手撸XXL-JOB（二）——定时任务的管理

在上一节中，我们介绍了SpringBoot中关于定时任务的执行方式，以及ScheduledExecutorService接口提供的定时任务执行方法。假设我们现在要写类似XXL-JOB这样的任务调度平台，那么，对于任务的管理，是尤为重要的。接下来我们将一步一步，实现一个任务调度管理类。

YangJobManager类基础实现

假设我们现在的任务管理类，名为YangJobManager类。对于定时任务的执行，我们最终会调用到ScheduledExecutorService的相关方法，因此，我们的YangJobManager类，需要有ScheduledExecutorService属性，其次，我们希望能对要执行的定时线程任务，其命名进行修改，因此，我们需要有一个线程工厂的属性。基于上述两点，我们对YangJobManager类进行实现：

```
package com.yang.job;

import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ThreadFactory;
import java.util.concurrent.TimeUnit;

public class YangJobManager {
    private ScheduledExecutorService scheduledExecutorService;

    private ThreadFactory threadFactory;

    public YangJobManager(ScheduledExecutorService
scheduledExecutorService, ThreadFactory threadFactory) {
        this.scheduledExecutorService = scheduledExecutorService;
        this.threadFactory = threadFactory;
    }

    public void schedule(Runnable runnable, Long delay) {
        Thread thread = threadFactory.newThread(runnable);
        scheduledExecutorService.schedule(thread, delay,
TimeUnit.SECONDS);
    }

    public void scheduleWithFixedDelay(Runnable runnable, Long delay,
```

```
Long period) {
        Thread thread = threadFactory.newThread(runnable);
        scheduledExecutorService.scheduleWithFixedDelay(thread, delay,
period, TimeUnit.SECONDS);
    }

    public void scheduleWithFixedRate(Runnable runnable, Long delay,
Long period) {
        Thread thread = threadFactory.newThread(runnable);
        scheduledExecutorService.scheduleAtFixedRate(thread, delay,
period, TimeUnit.SECONDS);
    }

    public void shutdown() {
        if (this.scheduledExecutorService == null) {
            return;
        }
        if (this.scheduledExecutorService.isShutdown()) {
            return;
        }
        scheduledExecutorService.shutdown();
        try {
            if (!scheduledExecutorService.awaitTermination(10,
TimeUnit.SECONDS)) {
                scheduledExecutorService.shutdownNow();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}


```

然后，我们实现YangJobThreadFactory，完成对线程的命名

```
public class YangJobThreadFactory implements ThreadFactory {
    private String poolName;

    private String threadPrefixName;

    private static AtomicInteger poolNumber = new AtomicInteger(1);

    private AtomicInteger threadNumber = new AtomicInteger(1);
```

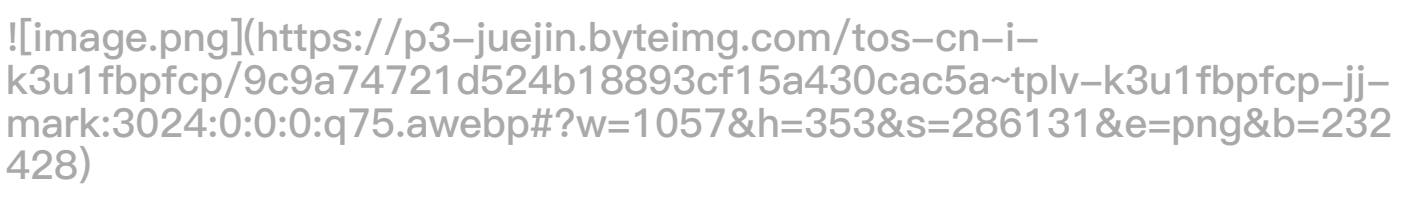
```
    public YangJobThreadFactory(String poolName) {
        this.poolName = poolName;
        this.threadPrefixName = poolName + "-pool-" +
poolNumber.getAndIncrement() + "-thread-";
    }

    public String getPoolName() {
        return this.poolName;
    }

    @Override
    public Thread newThread(Runnable r) {
        Thread thread = new Thread(r);
        thread.setName(this.threadPrefixName +
threadNumber.getAndIncrement());
        return thread;
    }

}
```

