链滴

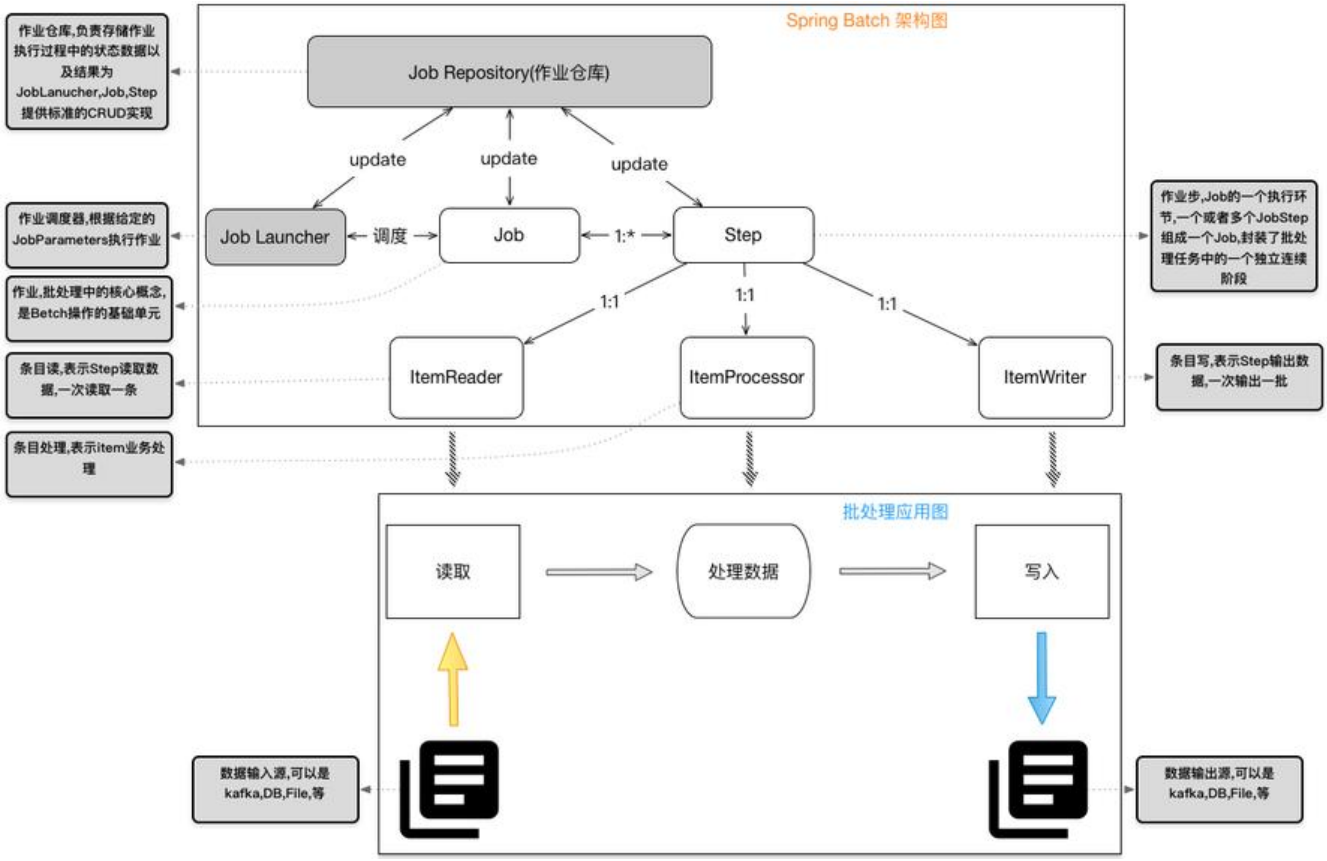
Spring Batch (二)- 搭建篇

作者：adongs

原文链接：https://ld246.com/article/1567153739105

来源网站：链滴

架构图



概念补充

对象	描述

原文链接：Spring Batch (二)- 搭建篇

Job Instance 作业实例,每个作业执行时,都会产生个新的实例,实例被放在JobRepository中,如果作业失败,下次重新执行该作业,会使用同一个作业实例,Jb 和Job Instance 的关系可以认为 new Job = Job Instance

Job Parameters 作业参数,是用来启动批处理任的参数,在Job启动的时候,可以设置任何需要作业的参数,需要注意的是作业参数会用来做标识作业实例不同的Job实例通过不同的参数来区分

Job Execution 作业执行器,具体负责Job的执行,次运行Job都会启动一个新的job执行器

Step Execution 作业步执行器,他负责具体Step行,每次运行Step都会齐东东一个新的执行器

Tasklet Step中具体的路基操作,可以设置同步,异重复执行等操作

Execution Context 执行上下文,他是一组矿建持化预控的Key/value对,能够让开发者在Step Execution或者Job Execution 范畴保存需要进行持久化状态

