# Quartz

## 1：安装和介绍

### 1.1：简单介绍：

**是一个任务调度框架**，通过触发器设**置作业的定时运行规则**，**来执行定时任务**

#### **1.1.1:任务：处理逻辑**

#### **1.1.2触发器：什么时候执行**

### 1.2：官方文档地址

<http://www.quartz-scheduler.org/documentation/quartz-2.2.x/tutorials/>

### 1.3：构建

#### 1.3.1：在maven中引入依赖

<!-- https://mvnrepository.com/artifact/org.quartz-scheduler/quartz -->

<dependency>

<groupId>org.quartz-scheduler</groupId>

<artifactId>quartz</artifactId>

<version>2.3.0</version>

</dependency>

## 2:简单示例:

### 2.1:定义任务

**public** **class** HelloJob **implements** Job {

@Override

**public** **void** execute(JobExecutionContext context) **throws** JobExecutionException {

//你的执行逻辑

System.***out***.println("hellojob");

}

}

继承Job接口 重写方法execute，实现自己的任务逻辑

#### 2.1.1:Job和jobDetail

Job:表示要完成的任务

JobDetail：job实例包含的属性

在job类中，不应该定义有状态的数据属性，因为在job的多次执行中，这些属性的值不会保留。

#### 2.1.2: 两个注解

@DisallowConcurrentExecution

不能并发执行同一个jobdetail实例

@PersistJobDataAfterExecution

在execute执行后更新jobdetail实例的jobdatamap

#### 2.1.3：任务的其他属性

Durability(耐久性？)

如果一个任务不是durable，那么当没有Trigger关联它的时候，它就会被自动删除。

RequestsRecovery

如果一个任务是"requests recovery"，那么当任务运行过程非正常退出时（比如进程崩溃，机器断电，但不包括抛出异常这种情况），Quartz再次启动时，会重新运行一次这个任务实例。

#### 2.1.4: Quartz调度一次任务

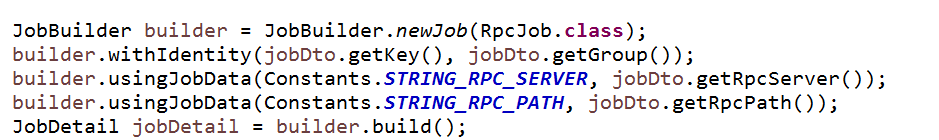
JobClass jobClass=JobDetail.getJobClass()

Job jobInstance=jobClass.newInstance()。所以Job实现类，必须有一个public的无参构建方法。

jobInstance.execute(JobExecutionContext context)。JobExecutionContext是Job运行的上下文，可以获得Trigger、Scheduler、JobDetail的信息。

也就是说，每次调度都会创建一个新的Job实例，这样的好处是有些任务并发执行的时候，不存在对临界资源的访问问题——当然，如果需要共享JobDataMap的时候，还是存在临界资源的并发访问的问题。

#### 2.1.5:利用jobdatamap为job实例配置属性



### 2.2:配置触发器，调度任务

**public** **class** QuartzTest {

Logger log = LoggerFactory.*getLogger*(QuartzTest.**class**);

**public** **static** **void** main(String args[]) {

SchedulerFactory sf=**new** StdSchedulerFactory();

//获取scheduler实例

Scheduler sched=**null**;

**try** {

sched=sf.getScheduler();

//JobDetail 定义的是任务数据，而真正的执行逻辑是在Job中

JobDetail job=*newJob*(HelloJob.**class**)

.withIdentity("job1","group1")

.build();

//触发器 使用simpletrigger 每1秒执行一次

Trigger trigger=*newTrigger*()

.withIdentity("trigger1","group2")

.startNow()

.withSchedule(*simpleSchedule*()

.withIntervalInSeconds(1)

.repeatForever()

)

.build();

//绑定触发器和任务

sched.scheduleJob(job, trigger);

// Starts the Scheduler's threads

sched.start();

Thread.*sleep*(1000L);

//终止scheduler 回收全部资源

//The scheduler cannot be re-started.