然后我们添加测试方法：

```
 public static void main(String[] args) {
        ThreadFactory threadFactory = new
YangJobThreadFactory("yang");
        ScheduledExecutorService scheduledExecutorService =
Executors.newScheduledThreadPool(4, threadFactory);
        YangJobManager yangJobManager = new
YangJobManager(scheduledExecutorService, threadFactory);

        yangJobManager.schedule(() -> {
            System.out.println(Thread.currentThread().getName() +
"schedule定时任务开始执行：" + new Date());
        }, 1L);

        yangJobManager.scheduleWithFixedDelay(() -> {
            System.out.println(Thread.currentThread().getName() +
"withFixedDelay定时任务开始执行：" + new Date());
        }, 0L, 1L);

        yangJobManager.scheduleWithFixedRate(() -> {
```

```
            System.out.println(Thread.currentThread().getName() +
”withFixedRate定时任务开始执行：” + new Date());
        }, 0L, 1L);

        try {
            Thread.sleep(20000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        yangJobManager.shutdown();
    }

```

执行结果如下：



提供统一的schedule方法

虽然我们能顺利将任务提交给YangJobManager执行，当感觉还不够收敛，因为我们创建了三个方法：schedule，scheduleWithFixedDelay，shceduleWithFixedRate，每个方法执行逻辑都差不多，最后都是调用scheduledExecutorService的相关方法，我们可以将这些方法都收敛到一个入口——schedule，然后在入参中添加一个参数，表示要执行的策略，根据入参的参数，选择对应的方法执行。
首先，我们添加一个执行策略枚举：

```
package com.yang.job.enums;

public enum JobExecuteStrategyEnum {
    IMMEDIATE_EXECUTE(”immediate”, ”立即执行”),
    ONCE(”once”, ”执行一次”),
    WITH_FIXED_DELAY(”withFixedDelay”, ”任务执行完毕后间隔执行”),
    WITH_FIXED_RATE(”withFixedRate”, ”任务执行开始后间隔执行”);

    private String name;

    private String description;
```

```
    JobExecuteStrategyEnum(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String getName() {
        return this.name;
    }

    public static JobExecuteStrategyEnum
getJobExecuteStrategyByName(String name) {
        if (name == null) {
            return null;
        }
        for (JobExecuteStrategyEnum value : values()) {
            if (name.equals(value.getName())) {
                return value;
            }
        }
        return null;
    }

    public static boolean isLegal(String name) {
        JobExecuteStrategyEnum jobExecuteStrategyByName =
getJobExecuteStrategyByName(name);
        return jobExecuteStrategyByName != null;
    }

    public String getDescription() {
        return description;
    }
}
```

然后添加YangJobManager的schedule方法的入参类：

```
package com.yang.job.request;

import com.yang.job.enums.JobExecuteStrategyEnum;
import lombok.Data;

import java.io.Serializable;
```

```
@Data
public class YangJobSubmitParam implements Serializable {
    private Runnable runnable;

    private Integer initialDelay;

    private Integer period;

    private JobExecuteStrategyEnum jobExecuteStrategy;
}
```

最后，修改YangJobManager类，将执行定时任务收敛到schedule方法，进入该方法，首先根据入参判断执行策略，如果是immediate，那么直接对入参的runnable调用run方法执行接口，其他的策略则分别对应scheduledExecutorService的schedule、scheduledWithFixedDelay、scheduledWithFixedRate方法，此外，这里对属性也进行修改，去除ThreadFactory属性。

```
package com.yang.job;

import com.yang.job.enums.JobExecuteStrategyEnum;
import com.yang.job.request.YangJobSubmitParam;

import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public class YangJobManager {
    private ScheduledExecutorService scheduledExecutorService;


    public YangJobManager(ScheduledExecutorService
scheduledExecutorService) {
        this.scheduledExecutorService = scheduledExecutorService;
    }

    public void schedule(YangJobSubmitParam yangJobSubmitParam) {
        JobExecuteStrategyEnum jobExecuteStrategy =
yangJobSubmitParam.getJobExecuteStrategy();
        if (jobExecuteStrategy == null) {
            throw new RuntimeException("缺少执行策略=========");
        }
        Runnable runnable = yangJobSubmitParam.getRunnable();
```

```
        Integer initialDelay = yangJobSubmitParam.getInitialDelay();
        Integer period = yangJobSubmitParam.getPeriod();
        switch (jobExecuteStrategy) {
            case IMMEDIATE_EXECUTE:
                runnable.run();
                break;
            case ONCE:
                scheduledExecutorService.schedule(runnable, initialDelay,
TimeUnit.SECONDS);
                break;
            case WITH_FIXED_DELAY:
                scheduledExecutorService.scheduleWithFixedDelay(runnable,
initialDelay, period, TimeUnit.SECONDS);
                break;
            case WITH_FIXED_RATE:
                scheduledExecutorService.scheduleAtFixedRate(runnable,
initialDelay, period, TimeUnit.SECONDS);
                break;
        }
    }

    public void shutdown() {
        if (this.scheduledExecutorService == null) {
            return;
        }
        if (this.scheduledExecutorService.isShutdown()) {
            return;
        }
        scheduledExecutorService.shutdown();
        try {
            if (!scheduledExecutorService.awaitTermination(10,
TimeUnit.SECONDS)) {
                scheduledExecutorService.shutdownNow();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}