Item 条目,一条数据记录

Chunk Item集合,他给定数量Item集合,可以定义Chunk的读,处理,写操作提交间隔等

领域对象详解

Job(接口)

Job 接口方法说明

- getName() 获取Job名称
- isRestartable() 是否可以重启作业,返回true表示可以重启
- execute(JobExecution execution) 运行JobExecution
- getJobParametersIncrementer() 获取Job Parameters实例
- getJobParametersValidator() 获取JobParametersValidator(参数验证器)实例

Step(接口)

Step 接口方法说明

- getName() 获取Step名称
- execute(StepExecution stepExecution) 执行StepExecution,并将元信息保存在StepExecution中
- getStartLimit() 可以启动次数
- isAllowStartIfComplete() 返回true 表示可以再次启动

ItemReader(接口)

ItemReader接口方法说明

- read() 读取数据返回模型

ItemProcessor(接口)

ItemProcessor接口方法说明

- process(I item) 处理模型

ItemWriter(接口)

ItemWriter接口方法说明

- write(List<? extends T> items) 写入数据

JobRepository(接口)

JobRepository接口方法说明

- isJobInstanceExists(String jobName, JobParameters jobParameters)
- createJobInstance(String jobName, JobParameters jobParameters)
- createJobExecution(JobInstance jobInstance, JobParameters jobParameters, String jobConfi urationLocation)
- createJobExecution(String jobName, JobParameters jobParameters)
- update(JobExecution jobExecution)
- add(StepExecution stepExecution)
- addAll(Collection <StepExecution> stepExecutions)
- update(StepExecution stepExecution)
- updateExecutionContext(StepExecution stepExecution)
- updateExecutionContext(JobExecution jobExecution)
- getLastStepExecution(JobInstance jobInstance, String stepName)
- getStepExecutionCount(JobInstance jobInstance, String stepName)
- getLastJobExecution(String jobName, JobParameters jobParameters);

JobExplorer(接口)

JobExplorer接口方法说明

- getJobInstances(String jobName, int start, int count)
- getJobExecution(@Nullable Long executionId)
- getStepExecution(@Nullable Long jobExecutionId, @Nullable Long stepExecutionId)
- getJobInstance(@Nullable Long instanceId)
- getJobExecutions(JobInstance jobInstance)
- findRunningJobExecutions(@Nullable String jobName)
- getJobNames()
- findJobInstancesByJobName(String jobName, int start, int count)

- getJobInstanceCount(@Nullable String jobName)

JobLauncher(接口)

JobLauncher接口方法说明

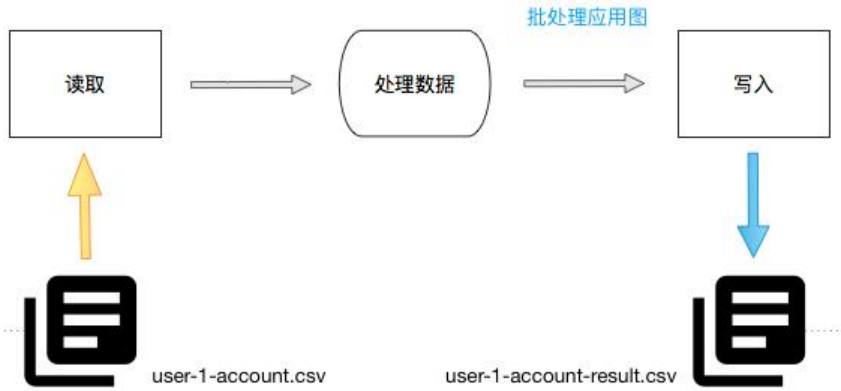
- run(Job job, JobParameters jobParameters)

搭建

创建项目(内存版本)