Sched.shutdown();

} **catch** (SchedulerException | InterruptedException e) {

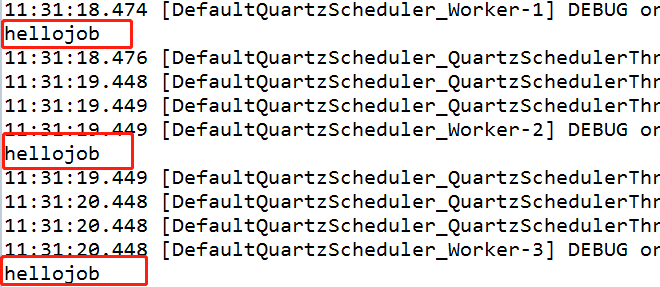
e.printStackTrace();

}

}

}

### 2.3:执行结果



每秒执行一次

## 3：配置参数

### 3.1：主要配置参数

|  |  |  |  |
| --- | --- | --- | --- |
| org.quartz.scheduler.instanceName | no | string | 'QuartzScheduler' |
| org.quartz.scheduler.instanceId | no | string | 'NON\_CLUSTERED' |
| org.quartz.scheduler.instanceIdGenerator.class | no | string (class name) | org.quartz.simpl .SimpleInstanceIdGenerator |
| org.quartz.scheduler.threadName | no | string | instanceName + '\_QuartzSchedulerThread' |
| org.quartz.scheduler .makeSchedulerThreadDaemon | no | boolean | false |
| org.quartz.scheduler .threadsInheritContextClassLoaderOfInitializer | no | boolean | false |
| org.quartz.scheduler.idleWaitTime | no | long | 30000 |
| org.quartz.scheduler.dbFailureRetryInterval | no | long | 15000 |
| org.quartz.scheduler.classLoadHelper.class | no | string (class name) | org.quartz.simpl .CascadingClassLoadHelper |
| org.quartz.scheduler.jobFactory.class | no | string (class name) | org.quartz.simpl.PropertySettingJobFactory |
| org.quartz.context.key.SOME\_KEY | no | string | none |
| org.quartz.scheduler.userTransactionURL | no | string (url) | 'java:comp/UserTransaction' |
| org.quartz.scheduler .wrapJobExecutionInUserTransaction | no | boolean | false |
| org.quartz.scheduler.skipUpdateCheck | no | boolean | false |
| org.quartz.scheduler .batchTriggerAcquisitionMaxCount | no | int | 1 |
| org.quartz.scheduler .batchTriggerAcquisitionFireAheadTimeWindow | no | long | 0 |

### 3.2：主要参数说明

#### **1：org.quartz.scheduler.instanceName**

**可以为任意字符串，对scheduler来说此值没有意义，可用来区分同一个系统中的多个实例，如果在集群环境下使用，对每个实例使用同个名字，是这些实例在逻辑上是同一个scheduler**

#### **2：org.quartz.scheduler.instanceId**

**可以为任意字符串，如果在集群环境下使用，必须保证所有的scheduler实例的此值唯一，如果希望instanceId自动生成，可以使用**“AUTO” **，如果希望此值来源与系统属性，可以使用**“SYS\_PROP”

#### 3：**org.quartz.scheduler.instanceIdGenerator.class**

**只有当**org.quartz.scheduler.instanceId 被设置为 “AUTO”时生效，默认是org.quartz.simpl.SimpleInstanceIdGenerator，，该instanceId生成器根据主机名和时间戳，其他的instanceId生成器包括SystemPropertyInstanceIdGenerator，从系统属性org.quartz.scheduler.instanceId获取instanceId，HostnameInstanceIdGenerator ，使用当前主机名，InetAddress.getLocalHost().getHostName()，也可以继承InstanceIdGenerator接口自己实现

#### **4: org.quartz.scheduler.threadName**

可以任何有效的java线程名称，如果此值没有被设置，此值将会为“org.quartz.scheduler.instanceName”的值加上‘\_QuartzSchedulerThread’

#### 5：**org.quartz.scheduler.threadsInheritContextClassLoaderOfInitializer**

True或是false，标志quartz产生的线程是否需要继承初始化quartz实例的线程，这将会影响到quartz的主任务线程，DBCJobStore’s misfire handling thread (if JDBCJobStore is used), cluster recovery thread (if clutering is used), and threads in SimpleThreadPool (if SimpleThreadPool is used)，将此值设置为true有助于加载类，查找JNDI