```

最后，我们添加测试方法：

```
```

```
public static void main(String[] args) {
        ThreadFactory threadFactory = new
YangJobThreadFactory("yang");
        ScheduledExecutorService scheduledExecutorService =
Executors.newScheduledThreadPool(4, threadFactory);
        YangJobManager yangJobManager = new
YangJobManager(scheduledExecutorService);

        YangJobSubmitParam yangJobSubmitParam1 = new
YangJobSubmitParam();
        yangJobSubmitParam1.setRunnable(() -> System.out.println("立即
执行======" + new Date()));
yangJobSubmitParam1.setJobExecuteStrategy(JobExecuteStrategyEnum
.IMMEDIATE_EXECUTE);

        YangJobSubmitParam yangJobSubmitParam2 = new
YangJobSubmitParam();
        yangJobSubmitParam2.setRunnable(() -> System.out.println("执行
一次======" + new Date()));
        yangJobSubmitParam2.setInitialDelay(1);
yangJobSubmitParam2.setJobExecuteStrategy(JobExecuteStrategyEnum
.ONCE);

        YangJobSubmitParam yangJobSubmitParam3 = new
YangJobSubmitParam();
        yangJobSubmitParam3.setRunnable(() ->
System.out.println("withFixedDelay=====" + new Date()));
        yangJobSubmitParam3.setInitialDelay(1);
        yangJobSubmitParam3.setPeriod(2);
yangJobSubmitParam3.setJobExecuteStrategy(JobExecuteStrategyEnum
.WITH_FIXED_DELAY);

        YangJobSubmitParam yangJobSubmitParam4 = new
YangJobSubmitParam();
        yangJobSubmitParam4.setRunnable(() ->
System.out.println("withFixedRate=====" + new Date()));
        yangJobSubmitParam4.setInitialDelay(1);
        yangJobSubmitParam4.setPeriod(2);
yangJobSubmitParam4.setJobExecuteStrategy(JobExecuteStrategyEnum
.WITH_FIXED_RATE);

        yangJobManager.schedule(yangJobSubmitParam1);
        yangJobManager.schedule(yangJobSubmitParam2);
        yangJobManager.schedule(yangJobSubmitParam3);
        yangJobManager.schedule(yangJobSubmitParam4);

        try {
            Thread.sleep(20000);
```

```
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        yangJobManager.shutdown();
    }

```

执行结果如下：

提交任务和取消任务

任务的提交对应的是schedule方法，但我们的YangJobManager类缺少了关于任务的取消逻辑。在ScheduledExecutorService的各个定时执行方法中，其返回值是一个ScheduleFuture类，我们可以通过该类的cancel方法，来将对应的线程任务进行取消。此外，对于每一个任务，我们需要有一个任务标识，所以，我们先修改YangJobSubmitParam类：

```
package com.yang.job.request;

import com.yang.job.enums.JobExecuteStrategyEnum;
import lombok.Data;

import java.io.Serializable;

@Data
public class YangJobSubmitParam implements Serializable {
    private Integer jobId;

    private Runnable runnable;

    private Integer initialDelay;

    private Integer period;

    private JobExecuteStrategyEnum jobExecuteStrategy;
}
```

```

然后，我们修改YangJobManager类，首先将schedule方法改为submit方法，这样更见名知义，在submit方法中，除了理解执行策略外，其他策略都会获取返回的ScheduleFuture，然后存入对应的map，在取消的时候，我们根据jobId从map中找到对应的ScheduleFuture，并执行cancel方法，以此来取消任务。