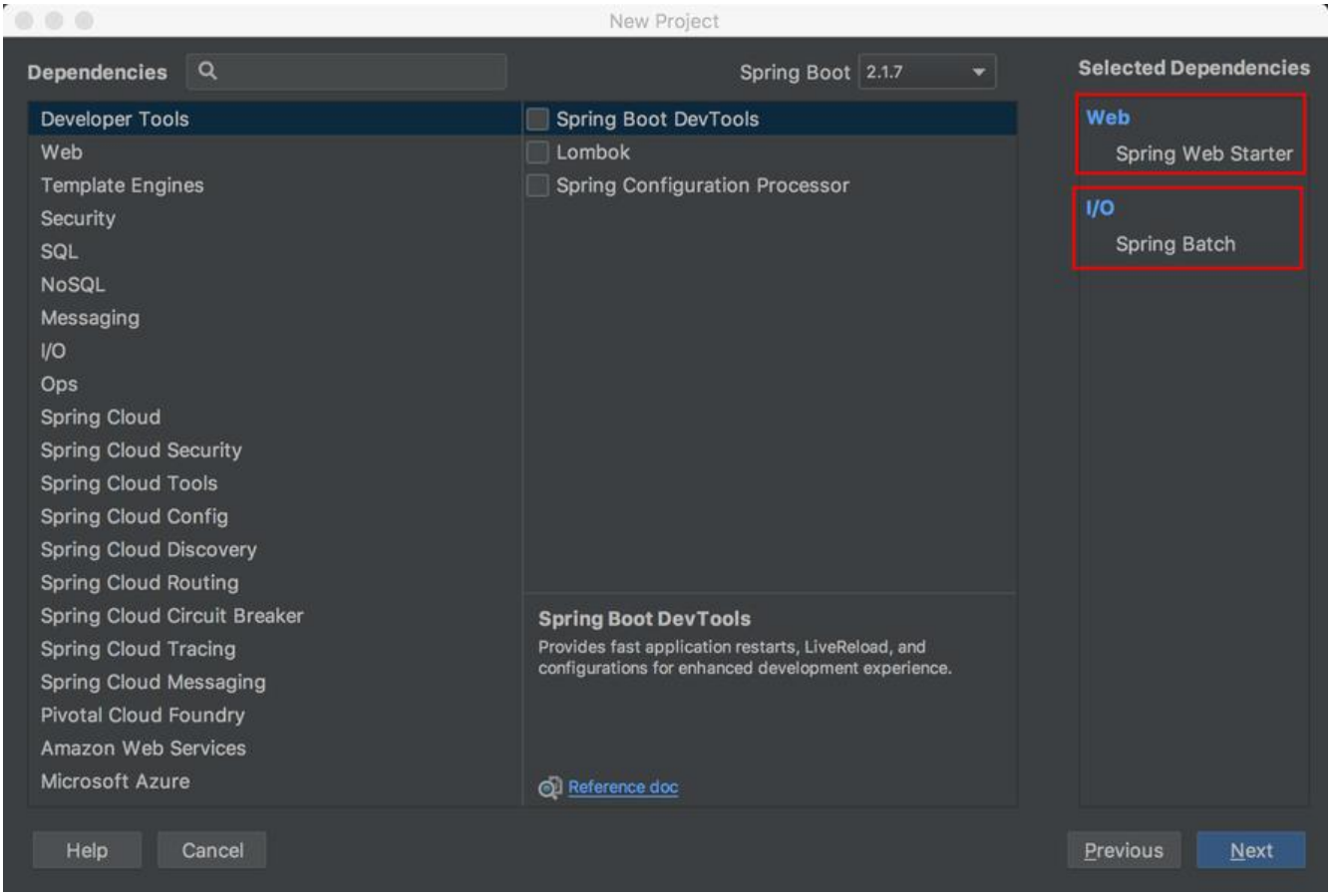
0.任务流程

使用spring bacth 读取user-1-account.csv文件的内容
处理完成写入到user-1-account-result.csv文件中



1.使用开发工具(这里笔者使用IDEA)创建项目

2.选择如下



3.修改pom文件如下

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
MLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd
maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.7.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.adongs</groupId>
    <artifactId>spring-batch</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring-batch</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-batch</artifactId>
```

```
            <exclusions>
              <! -- 去掉jdbc,不然会使用数据模式 -->
               <exclusion>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-jdbc</artifactId>
               </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
      <!-- 添加测试组件 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.batch</groupId>
            <artifactId>spring-batch-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

4.关闭job自动启动

```
spring:
  batch:
    job:
      enabled: false
```

5.配置MapBatchConfig(基于内存的Batch Config),需要实现BatchConfigurer接口

```
package com.adongs.springbatch;
```

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.batch.core.configuration.BatchConfigurationException;
import org.springframework.batch.core.configuration.annotation.BatchConfigurer;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.configuration.support.MapJobRegistry;
import org.springframework.batch.core.explore.JobExplorer;
import org.springframework.batch.core.explore.support.MapJobExplorerFactoryBean;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.launch.support.SimpleJobLauncher;
import org.springframework.batch.core.repository.JobRepository;
import org.springframework.batch.core.repository.support.MapJobRepositoryFactoryBean;
import org.springframework.batch.support.transaction.ResourcelessTransactionManager;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.transaction.PlatformTransactionManager;

import javax.annotation.PostConstruct;

/**
 * 基于内存处理
 * @Author yudong
 * @Date 2019/8/16 上午10:09
 * @Version 1.0
 */
@Configuration
public class MapBatchConfig implements BatchConfigurer {

    private static final Log logger = LogFactory.getLog(MapBatchConfig.class);

    /**
     * 事务管理器
     */
    private PlatformTransactionManager transactionManager = new ResourcelessTransaction
anager();
    private JobRepository jobRepository;
    private JobLauncher jobLauncher;
    private JobExplorer jobExplorer;
    private MapJobRepositoryFactoryBean factoryBean;

    public MapBatchConfig() {}


    @Override
    @Bean
    public JobRepository getJobRepository() {
        return jobRepository;
    }

    @Bean
    public MapJobRegistry mapJobRegistry(){
        return new MapJobRegistry();
    }
```

