## Quartz简介

### Quartz中用到的设计模式

Builder模式

Factory模式

组件模式：组件之间的耦合度低，并且可插拔，易于扩展

链式写法

### 核心概念

调度器

任务

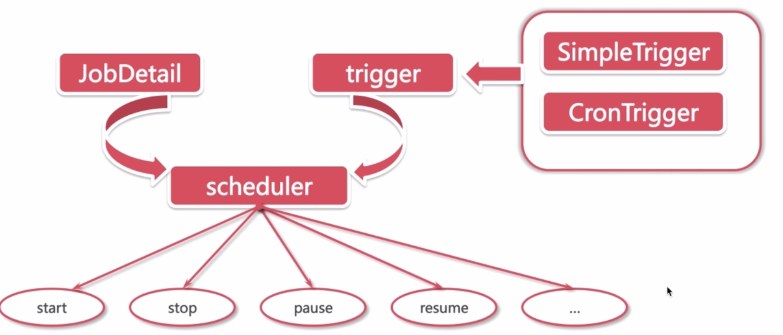
触发器

### Quartz体系结构

JobDetail：包含任务的实现类以及任务的信息

trigger：触发器（SimpleTrigger，CronTrigger可实现复杂的功能）

scheduler：调度器，将JobDetail和trigger绑定在一起



### 重要组成

Job：接口，需要实现execute方法，只有一个参数，提供了调度上下文的各种信息，job运行时的信息就保存在该参数的jobmetamap实例中。

JobDetail：quartz在每次执行任务的时候都会新创建一个job的实例，所以不接受job的实例，但是接受一个job实现类，可以通过反射机制实例化job。因此需要一个类来描述job的实现类和其他相关信息（名字 描述等）

JobBuilder：用来定义和创建Job实例

JobStore：用于保存Job的数据

Trigger：触发器，描述任务触发的规则

TriggerBuilder：用于定义和创建Trigger实例

ThreadPool：通过共享线程池中的线程提高效率

Scheduler：Trigger和JobDetail可以注册到Scheduler中，拥有自己的组和名称，组和名称必须唯一

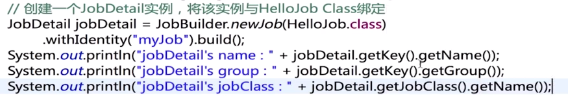
Calendar：一个Trigger可以和多个Calendar关联，以排除或包含某些时间点

监听器：JobListener TriggerListener SchedulerListener

## Quartz使用

自定义一个job类，实现Job接口，重写execute方法

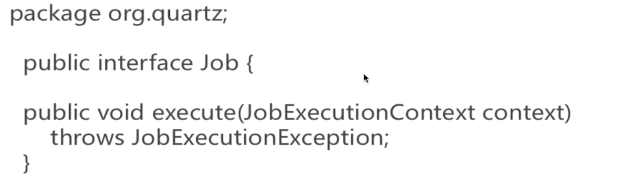
自定义一个Scheduler类，用于实现调度。需要创建JobDetail和Trigger。（这里的JobDetail和Trigger的组名可以相同，但是属于不同的组）。创建Scheduler，通过Factory创建。



可以在运行时获取这些信息然后执行不同的逻辑操作

## 具体类/接口详解

### job



当任务执行失败时，会抛出这个异常，因此可以捕获这个异常，对任务执行失败的情况进行处理

job实例的生命周期：每次调度器执行job时，它在调用execute方法前会创建一个job实例。当调用完成后，关联的job对象实例会被释放，释放的实例会被垃圾回收器回收。

### JobDetail

JobDetai会Job实例提供了许多设置属性，以及JobDataMap成员变量属性，用来存储特定Job实例的状态信息，调度器需要借助JobDetail对象来增加Job实例。