#### 6：**org.quartz.scheduler.idleWaitTime**

这个属性设置了scheduler从空闲状态转为重新查询可用triggers的毫秒数。通常不建议修改它，除非你用到了XA transactions或者遇到了triggers本该出发但发生延迟的情况。5000ms以下通常不建议使用，这将产生大量的数据库查询操作。小于1000ms是不支持的。

#### 7：**org.quartz.scheduler.dbFailureRetryInterval**

当scheduler检测到jobstore中到某处的连接断开后，再次尝试重新连接等待的毫秒数，当使用ramjobstore时这个参数无效。

#### 8：**org.quartz.scheduler.classLoadHelper.class**

大多数应用都使用默认值，“org.quartz.simpl.CascadingClassLoadHelper” ，他会依次使用其他的ClassLoadHelper 直到可用为止。

#### 9：**org.quartz.scheduler.jobFactory.class**

**JobFactory的类名。默认“org.quartz.simpl.SimpleJobFactory”，你也可以用“org.quartz.simpl.PropertySettingJobFactory”。SimpleJobFactory调用newInstance()方法，PropertySettingJobFactory也是，但它可以将SchedulerContext和Job的内容或者trigger的JobDataMaps利用反射设置job的属性。**

#### ****10：org.quartz.context.key.SOME\_KEY****

Scheduler context中的新建的键值对，“org.quartz.context.key.MyKey = MyValue”等价于scheduler.getContext().put(“MyKey”, “MyValue”)

#### 11：****org.quartz.scheduler.wrapJobExecutionInUserTransaction****

在执行job之前Quartz开启UserTransaction ，此属性应该设置为true。Job执行完，JobDataMap更新完（如果是StatefulJob）事务会提交。默认值是false，你也可以在你的job类上使用@ExecuteInJTATransaction 注解，以便在各自的job上决定是否开启JTA事务。

#### 12：****org.quartz.scheduler.skipUpdateCheck****

设置了是否需要检查更新。在生产环境，最好关闭检查更新。

#### 13：****org.quartz.scheduler.batchTriggerAcquisitionMaxCount****

一个scheduler节点允许接收的trigger的最大数，默认值为1，这个值越大，定时任务执行的越多，但代价是集群节点之间的不均衡

#### 14：****org.quartz.scheduler.batchTriggerAcquisitionFireAheadTimeWindow****

trigger被接受和提前执行的毫秒数。默认值为0。这个值约大，trigger接受和执行的数量越大，但其代价不能精确的估计。这对那些scheduler有大量的需要相近的时间执行的triggers有用。

#### 15：****org.quartz.scheduler.userTransactionURL****

应该设置成JNDI URL，通过它Quartz可以找到应用服务的UserTransaction manger。默认值是“java:comp/UserTransaction”-大部分应用服务都是用这个配置。Websphere用户需要设置成“jta/usertransaction”。这个属性仅用于Quartz配置使用JobStoreCMT，并且org.quartz.scheduler.wrapJobExecutionInUserTransaction设置为true。

