作业隔离。 将需要隔离的作业的负载值设为远超其它作业的值（比如本作业负载设置为999，其它作业负载1），作业调度器在进行作业调度时会尽量保证执行结点的负载均衡。 负载值大的分片优先分配，而且一旦分配到某个执行结点，将会导致该执行结点的负载值远超其它结点，从而保证不会有其他作业被分配到该执行结点，于是可以达到作业隔离的目的。
7、统一配置：Saturn通过统一的控制台进行全部域及全部作业的配置，执行情况监控，结点监控等。

七、关键流程

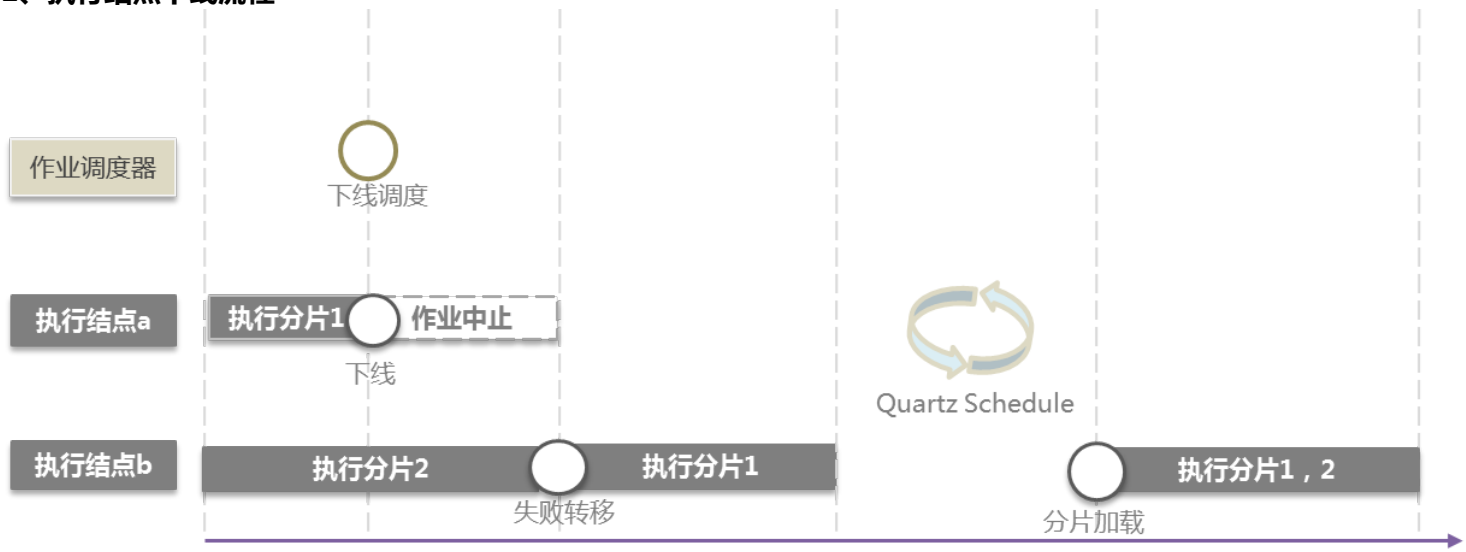
1、执行结点上线流程



步骤说明：

名称	说明
上线	执行结点启动，往ZK临时结点/a.vip.vip.com/$SaturnExecutors/executors/excutor_001/ip写入本机IP 。
上线调度	作业分片调度器监听ZK路径/a.vip.vip.com/$SaturnExecutors/executors/，当路径为/ip并且事件为ADD，则判断出有结点新上线；作业分片调度器根据算法进行分片调度，并将度结果写入/a.vip.vip.com/$SaturnExecutors/sharding/content结点；作业分片调度器将重新加载指令(随机数字串，大于0)写入全部作业的/a.vip.vip.com/$Jobs/job_01(这里为jobName)/leader/sharding/necessary结点；
分片加载	执行结点从/a.vip.vip.com/$Jobs/job_01(这里为jobName)/leader/sharding/necessary结点读取重新加载指令，如果大于0，则重新加载分片；作业的全部执行结点会选举出一个leader，并由leader进行分片重载，过程如下：从/a.vip.vip.com/$SaturnExecutors/sharding/content结点读取最新的分片信息,找出属于本作业的各个执行结点分片信息，并写入/a.vip.vip.com/$Jobs/job_01/servers/executor_001(执行结点名称)/sharding结点。