重要属性：

Name 任务的名称

Group 任务所在的组 默认是default组

jobClass 任务的实现类

jobDataMap 用来传递参数

### JobExecutionContext

当Scheduler调用一个Job，就会将JobExecutionContext传递给Job的execute方法

Job能通过JobExecutionContext对象访问到Quartz运行时候的环境以及Job本身的明细数据

### JobDataMap

在进行任务调度时，JobDataMap存储在JobExecutionContext中，非常方便获取

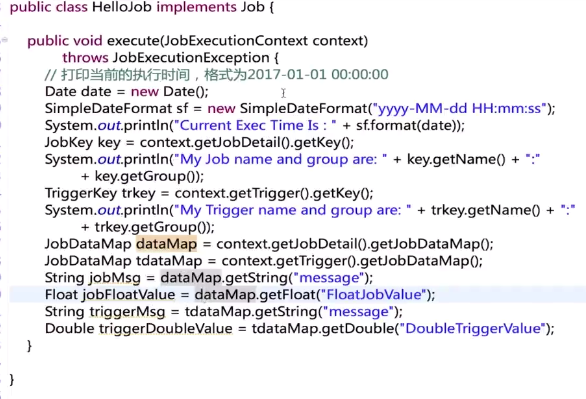
JobDataMap可以用来装载任何可序列化的数据对象，当job实例对象被执行时，这些参数对象会传递它

JobDataMap实现了JDK的Map接口，并且增加了一些非常方便的方法来存取基本数据类型

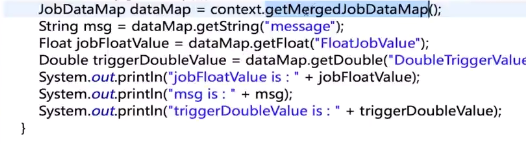
通过usingJobData方法可以传递自定义的参数



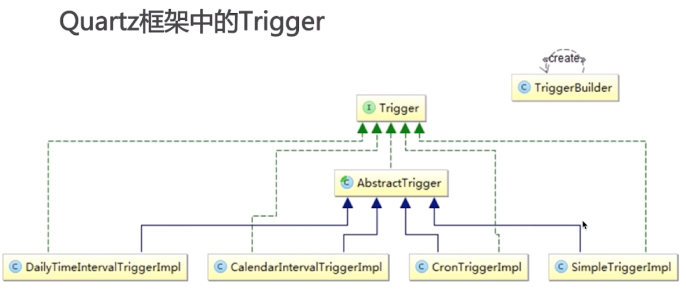
然后在Job类中获取到这些参数



第二种方式获取JobDataMap



### Trigger



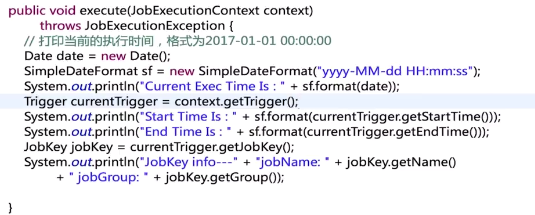
属性：

JobKey：表示job实例的标识，触发器被触发时，该指定的job实例会执行

StartTime：表示触发器的时间表首次被触发的时间，java.util.Date类型

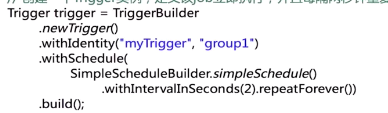
EndTime：指定触发器的不再被触发的时间，java.util.Date类型

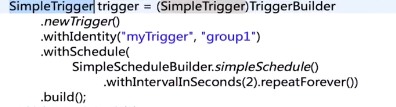




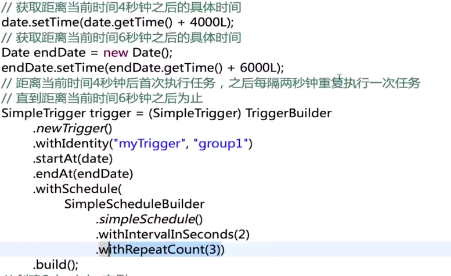
#### SimpleTrigger

在一个指定时间段内执行一次作业任务，或是在指定的时间间隔内多次执行作业任务









注意：

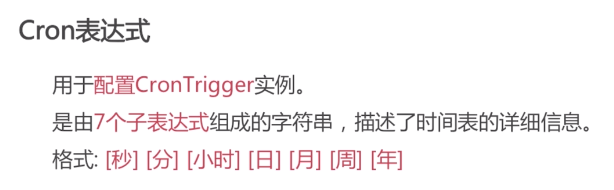
重复次数可以为0 正整数或者是SimpleTrigger.REPEAT\_INDEFINITELY常量值