```
/**
 * 作业构建工厂
 * @return
 */
@Bean
public JobBuilderFactory jobBuilderFactory(){
    return new  JobBuilderFactory(jobRepository);
}

/**
 * 作业步构建工厂
 * @return
 */
@Bean
public StepBuilderFactory stepBuilderFactory(){
    return new StepBuilderFactory(jobRepository,transactionManager);
}

@Override
@Bean
public PlatformTransactionManager getTransactionManager() {
    return transactionManager;
}

@Override
@Bean
public JobLauncher getJobLauncher() {
    return jobLauncher;
}

@Override
@Bean
public JobExplorer getJobExplorer() {
    return jobExplorer;
}

/**
 * 初始化项目
 */
@PostConstruct
public void initialize() {
    try {
        this.jobRepository = createJobRepository();
        this.jobExplorer = createJobExplorer();
        this.jobLauncher = createJobLauncher();
    } catch (Exception e) {
        throw new BatchConfigurationException(e);
    }
}

/**
 * 创建调度器
 * @return
```

```
     * @throws Exception
     */
    public JobLauncher createJobLauncher() throws Exception {
        SimpleJobLauncher jobLauncher = new SimpleJobLauncher();
        jobLauncher.setJobRepository(jobRepository);
        jobLauncher.afterPropertiesSet();
        return jobLauncher;
    }




    /**
     * 创建基于内存JobRepository
     * @return
     * @throws Exception
     */
    public JobRepository createJobRepository() throws Exception {
        factoryBean = new MapJobRepositoryFactoryBean();
        factoryBean.setTransactionManager(getTransactionManager());
        factoryBean.afterPropertiesSet();
        JobRepository jobRepository = factoryBean.getObject();
        return jobRepository;
    }

    /**
     * 创建基于内存的JobExplorer
     * @return
     * @throws Exception
     */
    public JobExplorer createJobExplorer() throws Exception {
        MapJobExplorerFactoryBean mapJobExplorerFactoryBean = new MapJobExplorerFactory
ean(factoryBean);
        mapJobExplorerFactoryBean.afterPropertiesSet();
        return mapJobExplorerFactoryBean.getObject();
    }
}
```

6.在resources文件夹下创建两个文件user-1-account.csv,user-1-account-result.csv,并在user-1-ac
ount.csv写入如下内容

```
小张,2019,50
小张,2018,100
小李,2016,20
```

7.创建账户实体

```
package com.adongs.springbatch;

import org.springframework.stereotype.Component;

import java.util.Date;
```

```java
/**
 * 账户实体
 * @Author yudong
 * @Date 2019/8/15 下午4:57
 * @Version 1.0
 */
@Component
public class Account {

    /**
     * 姓名
     */
    private String name;

    /**
     * 时间
     */
    private String date;

    /**
     * 金额
     */
    private int money;

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

    public void setName(String name) {
        this.name = name;
    }

    public String getDate() {
        return this.date;
    }

    public void setDate(String date) {
        this.date = date;
    }

    public int getMoney() {
        return this.money;
    }

    public void setMoney(int money) {
        this.money = money;
    }

    @Override
    public String toString() {
        return "Account{" +
                "name='" + name + '\'' +
                ", date=" + date +
                ", money=" + money +
```

```
                '}';
    }
}
```

8.配置job

```
package com.adongs.springbatch;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.FlatFileItemWriter;
import org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper;
import org.springframework.batch.item.file.mapping.DefaultLineMapper;
import org.springframework.batch.item.file.transform.BeanWrapperFieldExtractor;
import org.springframework.batch.item.file.transform.DelimitedLineAggregator;
import org.springframework.batch.item.file.transform.DelimitedLineTokenizer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;

/**
 * @Author yudong
 * @Date 2019/8/16 上午11:11
 * @Version 1.0
 */
@Configuration
public class JobConfig {


    @Autowired
    private JobBuilderFactory jobBuilderFactory;

    @Autowired
    private StepBuilderFactory stepBuilderFactory;
    @Autowired
    private ApplicationContext applicationContext;

    /**
     * 配置Job
     * @return
     */
    @Bean
    public Job footballJob(Step step){
        return this.jobBuilderFactory.get("job")
                .incrementer(new RunIdIncrementer())
```

```
            .listener(sampleListener())
            .start(step).build();
    }

    /**
     * 构建step
     * @param reader
     * @param processor
     * @param flatFileItemWriter
     * @return
     */
    @Bean
    public Step reconciliation(FlatFileItemReader reader,
                               CreditBillProcessor processor,
                               FlatFileItemWriter flatFileItemWriter){

        return  stepBuilderFactory.get("reconciliation")
                .<Account,Account> chunk(2)
                .reader(reader)
                .processor(processor)
                .writer(flatFileItemWriter)
                .build();
    }