2、执行结点下线流程



步骤说明：

名称	说明
下线	执行结点关闭，临时结点/a.vip.vip.com/$SaturnExecutors/executors/excutor_001/ip自动清除；
下线调度	作业分片调度器监听ZK路径/a.vip.vip.com/$SaturnExecutors/executors/，当路径为/ip并且事件为REMOVE，则判断出有结点新下线；作业分片调度器根据算法进行分片调度，并将度结果写入/a.vip.vip.com/$SaturnExecutors/sharding/content结点；作业分片调度器将重新加载指令(随机数字串，大于0)写入全部作业的/a.vip.vip.com/$Jobs/job_01(这里为jobName)/leader/sharding/necessary结点；
失败转移	执行结点监听/a.vip.vip.com/$Jobs/job_01/execution/0/running结点，当事件为REMOVE，表示有分片执行中止，而且尚未完成执行，需要进行失败转移；需要失败转移的分片会在/a.vip.vip.com/$Jobs/job_01/leader/failover/items/下面建一个子结点；当作业完成本次执行后，从/a.vip.vip.com/$Jobs/job_01/leader/failover/items/下面摘取（获取并删除）一个结点并立刻执行。
分片加载	执行结点从/a.vip.vip.com/$Jobs/job_01(这里为jobName)/leader/sharding/necessary结点读取重新加载指令，如果大于0，则重新加载分片；作业的全部执行结点会选举出一个leader，并由leader进行分片重载，过程如下：从/a.vip.vip.com/$SaturnExecutors/sharding/content结点读取最

	新的分片信息，找出属于本作业的各个执行结点分片信息，并写入/a.vip.vip.com/$Jobs/job_01/servers/executor_001(执行结点名称)/sharding结点

3、 作业状态转换图

八、Saturn使用demo

1. 确定saturn console能否访问 ip+:9088 例如：192.168.1.xxx:9088
2. 在pom.xml添加dependency

```
<dependency>
  <groupId>com.vip.saturn</groupId>
  <artifactId>saturn-job-api</artifactId>
  <!-- 修改成指定版本 -->
  <version>3.2.0</version>
</dependency>

<!-- 以及插件 -->
<plugin>
  <groupId>com.vip.saturn</groupId>
  <artifactId>saturn-plugin</artifactId>
  <!-- 版本与saturn-job-api一致 -->
  <version>3.2.0</version>
</plugin>
```

注意：3.2.0版本还不是Saturn最新版本，可根据自己需求选择不同的版本。

3.编写相关代码

在创建Job类之前要创建工厂来加载类。MySpringApplicationContext.java
（当然如果仅仅只是先测试运行的话，可以直接创建一个job类，然后进行调用测试）

```
public class MySpringApplicationContext extends AbstractXmlApplicationContext {

    private Resource[] configResources;

    public MySpringApplicationContext(Resource[] configResources) throws BeansException {
        this(configResources, true, null, null);
    }

    public MySpringApplicationContext(Resource[] configResources, ClassLoader classLoader) throws BeansException {
        this(configResources, true, null, classLoader);
    }

    public MySpringApplicationContext(Resource[] configResources, boolean refresh, ApplicationContext parent,
                                      ClassLoader classLoader) throws BeansException {
        super(parent);
        if (classLoader != null) {
            this.setClassLoader(classLoader);
        }
```

```
        this.configResources = configResources;
        if (refresh) {
            refresh();
        }
        // 注册关闭钩子
        registerShutdownHook();
    }

    protected Resource[] getConfigResources() {
        return this.configResources;
    }

}
```