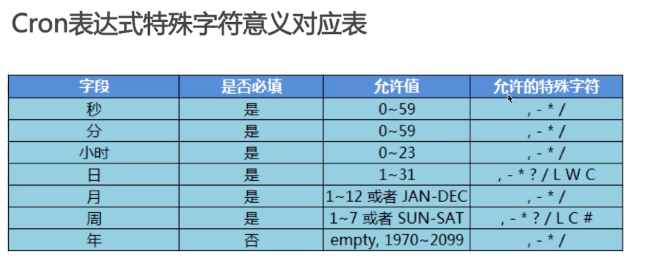
重复执行间隔必须为0或长整数

一旦被指定了endTime参数，它会覆盖重复次数参数的效果

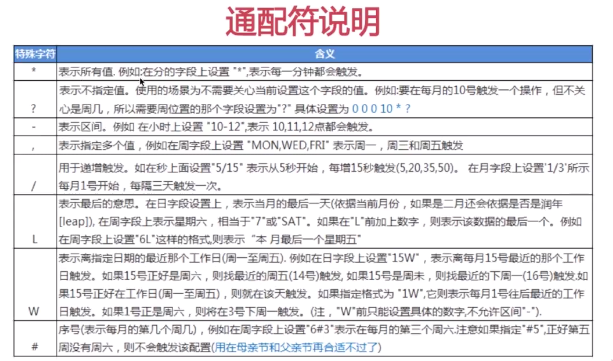
#### ConTrigger

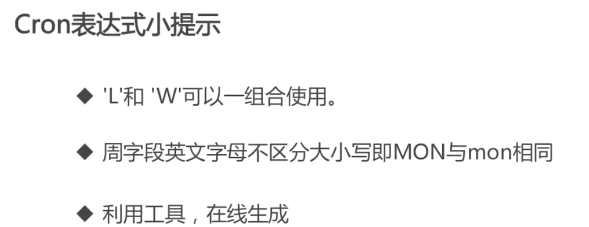
基于日历的作业调度器，而不是像SimpleTrigger那样精确指定间隔时间，比SimpleTrigger更常用