    /**
     * 读取文件
     * @return
     */
    @Bean
    public FlatFileItemReader flatFileItemReader(){
        FlatFileItemReader flatFileItemReader = new FlatFileItemReader();
        flatFileItemReader.setResource(new ClassPathResource("user-1-account.csv"));
        DelimitedLineTokenizer delimitedLineTokenizer = new DelimitedLineTokenizer();
        delimitedLineTokenizer.setDelimiter(",");
        delimitedLineTokenizer.setNames("name","date","money");
        BeanWrapperFieldSetMapper beanWrapperFieldSetMapper = new BeanWrapperFieldSe
Mapper();
        beanWrapperFieldSetMapper.setBeanFactory(applicationContext);
        beanWrapperFieldSetMapper.setPrototypeBeanName("account");
        DefaultLineMapper defaultLineMapper = new DefaultLineMapper();
        defaultLineMapper.setLineTokenizer(delimitedLineTokenizer);
        defaultLineMapper.setFieldSetMapper(beanWrapperFieldSetMapper);
        flatFileItemReader.setLineMapper(defaultLineMapper);
        return flatFileItemReader;
    }

    /**
     * 处理器
     * @return
     */
    @Bean
    public ItemProcessor creditBillProcessor(){
        return new ItemProcessor<Account,Account>(){
            @Override
```

```
            public Account process(Account item) throws Exception {
                System.out.println(item.toString());
                return item;
            }
        };
    }

    /**
     * 写入文件
     * @return
     */
    @Bean
    public FlatFileItemWriter flatFileItemWriter(){
        BeanWrapperFieldExtractor beanWrapperFieldExtractor = new BeanWrapperFieldExtract
r();
        beanWrapperFieldExtractor.setNames(new String[]{"name","date","money"});
        DelimitedLineAggregator delimitedLineAggregator = new DelimitedLineAggregator();
        delimitedLineAggregator.setDelimiter(",");
        delimitedLineAggregator.setFieldExtractor(beanWrapperFieldExtractor);
        FlatFileItemWriter flatFileItemWriter = new FlatFileItemWriter();
        flatFileItemWriter.setResource(new ClassPathResource("user-1-account-result.csv"));
        flatFileItemWriter.setLineAggregator(delimitedLineAggregator);
        return flatFileItemWriter;
    }



    /**
     * 监听
     * @return
     */
    @Bean
    public JobExecutionListener sampleListener(){
        return new JobExecutionListener(){
            @Override
            public void beforeJob(JobExecution jobExecution) {
                System.out.println("JobExecutionListener.beforeJob");
            }

            @Override
            public void afterJob(JobExecution jobExecution) {
                System.out.println("JobExecutionListener.afterJob");
            }
        };
    }
}
```

8.编写测试

```
/**
 * @Author yudong
 * @Date 2019/8/15 下午7:34
 * @Version 1.0
```

```
 */
@RestController
public class TestRunController {


    @Autowired
    JobLauncher jobLauncher;

    @Autowired
    Job job;

    @GetMapping("test")
    public String test() throws JobParametersInvalidException, JobExecutionAlreadyRunningExc
ption, JobRestartException, JobInstanceAlreadyCompleteException {
        JobParameters jobParameters = new JobParametersBuilder().addDate("date",new Date()).
oJobParameters();
        jobLauncher.run(job,jobParameters);
        return "ok";
    }

}
```

测试

1.启动项目

2.请求连接:http://localhost:8080/test 输出如下

```
2019-08-30 16:20:42.139  INFO 57596 --- [nio-8080-exec-1] o.s.b.c.l.support.SimpleJobLaunc
er      : Job: [SimpleJob: [name=job]] launched with the following parameters: [{date=1567153
42078}]
JobExecutionListener.beforeJob
2019-08-30 16:20:42.153  INFO 57596 --- [nio-8080-exec-1] o.s.batch.core.job.SimpleStepHa
dler     : Executing step: [reconciliation]
Account{name='小张', date=2018, money=100}
Account{name='小张', date=2018, money=100}
Account{name='小李', date=2016, money=20}
JobExecutionListener.afterJob
2019-08-30 16:20:42.412  INFO 57596 --- [nio-8080-exec-1] o.s.b.c.l.support.SimpleJobLaunc
er      : Job: [SimpleJob: [name=job]] completed with the following parameters: [{date=15671
3242078}] and the following status: [COMPLETED]
```

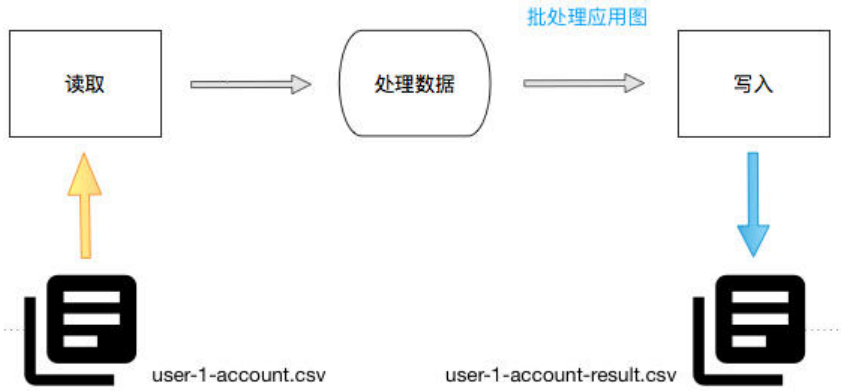
3.查看 项目>target/classes/user-1-account-result.csv