SpringFactory.java

```
public class SpringFactory {

        private static final String APPLICATION_CONTEXT_ROOT = "spring/applicationContext-root.xml";
        private static final String APPLICATION_CONTEXT_MQ = "spring/applicationContext-mq.xml";
        private static final String APPLICATION_CONTEXT_PERSISTENCE = "spring/applicationContext-persistence.xml";

        private static SpringFactory instance = new SpringFactory();

        public static SpringFactory getInstance() {
                return instance;
        }

        private BeanFactory factory;

        public Object getObject(String beanId) {
                return factory.getBean(beanId);
        }

        private SpringFactory() {
                List<Resource> resources = new ArrayList<Resource>();

                resources.add(new ClassPathResource(APPLICATION_CONTEXT_ROOT));
                resources.add(new ClassPathResource(APPLICATION_CONTEXT_MQ));
                resources.add(new ClassPathResource(APPLICATION_CONTEXT_PERSISTENCE));

                Resource[] resourceArrays = new Resource[resources.size()];
                try {
                        ApplicationContext context = new MySpringApplicationContext(resources.toArray(resourceArrays));
                        factory = (BeanFactory) context;
                } catch (RuntimeException e) {
                        e.printStackTrace();
                        throw e;
                }
        }
}
```

SaturnTest.java

```
@Component
public class SaturnTest extends AbstractSaturnJavaJob {

        @Override
        public SaturnJobReturn handleJavaJob(String jobName, Integer shardItem, String shardParam, SaturnJobExecutionContext
shardingContext) {
                SaturnJobReturn saturnJobReturn = new SaturnJobReturn("分片:" + shardParam);
                try {
                        SaturnTest instance = (SaturnTest ) SpringFactory.getInstance().getObject("saturnTest");
                        instance.setSaturnApi(instance);
                } catch (Exception e) {
                        saturnJobReturn.setErrorGroup(500);
                        saturnJobReturn.setReturnMsg("*********数据异常" + e.getMessage());
                        // logger.error("**********取得数据异常", Throwables.getStackTraceAsString(e));
                }
                return saturnJobReturn;

        }
}
```

SaturnSystemErrorGroup.java

```
public final class SaturnSystemErrorGroup {

        public static final int SUCCESS = 200;
```

```
        // general fail
        public static final int FAIL = 500;
        public static final int TIMEOUT = 550;
        // alarm will be raised with this error code
        public static final int FAIL_NEED_RAISE_ALARM = 551;

        public static Set<Integer> getAllSystemErrorGroups(){
                Set<Integer> resultSet = new HashSet<>();
                resultSet.add(SUCCESS);
                resultSet.add(FAIL);
                resultSet.add(TIMEOUT);
                resultSet.add(FAIL_NEED_RAISE_ALARM);

                return resultSet;
        }
}
```

DemoJob.java

```
public class DemoJob extends AbstractSaturnJavaJob {
        int count = 0;
        @Override
        public SaturnJobReturn handleJavaJob(final String jobName, final Integer shardItem, final String shardParam, final
SaturnJobExecutionContext context) {
                System.out.println("job start time " + System.currentTimeMillis());
                // do what you want here ...
                System.out.println("times is " + count++);
                System.out.println("job end time " + System.currentTimeMillis());
                // 返回一个SaturnJobReturn对象，默认返回码是200表示正常的返回
                return new SaturnJobReturn("我是分片"+shardItem+"的处理结果");
        }
}
```