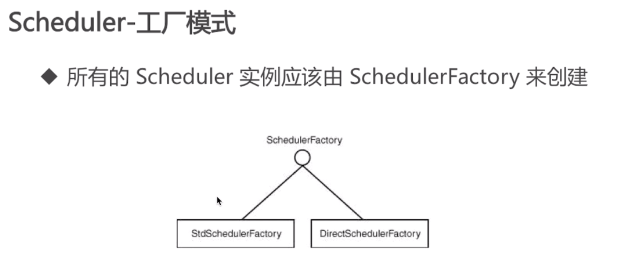


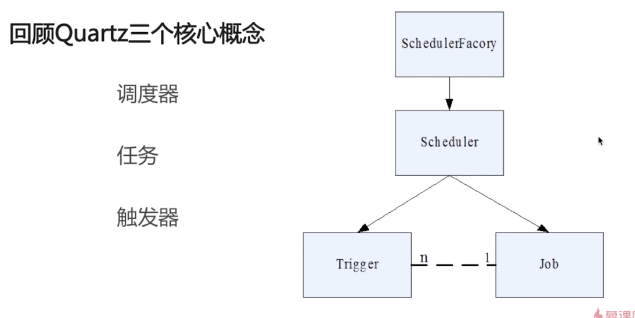


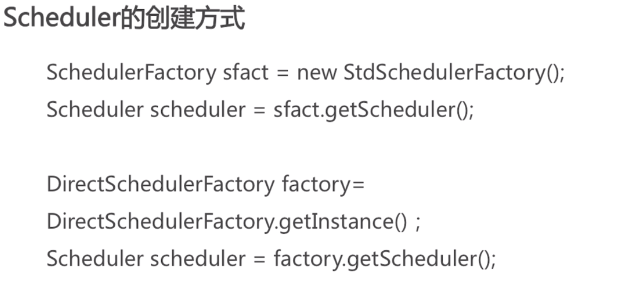




### Scheduler



job和trigger是一对多的关系



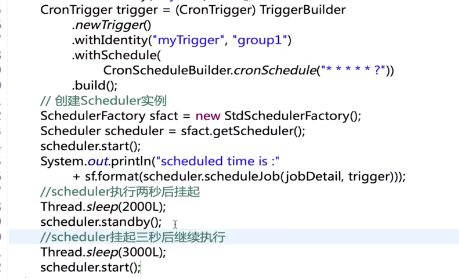




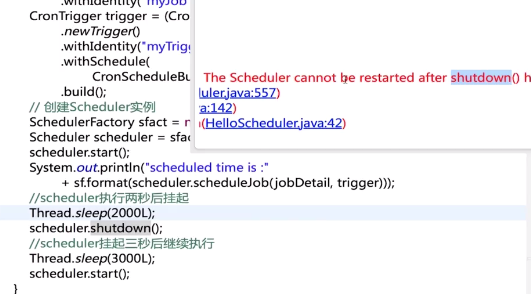
返回的Data是距离当前时间最近的一次执行时间

stangby是挂起当前scheduler，可以被重启

shutdown是关闭了scheduler，不可以再重启



shutdown之后再去start会抛出异常



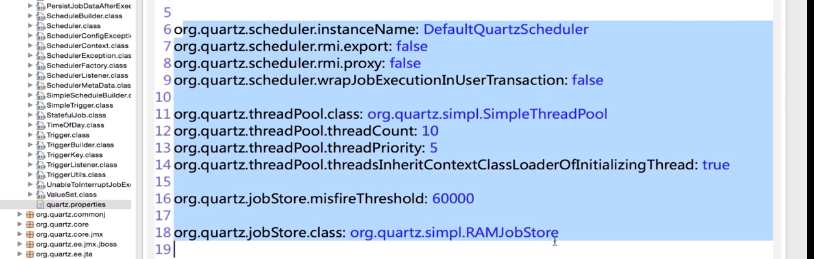
## quartz.properties

使用声明式编程，简化代码的实现，将配置的信息和代码分离，便于改变配置。

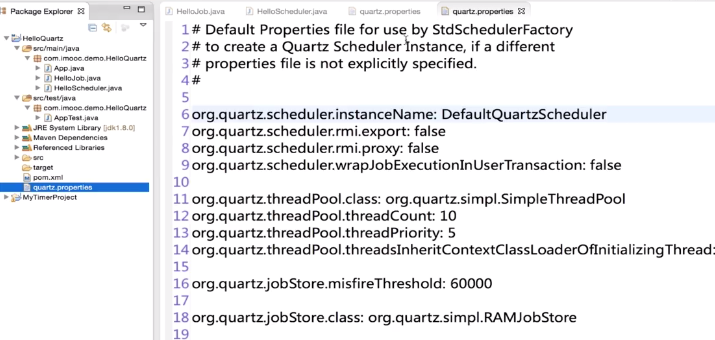
StdScheduler会读取quartz.properties文件

文档的加载顺序：

首先会先读取工程中的properties文件，如果工程中没有，则会读取quartz jar包中的org.quartz下的quartz.properties文件。

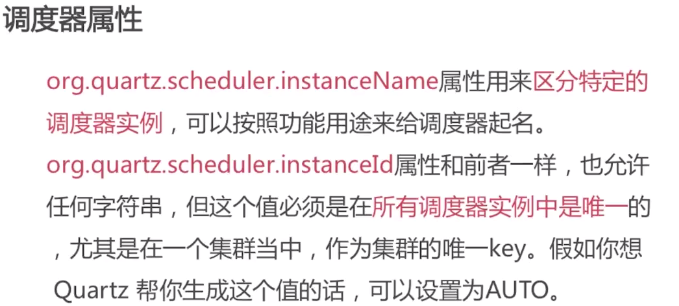


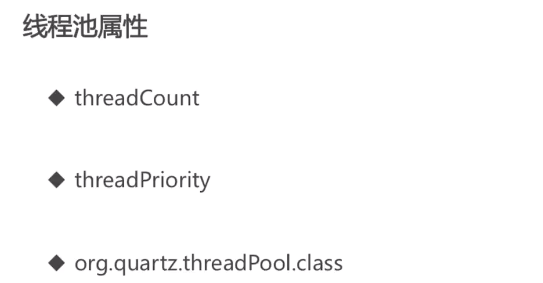
在工程中创建quartz.properties文件

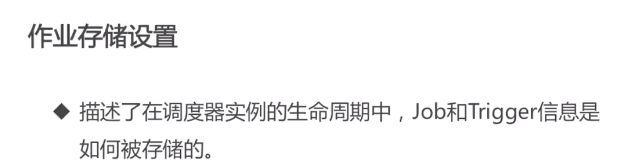


组成部分：











# Default Properties file for use by StdSchedulerFactory

# to create a Quartz Scheduler Instance, if a different

# properties file is not explicitly specified.

#

# ===========================================================================

# Configure Main Scheduler Properties 调度器属性

# ===========================================================================

org.quartz.scheduler.instanceName: DefaultQuartzScheduler

org.quartz.scheduler.instanceid:AUTO

org.quartz.scheduler.rmi.export: false

org.quartz.scheduler.rmi.proxy: false

org.quartz.scheduler.wrapJobExecutionInUserTransaction: false

# ===========================================================================

# Configure ThreadPool 线程池属性

# ===========================================================================

#线程池的实现类（一般使用SimpleThreadPool即可满足几乎所有用户的需求）

org.quartz.threadPool.class: org.quartz.simpl.SimpleThreadPool

#指定线程数，至少为1（无默认值）(一般设置为1-100直接的整数合适)

org.quartz.threadPool.threadCount: 10

#设置线程的优先级（最大为java.lang.Thread.MAX\_PRIORITY 10，最小为Thread.MIN\_PRIORITY 1，默认为5）

org.quartz.threadPool.threadPriority: 5

#设置SimpleThreadPool的一些属性

#设置是否为守护线程

#org.quartz.threadpool.makethreadsdaemons = false

#org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread: true

#org.quartz.threadpool.threadsinheritgroupofinitializingthread=false

#线程前缀默认值是：[Scheduler Name]\_Worker

#org.quartz.threadpool.threadnameprefix=swhJobThead;