```
package com.yang.job;

import com.yang.job.enums.JobExecuteStrategyEnum;
import com.yang.job.request.YangJobSubmitParam;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;

public class YangJobManager {
    private ScheduledExecutorService scheduledExecutorService;

    private Map<String, ScheduledFuture> jobId2ScheduleFutureMap =
new ConcurrentHashMap<>();

    public YangJobManager(ScheduledExecutorService
scheduledExecutorService) {
        this.scheduledExecutorService = scheduledExecutorService;
    }

    public void submitJob(YangJobSubmitParam yangJobSubmitParam) {
        Integer jobId = yangJobSubmitParam.getJobId();
        if (jobId == null) {
            throw new RuntimeException("缺少任务标识==========");
        }
        ScheduledFuture scheduledFuture =
jobId2ScheduleFutureMap.get(jobId.toString());
        if (scheduledFuture != null && !scheduledFuture.isCancelled()) {
            // jobId存在对应的任务
            return;
        }

        JobExecuteStrategyEnum jobExecuteStrategy =
```
```

```
yangJobSubmitParam.getJobExecuteStrategy();
        if (jobExecuteStrategy == null) {
            throw new RuntimeException("缺少执行策略=========");
        }

        if (jobExecuteStrategy ==
JobExecuteStrategyEnum.IMMEDIATE_EXECUTE) {
            yangJobSubmitParam.getRunnable().run();
            return;
        }
        scheduledFuture = scheduleJob(yangJobSubmitParam);
        jobId2ScheduleFutureMap.put(jobId.toString(), scheduledFuture);
    }

    public void cancelJob(Integer jobId) {
        if (jobId == null) {
            return;
        }
        ScheduledFuture scheduledFuture =
jobId2ScheduleFutureMap.get(jobId.toString());
        if (scheduledFuture == null) {
            return;
        }
        if (!scheduledFuture.isCancelled()) {
            scheduledFuture.cancel(true);
        }
        jobId2ScheduleFutureMap.remove(jobId.toString());
    }

    private ScheduledFuture scheduleJob(YangJobSubmitParam
yangJobSubmitParam) {
        Runnable runnable = yangJobSubmitParam.getRunnable();
        Integer initialDelay = yangJobSubmitParam.getInitialDelay();
        Integer period = yangJobSubmitParam.getPeriod();
        JobExecuteStrategyEnum jobExecuteStrategy =
yangJobSubmitParam.getJobExecuteStrategy();
        switch (jobExecuteStrategy) {
            case ONCE:
                return scheduledExecutorService.schedule(runnable,
initialDelay, TimeUnit.SECONDS);
            case WITH_FIXED_DELAY:
                return
scheduledExecutorService.scheduleWithFixedDelay(runnable, initialDelay,
period, TimeUnit.SECONDS);
            case WITH_FIXED_RATE:
                return
scheduledExecutorService.scheduleAtFixedRate(runnable, initialDelay,
period, TimeUnit.SECONDS);
```

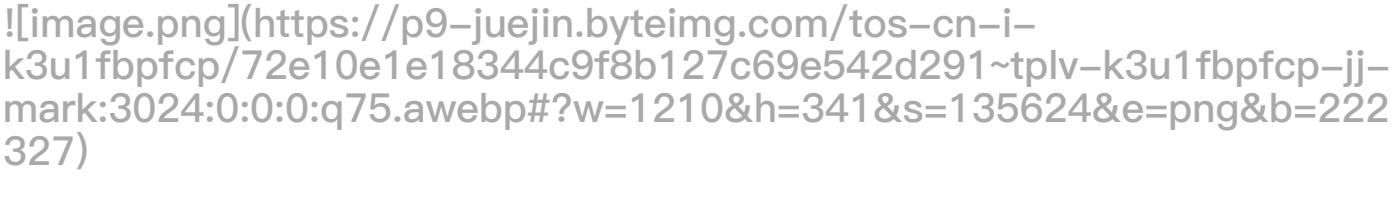
```
        }
        throw new RuntimeException("执行策略有误==========");
    }

    public void shutdown() {
        if (this.scheduledExecutorService == null) {
            return;
        }
        if (this.scheduledExecutorService.isShutdown()) {
            return;
        }
        scheduledExecutorService.shutdown();
        try {
            if (!scheduledExecutorService.awaitTermination(10,
TimeUnit.SECONDS)) {
                scheduledExecutorService.shutdownNow();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}