解释说明

*handleJavaJob*方法是作业调用主入口，当调度周期到达时，Saturn会调用该方法。

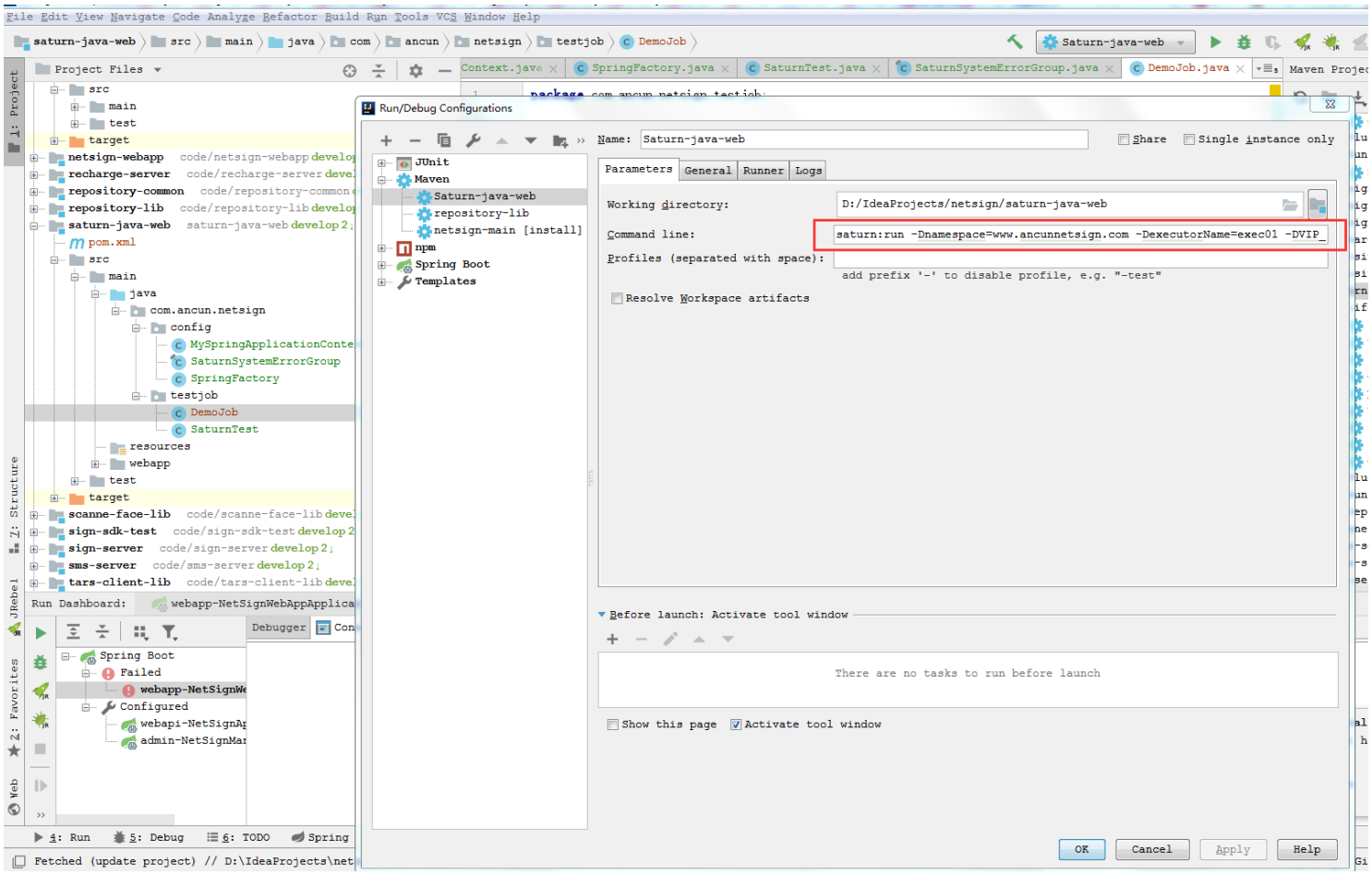
传入参数如下：

```
jobName：作业名
shardItem：分片编号（从0开始）分片参数（在Console配置）
shardParam：分片参数（在Console配置）
context：调用上下文
```

JavaJobReturn是作业结果返回的封装。里面三个成员变量，包括：

```
returnCode：结果码。0代表结果成功，其余值代表失败。默认为0。用户可以根据自己业务的情况设置返回值，但注意，如下返回码是保留字不能使用，包括：0，1，2，9999。
returnMsg：返回信息。将显示在Console里面。没有默认值。
errorGroup：异常码。详情参见教程。
```