# 配置全局监听(TriggerListener,JobListener) 则应用程序可以接收和执行 预定的事件通知

# ===========================================================================

# Configuring a Global TriggerListener 配置全局的Trigger监听器

# MyTriggerListenerClass 类必须有一个无参数的构造函数，和 属性的set方法，目前2.2.x只支持原始数据类型的值（包括字符串）

# ===========================================================================

#org.quartz.triggerListener.NAME.class = com.swh.MyTriggerListenerClass

#org.quartz.triggerListener.NAME.propName = propValue

#org.quartz.triggerListener.NAME.prop2Name = prop2Value

# ===========================================================================

# Configuring a Global JobListener 配置全局的Job监听器

# MyJobListenerClass 类必须有一个无参数的构造函数，和 属性的set方法，目前2.2.x只支持原始数据类型的值（包括字符串）

# ===========================================================================

#org.quartz.jobListener.NAME.class = com.swh.MyJobListenerClass

#org.quartz.jobListener.NAME.propName = propValue

#org.quartz.jobListener.NAME.prop2Name = prop2Value

# ===========================================================================

# Configure JobStore 存储调度信息（工作，触发器和日历等）

# ===========================================================================

# 信息保存时间 默认值60秒

org.quartz.jobStore.misfireThreshold: 60000

#保存job和Trigger的状态信息到内存中的类

org.quartz.jobStore.class: org.quartz.simpl.RAMJobStore

# ===========================================================================

# Configure SchedulerPlugins 插件属性 配置

# ===========================================================================

# 自定义插件

#org.quartz.plugin.NAME.class = com.swh.MyPluginClass

#org.quartz.plugin.NAME.propName = propValue

#org.quartz.plugin.NAME.prop2Name = prop2Value

#配置trigger执行历史日志（可以看到类的文档和参数列表）

org.quartz.plugin.triggHistory.class = org.quartz.plugins.history.LoggingTriggerHistoryPlugin

org.quartz.plugin.triggHistory.triggerFiredMessage = Trigger {1}.{0} fired job {6}.{5} at: {4, date, HH:mm:ss MM/dd/yyyy}

org.quartz.plugin.triggHistory.triggerCompleteMessage = Trigger {1}.{0} completed firing job {6}.{5} at {4, date, HH:mm:ss MM/dd/yyyy} with resulting trigger instruction code: {9}

#配置job调度插件 quartz\_jobs(jobs and triggers内容)的XML文档

#加载 Job 和 Trigger 信息的类 （1.8之前用：org.quartz.plugins.xml.JobInitializationPlugin）

org.quartz.plugin.jobInitializer.class = org.quartz.plugins.xml.XMLSchedulingDataProcessorPlugin

#指定存放调度器(Job 和 Trigger)信息的xml文件，默认是classpath下quartz\_jobs.xml

org.quartz.plugin.jobInitializer.fileNames = my\_quartz\_job2.xml

#org.quartz.plugin.jobInitializer.overWriteExistingJobs = false

org.quartz.plugin.jobInitializer.failOnFileNotFound = true

#自动扫描任务单并发现改动的时间间隔,单位为秒

org.quartz.plugin.jobInitializer.scanInterval = 10

#覆盖任务调度器中同名的jobDetail,避免只修改了CronExpression所造成的不能重新生效情况

org.quartz.plugin.jobInitializer.wrapInUserTransaction = false

# ===========================================================================

# Sample configuration of ShutdownHookPlugin ShutdownHookPlugin插件的配置样例

# ===========================================================================

#org.quartz.plugin.shutdownhook.class = \org.quartz.plugins.management.ShutdownHookPlugin

#org.quartz.plugin.shutdownhook.cleanShutdown = true

#

# Configure RMI Settings 远程服务调用配置

#

#如果你想quartz-scheduler出口本身通过RMI作为服务器，然后设置“出口”标志true(默认值为false)。

#org.quartz.scheduler.rmi.export = false

#主机上rmi注册表(默认值localhost)