```
小张,2018,100
小张,2018,100
小李,2016,20
```

创建项目(数据库版本,开箱即用)

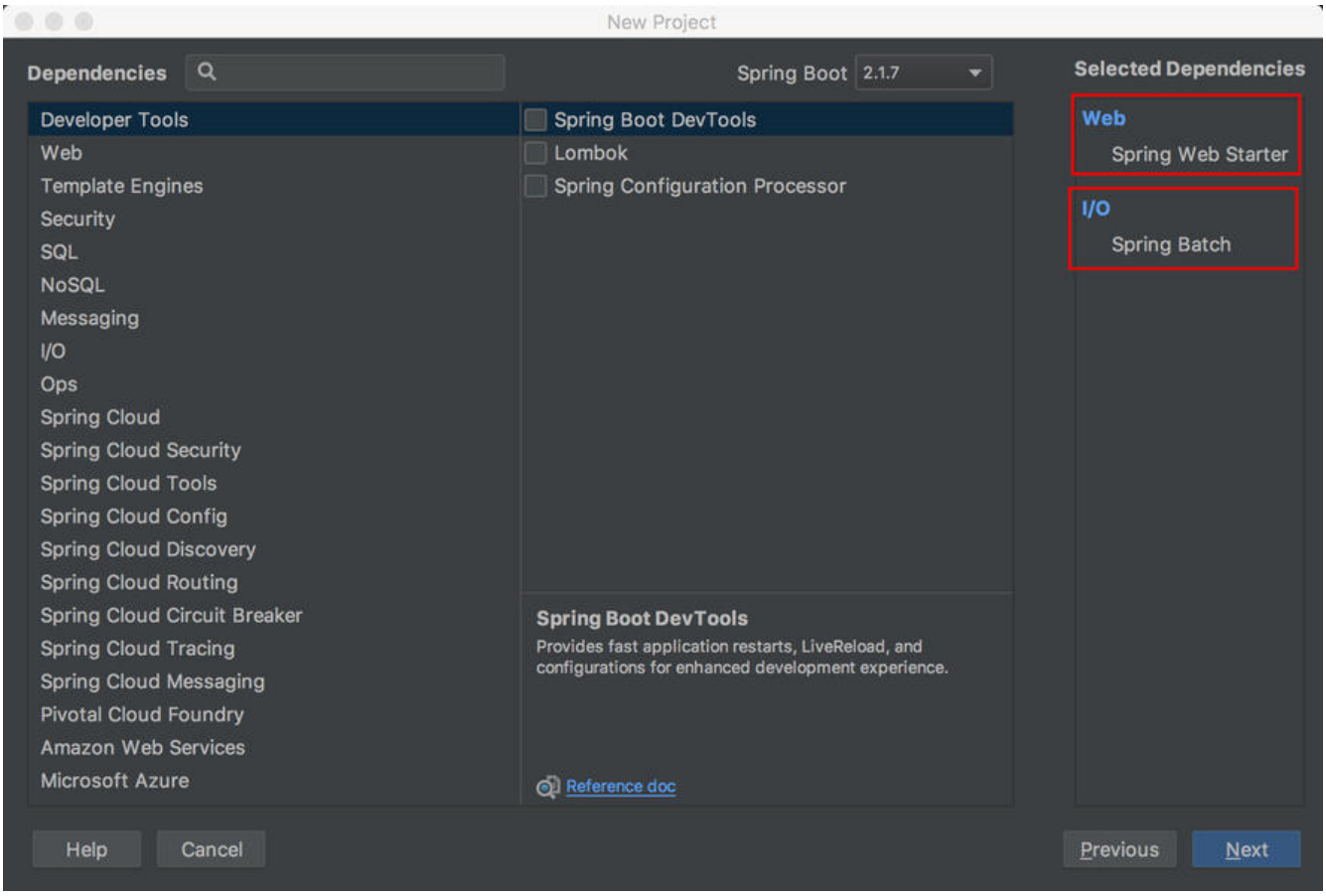
0.任务流程

使用spring bacth 读取user-1-account.csv文件的内容
处理完成写入到user-1-account-result.csv文件中



1.使用开发工具(这里笔者使用IDEA)创建项目

2.选择如下



3.修改pom文件如下

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
MLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd
maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.7.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.adongs</groupId>
<artifactId>spring-batch</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>spring-batch</name>
<description>Demo project for Spring Boot</description>

<properties>
    <java.version>1.8</java.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-batch</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.batch</groupId>
        <artifactId>spring-batch-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