4、IDEA调试：



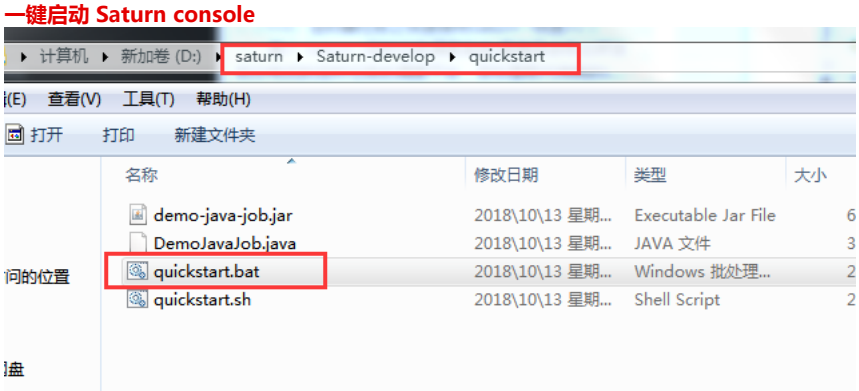
填上：saturn:run -Dnamespace=www.ancunnetsign.com（部署好的域名） -DexecutorName=exec01 -DVIP_SATURN_CONSOLE_URI=http://127.0.0.1:9088（部署机）

注：namespace要修改为使用的域名，URI要修改为正确的地址，executorName为执行节点名称，当然也可以在VM添加上： -Dspring.profiles.active=dev运行

- namespace： 命名空间。命名空间用于对作业进行分组，作业必须属于某个命名空间，同一个命名空间下的作业名不能重复。
- executorName：执行结点唯一标识

5、saturn console添加java作业

一键启动 Saturn console



首先，请确保本机安装了以下软件，当然在运行的时候回安装（第一次的话会很耗时间）：

```
JDK 7 or JDK 8
Maven 3.0.4+
node.js 8.7.0+
npm 5.4.2+
docker（版本不限）
```

然后，git clone本仓库到本地，checkout对应版本分支，进入quickstart目录。如果是Windows系统，请运行quickstart.bat，如果是Linux/Unix/MacOS系统，请运行quickstart.sh。

```
$ git clone https://github.com/vipshop/Saturn
$ git checkout develop
$ cd quickstart
$ chmod +x quickstart.sh
```

```
$ ./quickstart.sh
```

quickstart脚本将做如下事情：

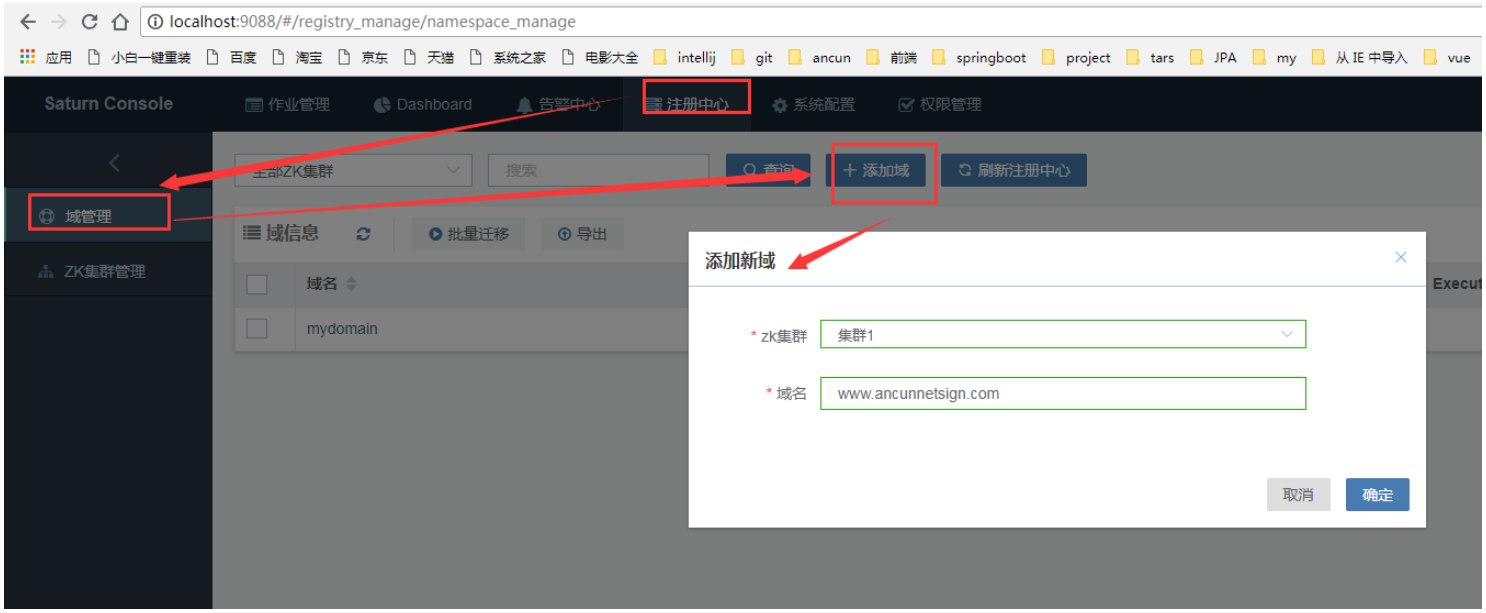
```
启动内嵌的ZooKeeper
启动内嵌的Saturn-Console
启动内嵌的Saturn-Executor（包含了一个Java作业的实现）
在Saturn-Console添加该Java作业
启动完成后，您可以访问Saturn-Console：http://localhost:9088
```