```

最后，我们添加对应的测试方法：

```
 public static void main(String[] args) {
        ThreadFactory threadFactory = new
YangJobThreadFactory("yang");
        ScheduledExecutorService scheduledExecutorService =
Executors.newScheduledThreadPool(4, threadFactory);

        YangJobManager yangJobManager = new
YangJobManager(scheduledExecutorService);
        YangJobSubmitParam yangJobSubmitParam = new
YangJobSubmitParam();
        yangJobSubmitParam.setJobId(1);
        yangJobSubmitParam.setRunnable(() -> System.out.println("执行任
务=====" + new Date()));
        yangJobSubmitParam.setInitialDelay(0);
        yangJobSubmitParam.setPeriod(2);
yangJobSubmitParam.setJobExecuteStrategy(JobExecuteStrategyEnum.
WITH_FIXED_RATE);
```

```
        yangJobManager.submitJob(yangJobSubmitParam);

        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("取消任务==========");
        yangJobManager.cancelJob(1);
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        yangJobManager.shutdown();

    }
```

在该方法中，我们提交任务，该任务间隔时间为2秒，10秒过后，取消任务，取消任务过后，再睡眠10秒，在后面10秒钟，不会执行任务（或执行一次，因为在cancel之前刚好有任务没执行完），执行结果如下：