#### 16：org.quartz.jobStore.misfireThreshold

Misfire触发阈值，任务应执行时间与实际执行时间的差值大于该阈值的话触发misfire策略

##### Misfire策略

###### **SimpleTrigger的Misfire Instruction**

MISFIRE\_INSTRUCTION\_IGNORE\_MISFIRE\_POLICY

在资源合适的情况下，重新触发所有的misfire任务，不影响现有的调度时间（所有的trigger都适用）

MISFIRE\_INSTRUCTION\_FIRE\_NOW

忽略已经misfire的任务，并且立即执行调度（适用于只执行一次的任务）

MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_EXISTING\_REPEAT\_COUNT

将starttime设置为当前时间，并且重新调度任务，包括misfire的任务

MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_REMAINING\_REPEAT\_COUNT

将starttime设置为当前时间，并且重新调度任务，忽略misfire的任务

MISFIRE\_INSTRUCTION\_RESCHEDULE\_NEXT\_WITH\_REMAINING\_COUNT

在下一次任务调度的时间，重新开始调度任务，忽略misfire的任务

MISFIRE\_INSTRUCTION\_RESCHEDULE\_NEXT\_WITH\_EXISTING\_COUNT

在下一次任务调度的时间，重新开始调度任务，包括misfire的任务

MISFIRE\_INSTRUCTION\_SMART\_POLICY 所有trigger的misfire的默认值

：如果是只执行一次的调度，使用MISFIRE\_INSTRUCTION\_FIRE\_NOW

：如果是不限次数的调度，使用MISFIRE\_INSTRUCTION\_RESCHEDULE\_NEXT\_WITH\_REMAINING\_COUNT

其他情况使用：MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_EXISTING\_REPEAT\_COUNT

###### **CronTrigger** **Misfire Instruction Constants**

MISFIRE\_INSTRUCTION\_IGNORE\_MISFIRE\_POLICY

在资源合适的情况下，重新触发所有的misfire任务，不影响现有的调度时间（所有的trigger都适用）

MISFIRE\_INSTRUCTION\_DO\_NOTHING

在上一个调度执行完之后，什么都不要做，等下一个触发时间，再照常执行

MISFIRE\_INSTRUCTION\_FIRE\_NOW

忽略已经misfire的任务，并且立即执行调度（适用于只执行一次的任务）

MISFIRE\_INSTRUCTION\_SMART\_POLICY 所有trigger的misfire的默认值

### 3.3：线程池配置参数

|  |  |  |  |
| --- | --- | --- | --- |
| org.quartz.threadPool.class | yes | string (class name) | null |
| org.quartz.threadPool.threadCount | yes | int | -1 |
| org.quartz.threadPool.threadPriority | no | int | Thread.NORM\_PRIORITY (5) |

#### org.quartz.threadPool.class 配置线程池实现类

默认实现为“org.quartz.simpl.SimpleThreadPool”

#### org.quartz.threadPool.threadCount 线程数

#### org.quartz.threadPool.threadPriority 线程优先级

### 3.4:datasource配置参数

|  |  |  |  |
| --- | --- | --- | --- |
| org.quartz.dataSource.NAME.driver | yes | String | null |
| org.quartz.dataSource.NAME.URL | yes | String | null |
| org.quartz.dataSource.NAME.user | no | String | "" |
| org.quartz.dataSource.NAME.password | no | String | "" |
| org.quartz.dataSource.NAME.maxConnections | no | int | 10 |
| org.quartz.dataSource.NAME.validationQuery | no | String | null |
| org.quartz.dataSource.NAME.idleConnectionValidationSeconds | no | int | 50 |
| org.quartz.dataSource.NAME.validateOnCheckout | no | boolean | false |
| org.quartz.dataSource.NAME.discardIdleConnectionsSeconds | no | int | 0 (disabled) |

### 3.5：jobstore配置参数

#### 3.5.1:RAMJobStore

将schedule相关信息保存在RAM中，轻量级，速度快，但是应用重启时相关信息都将丢失。

org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore

#### 3.5.2:JDBC-JobStore

将schedule相关信息保存在数据库中.有两种实现：JobStoreTX和JobStoreCMT

要在数据库中建表，表的脚本在quartz-2.3.0.jar\org\quartz\impl\jdbcjobstore下有mysql，oracle这些数据库的脚本

##### 3.5.2.1：JobStoreTX为quartz自己管理事务

|  |  |  |  |
| --- | --- | --- | --- |
| org.quartz.jobStore.driverDelegateClass | yes | string | null |
| org.quartz.jobStore.dataSource | yes | string | null |
| org.quartz.jobStore.tablePrefix | no | string | "QRTZ\_" |
| org.quartz.jobStore.useProperties | no | boolean | false |
| org.quartz.jobStore.misfireThreshold | no | int | 60000 |
| org.quartz.jobStore.isClustered | no | boolean | false |
| org.quartz.jobStore.clusterCheckinInterval | no | long | 15000 |