如果你见到如下界面，则恭喜你，你的console已经启动。

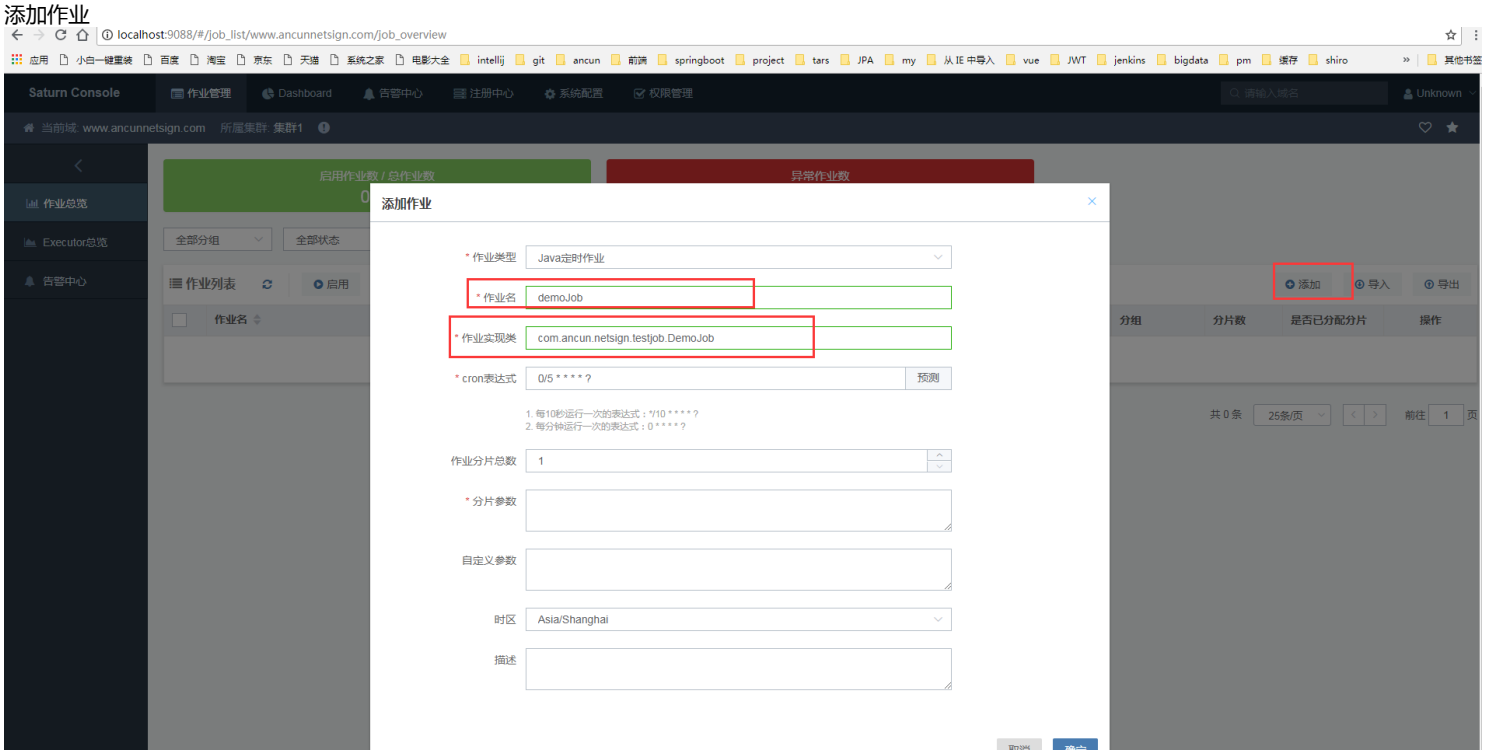
在首页的search bar点击会出现一个叫做'mydomain'的namespace。点击进去会见到一个名为'demoJavaJob'的作业，该作业有5个分片，每隔5秒调度一次。