YangJobManager建造者

对于YangJobManager，目前我们所拥有的属性、方法都比较简单，但是如果后续这个类进一步扩展，构造该类可能会变得很麻烦，因此，我们添加一个YangJobBuilder建造者类，用于构造YangJobManager，此外，我们将YangJobManager的构造方法设置为private，从而将构造YangJobManager的职责，彻底收敛到YangJobManagerBuilder类中，我们修改YangJobManager类如下：

```
package com.yang.job;

import com.yang.job.enums.JobExecuteStrategyEnum;
import com.yang.job.factory.YangJobThreadFactory;
```

```
import com.yang.job.request.YangJobSubmitParam;

import java.util.Map;
import java.util.concurrent.*;

public class YangJobManager {
    private ScheduledExecutorService scheduledExecutorService;

    private Map<String, ScheduledFuture> jobId2ScheduleFutureMap =
new ConcurrentHashMap<>();

    private YangJobManager(ScheduledExecutorService
scheduledExecutorService) {
        this.scheduledExecutorService = scheduledExecutorService;
    }

    public void submitJob(YangJobSubmitParam yangJobSubmitParam) {
        Integer jobId = yangJobSubmitParam.getJobId();
        if (jobId == null) {
            throw new RuntimeException("缺少任务标识=========");
        }
        ScheduledFuture scheduledFuture =
jobId2ScheduleFutureMap.get(jobId.toString());
        if (scheduledFuture != null && !scheduledFuture.isCancelled()) {
            // jobId存在对应的任务
            return;
        }

        JobExecuteStrategyEnum jobExecuteStrategy =
yangJobSubmitParam.getJobExecuteStrategy();
        if (jobExecuteStrategy == null) {
            throw new RuntimeException("缺少执行策略=========");
        }

        if (jobExecuteStrategy ==
JobExecuteStrategyEnum.IMMEDIATE_EXECUTE) {
            yangJobSubmitParam.getRunnable().run();
            return;
        }
        scheduledFuture = scheduleJob(yangJobSubmitParam);
        jobId2ScheduleFutureMap.put(jobId.toString(), scheduledFuture);
    }

    public void cancelJob(Integer jobId) {
        if (jobId == null) {
            return;
        }
        ScheduledFuture scheduledFuture =
```

```
jobId2ScheduleFutureMap.get(jobId.toString());
        if (scheduledFuture == null) {
            return;
        }
        if (!scheduledFuture.isCancelled()) {
            scheduledFuture.cancel(true);
        }
        jobId2ScheduleFutureMap.remove(jobId.toString());
    }

    private ScheduledFuture scheduleJob(YangJobSubmitParam
yangJobSubmitParam) {
        Runnable runnable = yangJobSubmitParam.getRunnable();
        Integer initialDelay = yangJobSubmitParam.getInitialDelay();
        Integer period = yangJobSubmitParam.getPeriod();
        JobExecuteStrategyEnum jobExecuteStrategy =
yangJobSubmitParam.getJobExecuteStrategy();
        switch (jobExecuteStrategy) {
            case ONCE:
                return scheduledExecutorService.schedule(runnable,
initialDelay, TimeUnit.SECONDS);
            case WITH_FIXED_DELAY:
                return
scheduledExecutorService.scheduleWithFixedDelay(runnable, initialDelay,
period, TimeUnit.SECONDS);
            case WITH_FIXED_RATE:
                return
scheduledExecutorService.scheduleAtFixedRate(runnable, initialDelay,
period, TimeUnit.SECONDS);
        }
        throw new RuntimeException("执行策略有误==========");
    }

    public void shutdown() {
        if (this.scheduledExecutorService == null) {
            return;
        }
        if (this.scheduledExecutorService.isShutdown()) {
            return;
        }
        scheduledExecutorService.shutdown();
        try {
            if (!scheduledExecutorService.awaitTermination(10,
TimeUnit.SECONDS)) {
                scheduledExecutorService.shutdownNow();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
```

```
        }
    }

    public static class YangJobManagerBuilder {
        private ThreadFactory threadFactory;

        private ScheduledExecutorService scheduledExecutorService;

        public YangJobManagerBuilder() {
        }

        public YangJobManagerBuilder setThreadFactory(ThreadFactory
threadFactory) {
            this.threadFactory = threadFactory;
            return this;
        }

        public YangJobManagerBuilder
setScheduledExecutorService(ScheduledExecutorService
scheduledExecutorService) {
            this.scheduledExecutorService = scheduledExecutorService;
            return this;
        }

        public YangJobManager build() {
            if (this.threadFactory == null) {
                this.threadFactory = new YangJobThreadFactory("yang");
            }
            if (this.scheduledExecutorService == null) {
                this.scheduledExecutorService =
Executors.newScheduledThreadPool(Runtime.getRuntime().availableProc
essors(),
                        this.threadFactory);
            } else {
                if (this.scheduledExecutorService instanceof
ScheduledThreadPoolExecutor) {
                    ScheduledThreadPoolExecutor
scheduledThreadPoolExecutor = (ScheduledThreadPoolExecutor)
this.scheduledExecutorService;
scheduledThreadPoolExecutor.setThreadFactory(this.threadFactory);
                }
            }
            return new YangJobManager(this.scheduledExecutorService);
        }
    }
}
```

```

### 任务执行类

在之前的代码中，我们的Runnable都是匿名函数类，但是在我们的定时任务调度平台中，一般情况下，这个任务是会持久化到数据库中的，我们一般不会说把这个Runnable的代码也存到数据库吧，一般存储的，应该就是某个任务执行类的类路径，和方法名，以及入参，然后在启动项目时，从数据库中加载这些数据，并通过反射或代理等方式，来构造这个Runnable。
首先，我们定义一个任务执行类，来规范任务的执行方法和入参格式：

```
// 任务执行类
package com.yang.job.execute;

public interface IYangJobExecutor {
    void execute(YangJobExecuteRequest yangJobExecuteRequest);
}

// 任务执行方法入参
package com.yang.job.execute;

import lombok.Data;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;

@Data
public class YangJobExecuteRequest implements Serializable {
    private String jobId;

    private Map<String, String> params = new HashMap<>();

    public void addParam(String key, String value) {
        params.put(key, value);
    }

    public String getParam(String key) {
        return params.get(key);
    }
}
```
```

接着，我们创建这个YangJobExecutor的实现类，用于测试，在该类中，执行任务的方法很简单，打印当前类的名字以及入参。

```
package com.yang.task;

import com.yang.job.execute.IYangJobExecutor;
import com.yang.job.execute.YangJobExecuteRequest;

import java.util.Date;

public class TestJobExecutor implements IYangJobExecutor {
    @Override
    public void execute(YangJobExecuteRequest
yangJobExecuteRequest) {
        System.out.println(String.format("%s 任务执行类执行了，入参为
: %s, 当前时间:%s",
                this.getClass().getName(), yangJobExecuteRequest.toString(),
                new Date().toString()));
    }
}
```