org.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreTX

配置DataSource、

org.quartz.jobStore.dataSource

##### 3.5.2.2：JobStoreCMT为托管容器管理事务，

|  |  |  |  |
| --- | --- | --- | --- |
| org.quartz.jobStore.driverDelegateClass | yes | string | null |
| org.quartz.jobStore.dataSource | yes | string | null |
| org.quartz.jobStore.nonManagedTXDataSource | yes | string | null |
| org.quartz.jobStore.tablePrefix | no | string | "QRTZ\_" |
| org.quartz.jobStore.useProperties | no | boolean | false |
| org.quartz.jobStore.misfireThreshold | no | int | 60000 |
| org.quartz.jobStore.isClustered | no | boolean | false |
| org.quartz.jobStore.clusterCheckinInterval | no | long | 15000 |
| org.quartz.jobStore.maxMisfiresToHandleAtATime | no | int | 20 |

org.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreCMT

配置DataSource、

org.quartz.jobStore.dataSource

## 4：quartz中的java API

### 4.1：官方文档地址：

<http://www.quartz-scheduler.org/api/2.2.1/index.html>

### 4．2常用api

#### 4.2.1：任务绑定触发器

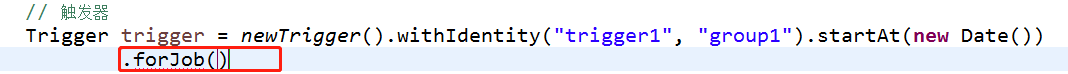
##### 绑定任务和触发器

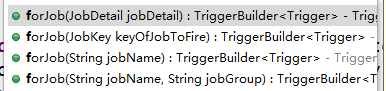
Date scheduleJob(JobDetail jobDetail, Trigger trigger)

throws SchedulerException;

##### 绑定任务和触发器（绑定的任务在触发器中定义）

Date scheduleJob(Trigger trigger) throws SchedulerException;





##### 绑定任务和触发器集合

void scheduleJob(JobDetail jobDetail, Set<? extends Trigger> triggersForJob, boolean replace) throws SchedulerException;

#### 4.2.2：任务解除绑定触发器

##### 移除触发器，（如果任务没有关联触发器，而且任务也没有持久化，任务也会被删除）

**boolean** unscheduleJob(TriggerKey triggerKey) **throws** SchedulerException;

移除触发器集合

**boolean** unscheduleJobs(List<TriggerKey> triggerKeys)

**throws** SchedulerException;

#### 任务替换原有的触发器（两个必须关联同一个任务，也就是任务的jobkey相同）

Date rescheduleJob(TriggerKey triggerKey, Trigger newTrigger)

**throws** SchedulerException;

#### 替换调度中的任务（该任务必须持久化，如果没有持久化会抛出SchedulerException）（replace设置为true）

**void** addJob(JobDetail jobDetail, **boolean** replace)

**throws** SchedulerException;

#### 删除任务

##### 删除任务（删除任务还有和任务关联的触发器）

**boolean** deleteJob(JobKey jobKey)

**throws** SchedulerException;

##### 批量删除任务

**boolean** deleteJobs(List<JobKey> jobKeys)

**throws** SchedulerException;

#### 立即执行任务

##### 执行任务

**void** triggerJob(JobKey jobKey)

**throws** SchedulerException;

##### 立即执行任务（data the (possibly <code>null</code>) JobDataMap to be

\* associated with the trigger that fires the job immediately.）

**void** triggerJob(JobKey jobKey, JobDataMap data)

**throws** SchedulerException;

#### 暂停任务(暂停期间，触发器misfire了，将会触发misfire策略)（可以设置misfire策略为忽略下一次触发时间触发，忽略已经misfire的）

**void** pauseJob(JobKey jobKey)

**throws** SchedulerException;

##### 暂停同组的任务

**void** pauseJobs(GroupMatcher<JobKey> matcher) **throws** SchedulerException;

#### 恢复任务

##### 单个任务

**void** resumeJob(JobKey jobKey)

**throws** SchedulerException;

##### 同组下的任务

**void** resumeJobs(GroupMatcher<JobKey> matcher) **throws** SchedulerException;

5：和spring集成

在与Spring整合时，Spring3.1.0之前的版本只支持Quartz1.6以往的版本。