一个叫做'executor-1'的executor执行器调度该作业。

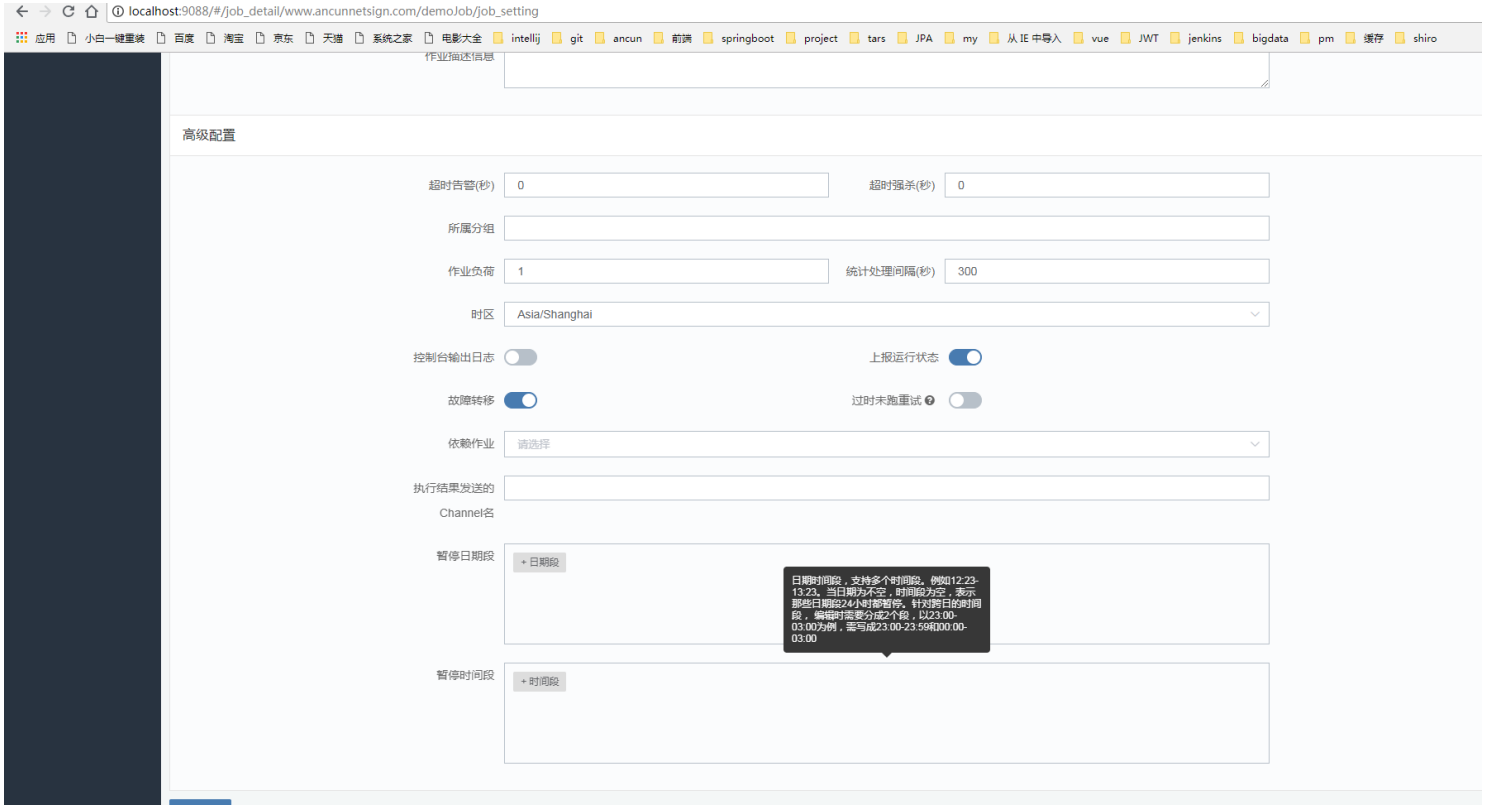
添加域



添加作业



作业类型： 分为Java定时作业和Shell定时作业，这里选择Java定时作业
作业名：作业ID标识，namespace下必须唯一
作业实现类：作业实现类的完整包名+类名
cron表达式：作业定时表达式（具体配置情况见《定时器任务之Quartz文档》）
作业分片总数：表示并发执行的数量，2代表该作业同时有两个进程在并发执行，每个进程都有自己专门的脚本和参数(这些进程可能同跑在不同机器上的)。
注：优先Executor：如果你想选择特定的物理机executor去运行你的作业，则需要设置优先executor。下拉框的候选项为当前域下的所有在线的executor。
只使用优先Executor：如果优先executor离线了，不会failover到其他机器，在本机测试作业的时候要勾选只使用优先executor。？
注：分片数可根据需求来定
分片序列号/参数对照表：定义每个分片执行的完整脚本路径及参数。这是saturn最重要的参数之一。
注：由于分片个数为1，所以分片参数只需要一个键值对（0=local01）key值为0，value：实际填写时要使用executorName作为value值，。
作业描述信息 ：作业描述
更多**高级的配置**，可以在作业编辑页面进行编辑（根据自身需要来定）



点击启动就可以看到控制台的运行结果了，如下图：

当然，如果需要把配置的情况保存下来，那么需要配置数据库。

参考资料：

git 官网 https://github.com/vipshop/Saturn