然后我们创建一个YangJobData，假设我们从数据库中获取的数据格式如下：

```
package com.yang.job.data;

import lombok.Data;

import java.io.Serializable;

@Data
public class YangJobData implements Serializable {
    private Integer jobId;

    private String cron;

    private String executeStrategy;

    private String executeClassPath;
```

```
    private String executeParams;
}
```

executeStrategy表示任务的执行策略，executeClassPath表示要执行的任务类的路径，executeParams表示执行任务方法的入参。
在XXL-JOB中，我们可以使用cron来设置定时任务的执行时间，因此我们这里，也使用cron作为定时任务的执行时间设置，为了解析cron表达式，我们添加下列依赖：

```
  <dependency>
            <groupId>com.cronutils</groupId>
            <artifactId>cron-utils</artifactId>
            <version>9.2.0</version>
        </dependency>
```

然后创建一个CronUtils工具类，用于解析cron表达式。

```
package com.yang.demo.infra.utils;

import com.cronutils.model.CronType;
import com.cronutils.model.definition.CronDefinition;
import com.cronutils.model.definition.CronDefinitionBuilder;
import com.cronutils.model.time.ExecutionTime;
import com.cronutils.parser.CronParser;

import java.time.ZonedDateTime;
import java.util.Optional;

public class CronUtils {
    private static final CronDefinition CRON_DEFINITION =
CronDefinitionBuilder.instanceDefinitionFor(CronType.QUARTZ);
    private static final CronParser CRON_PARSER = new
CronParser(CRON_DEFINITION);

    public static ZonedDateTime nextExecutionTime(String cron,
ZonedDateTime startTime) {
        ExecutionTime executionTime =
```

```
ExecutionTime.forCron(CRON_PARSER.parse(cron));
        Optional<ZonedDateTime> zonedDateTime =
executionTime.nextExecution(startTime);
        return zonedDateTime.get();
    }
}
```

对于执行方法的入参，一般情况下，就是任务的id，以及一些扩展信息，这些扩展信息一般以键值对的形式存储，即”key:value;key:value;”这些形式，所以这里添加一个FeaturesUtils类，用于解析这些键值对信息：

```
package com.yang.job.utils;


import java.util.HashMap;
import java.util.Map;

public class FeaturesUtils {
    private final static String KEY_KEY_SEPARATOR = ”;”;
    private final static String KEY_VALUE_SEPARATOR = ”:”;

    public static Map<String, String> convert2FeatureMap(String features)
{
        Map<String, String> featureMap = new HashMap<>();
        if (features == null || features.isEmpty()) {
            return featureMap;
        }
        String[] keyValues = features.split(KEY_KEY_SEPARATOR);
        for (String keyValue : keyValues) {
            String[] split = keyValue.split(KEY_VALUE_SEPARATOR);
            String key = split[0];
            String value = split[1];
            featureMap.put(key, value);
        }
        return featureMap;
    }

    public static String convert2Features(Map<String, String> featureMap)
{
        if (featureMap == null || featureMap.isEmpty()) {
            return ””;
        }
        StringBuilder stringBuilder = new StringBuilder();
```

```
        featureMap.forEach((key, value) -> {
            stringBuilder.append(key)
                    .append(KEY_VALUE_SEPARATOR)
                    .append(value)
                    .append(KEY_KEY_SEPARATOR);
        });
        return stringBuilder.toString();
    }
}