#org.quartz.scheduler.rmi.registryhost = localhost

#注册监听端口号（默认值1099）

#org.quartz.scheduler.rmi.registryport = 1099

#创建rmi注册，false/never：如果你已经有一个在运行或不想进行创建注册

# true/as\_needed:第一次尝试使用现有的注册，然后再回来进行创建

# always:先进行创建一个注册，然后再使用回来使用注册

#org.quartz.scheduler.rmi.createregistry = never

#Quartz Scheduler服务端端口，默认是随机分配RMI注册表

#org.quartz.scheduler.rmi.serverport = 1098

#true:链接远程服务调度(客户端),这个也要指定registryhost和registryport，默认为false

# 如果export和proxy同时指定为true，则export的设置将被忽略

#org.quartz.scheduler.rmi.proxy = false

## Spring和Quartz

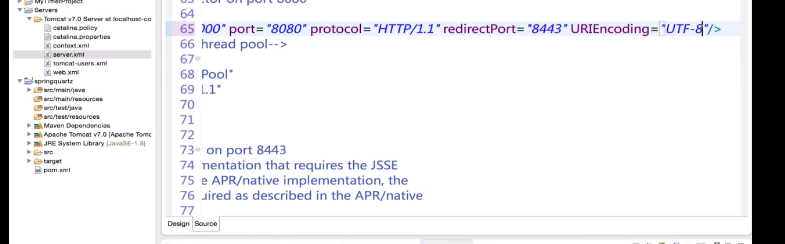
Maven创建web工程

首先修改该项目的buildpath的jdk版本

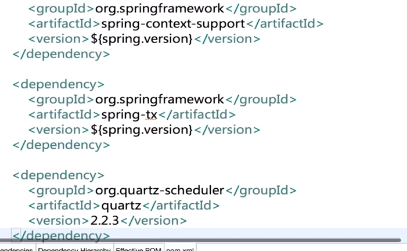
手动创建src/test/resources，修改buildpath下的outputfolder