```
</project>
```

4.修改配置文件application.yml

```
spring:
  batch:
    job:
      enabled: false
  datasource:
    url: jdbc:mysql://localhost:3306/test
    username: root
    password: root
    driver-class-name: com.mysql.cj.jdbc.Driver
```

5.创建账户实体

```
package com.adongs.springbatch;

import org.springframework.stereotype.Component;

import java.util.Date;

/**
 * 账户实体
 * @Author yudong
 * @Date 2019/8/15 下午4:57
 * @Version 1.0
 */
@Component
public class Account {

    /**
     * 姓名
     */
    private String name;

    /**
     * 时间
     */
    private String date;

    /**
     * 金额
     */
    private int money;

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

    public void setName(String name) {
        this.name = name;
```

```
    }

    public String getDate() {
        return this.date;
    }

    public void setDate(String date) {
        this.date = date;
    }

    public int getMoney() {
        return this.money;
    }

    public void setMoney(int money) {
        this.money = money;
    }

    @Override
    public String toString() {
        return "Account{" +
                "name='" + name + '\'' +
                ", date=" + date +
                ", money=" + money +
                '}';
    }
}
```

6.创建一个配置类JobConfig

```
package com.adongs.springbatch;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.FlatFileItemWriter;
import org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper;
import org.springframework.batch.item.file.mapping.DefaultLineMapper;
import org.springframework.batch.item.file.transform.BeanWrapperFieldExtractor;
import org.springframework.batch.item.file.transform.DelimitedLineAggregator;
import org.springframework.batch.item.file.transform.DelimitedLineTokenizer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
```

```
import org.springframework.core.io.ClassPathResource;

/**
 * @Author yudong
 * @Date 2019/8/16 上午11:11
 * @Version 1.0
 */
@Configuration
@EnableBatchProcessing
public class JobConfig {


    @Autowired
    private JobBuilderFactory jobBuilderFactory;

    @Autowired
    private StepBuilderFactory stepBuilderFactory;
    @Autowired
    private ApplicationContext applicationContext;

    /**
     * 配置Job
     * @return
     */
    @Bean
    public Job footballJob(Step step){
        return this.jobBuilderFactory.get("job")
                .incrementer(new RunIdIncrementer())
                .listener(sampleListener())
                .start(step).build();
    }

    /**
     * 构建step
     * @param reader
     * @param processor
     * @param flatFileItemWriter
     * @return
     */
    @Bean
    public Step reconciliation(FlatFileItemReader reader,
                               ItemProcessor processor,
                               FlatFileItemWriter flatFileItemWriter){

        return  stepBuilderFactory.get("reconciliation")
                .<Account,Account> chunk(2)
                .reader(reader)
                .processor(processor)
                .writer(flatFileItemWriter)
                .build();
    }

    /**
     * 读取文件
```

```
     * @return
     */
    @Bean
    public FlatFileItemReader flatFileItemReader(){
        FlatFileItemReader flatFileItemReader = new FlatFileItemReader();
        flatFileItemReader.setResource(new ClassPathResource("user-1-account.csv"));
        DelimitedLineTokenizer delimitedLineTokenizer = new DelimitedLineTokenizer();
        delimitedLineTokenizer.setDelimiter(",");
        delimitedLineTokenizer.setNames("name","date","money");
        BeanWrapperFieldSetMapper beanWrapperFieldSetMapper = new BeanWrapperFieldSe
Mapper();
        beanWrapperFieldSetMapper.setBeanFactory(applicationContext);
        beanWrapperFieldSetMapper.setPrototypeBeanName("account");
        DefaultLineMapper defaultLineMapper = new DefaultLineMapper();
        defaultLineMapper.setLineTokenizer(delimitedLineTokenizer);
        defaultLineMapper.setFieldSetMapper(beanWrapperFieldSetMapper);
        flatFileItemReader.setLineMapper(defaultLineMapper);
        return flatFileItemReader;
    }


    /**
     * 处理器
     * @return
     */
    @Bean
    public ItemProcessor creditBillProcessor(){
        return new ItemProcessor<Account,Account>(){
            @Override
            public Account process(Account item) throws Exception {
                System.out.println(item.toString());
                return item;
            }
        };
    }

    /**
     * 写入文件
     * @return
     */
    @Bean
    public FlatFileItemWriter flatFileItemWriter(){
        BeanWrapperFieldExtractor beanWrapperFieldExtractor = new BeanWrapperFieldExtract
r();
        beanWrapperFieldExtractor.setNames(new String[]{"name","date","money"});
        DelimitedLineAggregator delimitedLineAggregator = new DelimitedLineAggregator();
        delimitedLineAggregator.setDelimiter(",");
        delimitedLineAggregator.setFieldExtractor(beanWrapperFieldExtractor);
        FlatFileItemWriter flatFileItemWriter = new FlatFileItemWriter();
        flatFileItemWriter.setResource(new ClassPathResource("user-1-account-result.csv"));
        flatFileItemWriter.setLineAggregator(delimitedLineAggregator);
        return flatFileItemWriter;
    }
```