```

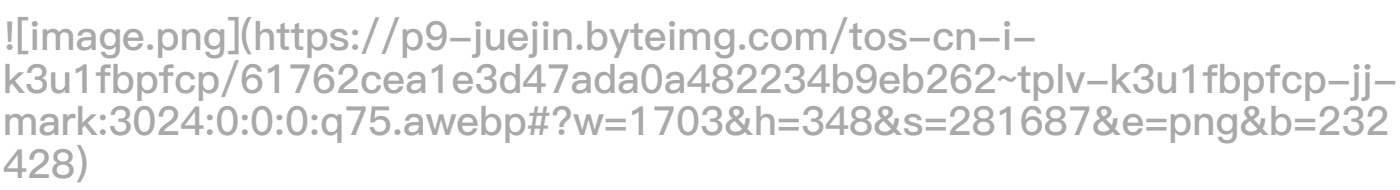
然后我们添加测试方法，模拟从数据库中获取数据，并根据任务类路径，获取对应的runnable并提交到YangJobManager中。

```
  public static void main(String[] args) {
        YangJobData yangJobData = mockYangJobData();
        YangJobSubmitParam yangJobSubmitParam =
convert2YangJobSubmitParam(yangJobData);

        YangJobManager yangJobManager = new
YangJobManager.YangJobManagerBuilder()
                .setThreadFactory(new YangJobThreadFactory("yang"))
                .build();
        yangJobManager.submitJob(yangJobSubmitParam);

        try {
            Thread.sleep(20000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        yangJobManager.shutdown();
    }

    private static YangJobSubmitParam
convert2YangJobSubmitParam(YangJobData yangJobData) {
        YangJobSubmitParam yangJobSubmitParam = new
YangJobSubmitParam();
        yangJobSubmitParam.setJobId(yangJobData.getJobId());
yangJobSubmitParam.setJobExecuteStrategy(JobExecuteStrategyEnum.
getJobExecuteStrategyByName(yangJobData.getExecuteStrategy()));
        ZonedDateTime nextExecutionTime =
CronUtils.nextExecutionTime(yangJobData.getCron(),
ZonedDateTime.now());
```

```
        ZonedDateTime nextNextExecutionTime =
CronUtils.nextExecutionTime(yangJobData.getCron(),
nextExecutionTime);
        long nowEochMill = ZonedDateTime.now().toInstant().toEpochMilli();
        long executeEochMill =
nextExecutionTime.toInstant().toEpochMilli();
        long secondExecuteEochMill =
nextNextExecutionTime.toInstant().toEpochMilli();
        yangJobSubmitParam.setInitialDelay((int)(executeEochMill -
nowEochMill) / 1000);
        yangJobSubmitParam.setPeriod((int)(secondExecuteEochMill -
executeEochMill) / 1000);

        try {
            Class<?> aClass =
Class.forName(yangJobData.getExecuteClassPath());
            if (!IYangJobExecutor.class.isAssignableFrom(aClass)) {
                throw new RuntimeException("任务类必须实现
IYangJobExecutor接口");
            }
            IYangJobExecutor executor = (IYangJobExecutor)
aClass.newInstance();
            YangJobExecuteRequest yangJobExecuteRequest =
convert2YangJobExecuteRequest(yangJobData);
            Runnable runnable = () ->
executor.execute(yangJobExecuteRequest);
            yangJobSubmitParam.setRunnable(runnable);
        } catch (InstantiationException | IllegalAccessException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return yangJobSubmitParam;
    }

    private static YangJobExecuteRequest
convert2YangJobExecuteRequest(YangJobData yangJobData) {
        YangJobExecuteRequest yangJobExecuteRequest = new
YangJobExecuteRequest();
yangJobExecuteRequest.setJobId(yangJobData.getJobId().toString());
yangJobExecuteRequest.setParams(FeaturesUtils.convert2FeatureMap(y
angJobData.getExecuteParams()));
        return yangJobExecuteRequest;
    }

    private static YangJobData mockYangJobData() {
        YangJobData yangJobData = new YangJobData();
        yangJobData.setJobId(1);
```

```
        yangJobData.setCron("0/5 * * * * ?");
yangJobData.setExecuteStrategy(JobExecuteStrategyEnum.WITH_FIXED_DELAY.getName());
yangJobData.setExecuteClassPath("com.yang.task.TestJobExecutor");
    yangJobData.setExecuteParams("jobId:1;startIndex:1;endIndex:10;");
        return yangJobData;
    }

```

这里对于cron的解析，其实不是特别好，这里的思路是，获取下一次执行的时间，和下下一次执行的时间，然后以此来计算initialDelay和period，但是如果这个cron表示的是某几天、某几个小时，比如说星期一、星期二、星期三执行，那么我们那种解析方式是有误的，这个可以后续再好好斟酌一下，目前先这样解析。
执行结果如下：



原文链接: https://juejin.cn/post/7368477662375428123