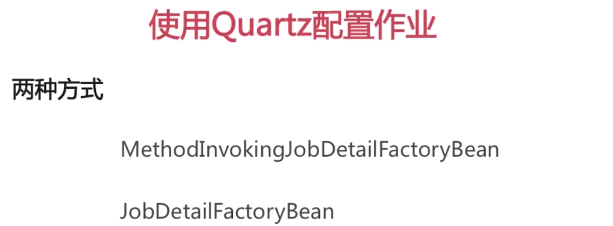
工程报错，需要引入tomcat jar包

如果页面输入中文是乱码，需要修改tomcat的server配置文件



在pom文件中加入quartz，还需要加入其依赖的包



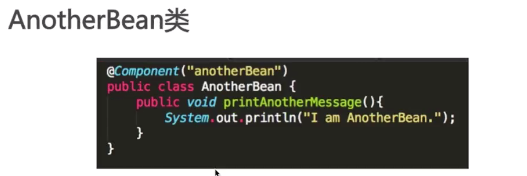




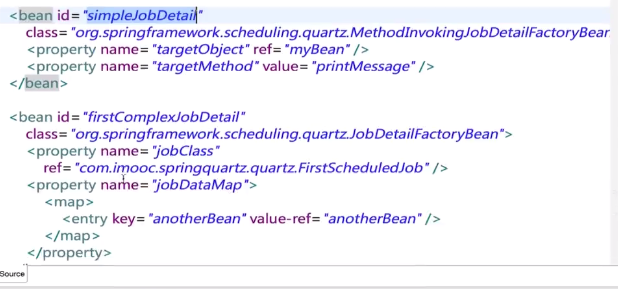


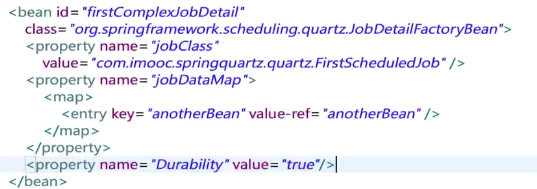


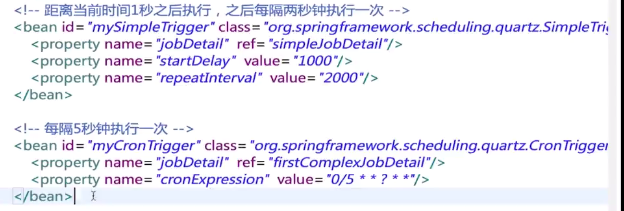


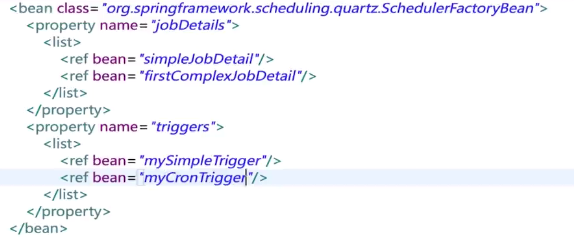


在dispatcher-servlet.xml中加入对quartz的配置









## Quartz分布式

在集群中，每台机器都会执行任务，从而造成了重复执行任务的问题。

解决：quartz支持分布式的任务调度，在集群中，各个不同的机器共用一个调度器，调度器按照一定的算法选择集群中某一台机器执行任务。

这样就避免了每台机器都进行同一个任务的调度。

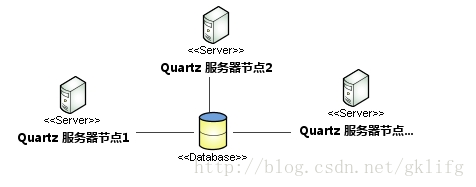
### 步骤

1 在数据库中创建quartz表，这些表由quartz自己维护，用于保存数据和维护程序运行的状态，这些表对于程序员来说是不需要关注的，只需要创建好表就行。

2 在工程中的src下增加properties配置文件，进行集群相关操作的配置。

1. <span style="font-family:Microsoft YaHei;font-size:12px;">#==============================================================
2. #Configure Main Scheduler Properties
3. #==============================================================
4. #配置集群时，quartz调度器的id，由于配置集群时，只有一个调度器，必须保证每个服务器该值都相同，可以不用修改，只要每个ams都一样就行
5. org.quartz.scheduler.instanceName = Scheduler1
6. #集群中每台服务器自己的id，AUTO表示自动生成，无需修改
7. org.quartz.scheduler.instanceId = AUTO
8. #==============================================================
9. #Configure ThreadPool
10. #==============================================================
11. #quartz线程池的实现类，无需修改
12. org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
13. #quartz线程池中线程数，可根据任务数量和负责度来调整
14. org.quartz.threadPool.threadCount = 5
15. #quartz线程优先级
16. org.quartz.threadPool.threadPriority = 5
17. #==============================================================
18. #Configure JobStore
19. #==============================================================
20. #表示如果某个任务到达执行时间，而此时线程池中没有可用线程时，任务等待的最大时间，如果等待时间超过下面配置的值(毫秒)，本次就不在执行，而等待下一次执行时间的到来，可根据任务量和负责程度来调整
21. org.quartz.jobStore.misfireThreshold = 60000
22. #实现集群时，任务的存储实现方式，org.quartz.impl.jdbcjobstore.JobStoreTX表示数据库存储，无需修改
23. org.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreTX
24. #quartz存储任务相关数据的表的前缀，无需修改
25. org.quartz.jobStore.tablePrefix = QRTZ\_
26. #连接数据库数据源名称，与下面配置中org.quartz.dataSource.myDS的myDS一致即可，可以无需修改
27. org.quartz.jobStore.dataSource = myDS
28. #是否启用集群，启用，改为true,注意：启用集群后，必须配置下面的数据源，否则quartz调度器会初始化失败
29. org.quartz.jobStore.isClustered = false
30. #集群中服务器相互检测间隔，每台服务器都会按照下面配置的时间间隔往服务器中更新自己的状态，如果某台服务器超过以下时间没有checkin，调度器就会认为该台服务器已经down掉，不会再分配任务给该台服务器
31. org.quartz.jobStore.clusterCheckinInterval = 20000
32. #==============================================================
33. #Non-Managed Configure Datasource
34. #==============================================================
35. #配置连接数据库的实现类，可以参照IAM数据库配置文件中的配置
36. org.quartz.dataSource.myDS.driver = com.mysql.jdbc.Driver
37. #配置连接数据库连接，可以参照IAM数据库配置文件中的配置
38. org.quartz.dataSource.myDS.URL = jdbc:mysql://localhost:3306/test
39. #配置连接数据库用户名
40. org.quartz.dataSource.myDS.user = yunxi
41. #配置连接数据库密码
42. org.quartz.dataSource.myDS.password = 123456
43. #配置连接数据库连接池大小，一般为上面配置的线程池的2倍
44. org.quartz.dataSource.myDS.maxConnections = 10
45. </span>

### 集群架构



数据库是各个节点上调度器的枢纽，各个节点并不知道其他节点的存在，只是通过数据库进行间接的沟通。

每个节点都遵守相同的操作贵伐，使得对数据库的操作可以串行执行，而不同名称的调度器又可以互不影响的并行执行。

http://www.cnblogs.com/davidwang456/p/4205237.html