Saturn文档 https://vipshop.github.io/Saturn/#/zh-cn/3.0/quickstart

saturn开发指引之java · vipshop/Saturn Wiki ·
GitHub https://github.com/vipshop/Saturn/wiki/saturn%E5%BC%80%E5%8F%91%E6%8C%87%E5%BC%95%E4%B9%8Bjava
Saturn架构文档 · vipshop/Saturn Wiki ·
GitHub https://github.com/vipshop/Saturn/wiki/Saturn%E6%9E%B6%E6%9E%84%E6%96%87%E6%A1%A3
Saturn作业调度算
法 https://github.com/vipshop/Saturn/wiki/Saturn%E4%BD%9C%E4%B8%9A%E8%B0%83%E5%BA%A6%E7%AE%97%E6%B3%95
作业开发与Spring和SpringBoot的集成
https://github.com/vipshop/Saturn/wiki/%E4%BD%9C%E4%B8%9A%E5%BC%80%E5%8F%91%E4%B8%8ESpring%E5%92%8CSpringBoot%E7%9A
分布式定时任务调度系统 Saturn 安装部署 https://blog.csdn.net/yuguotiang/article/details/77043874
唯品会分布式任务调度平台Saturn使用手册 https://blog.csdn.net/qq_34982426/article/details/82153376