```
    /**
     * 监听
     * @return
     */
    @Bean
    public JobExecutionListener sampleListener(){
        return new JobExecutionListener(){
            @Override
            public void beforeJob(JobExecution jobExecution) {
                System.out.println("JobExecutionListener.beforeJob");
            }

            @Override
            public void afterJob(JobExecution jobExecution) {
                System.out.println("JobExecutionListener.afterJob");
            }
        };
    }
}
```

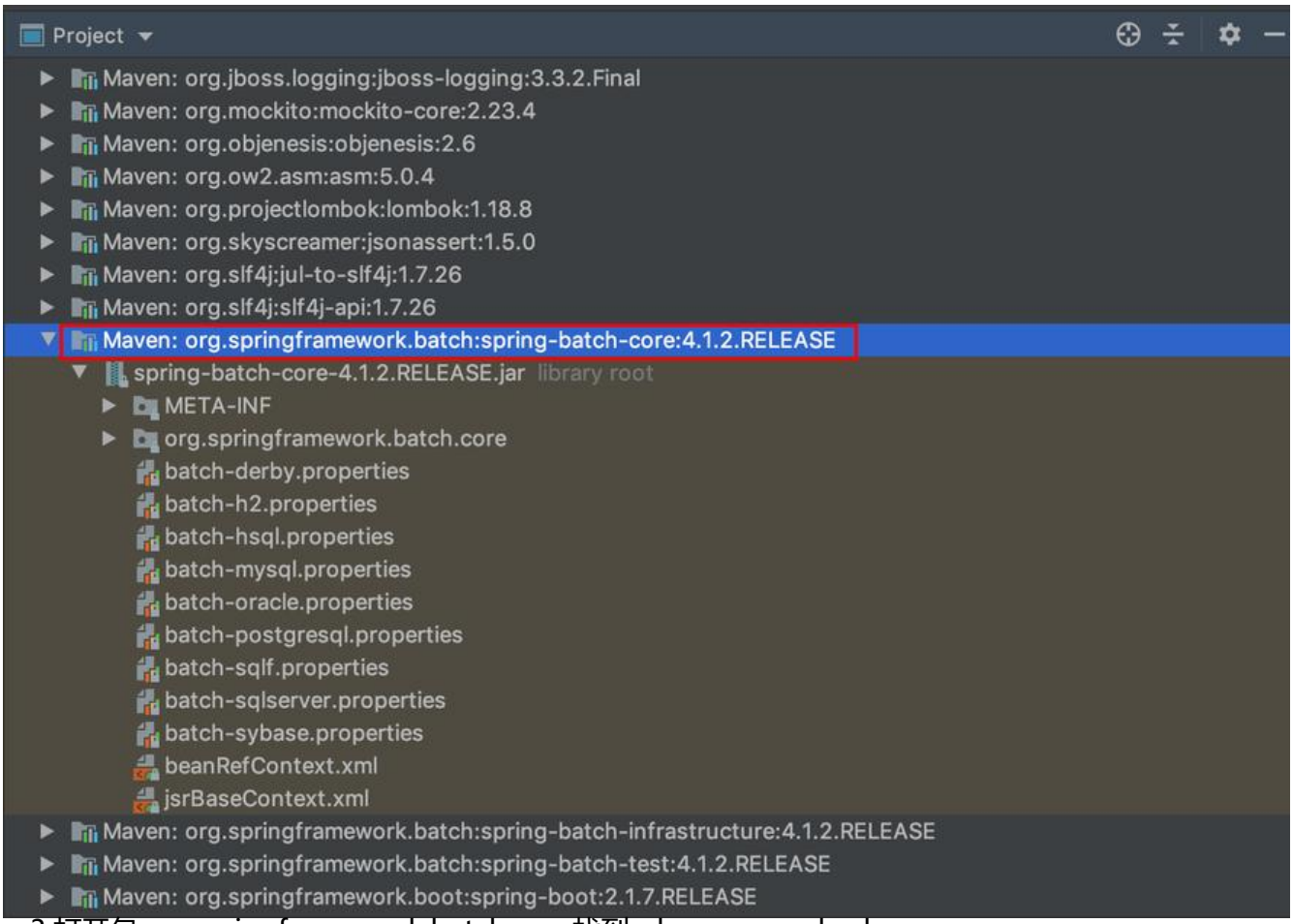
7.创建测试TestRunController

```
package com.adongs.springbatch;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.JobParametersBuilder;
import org.springframework.batch.core.JobParametersInvalidException;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.repository.JobExecutionAlreadyRunningException;
import org.springframework.batch.core.repository.JobInstanceAlreadyCompleteException;
import org.springframework.batch.core.repository.JobRestartException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.annotation.Resource;
import java.util.Date;

/**
 * @Author yudong
 * @Date 2019/8/15 下午7:34
 * @Version 1.0
 */
@RestController
public class TestRunController {


    @Autowired
    JobLauncher jobLauncher;
```

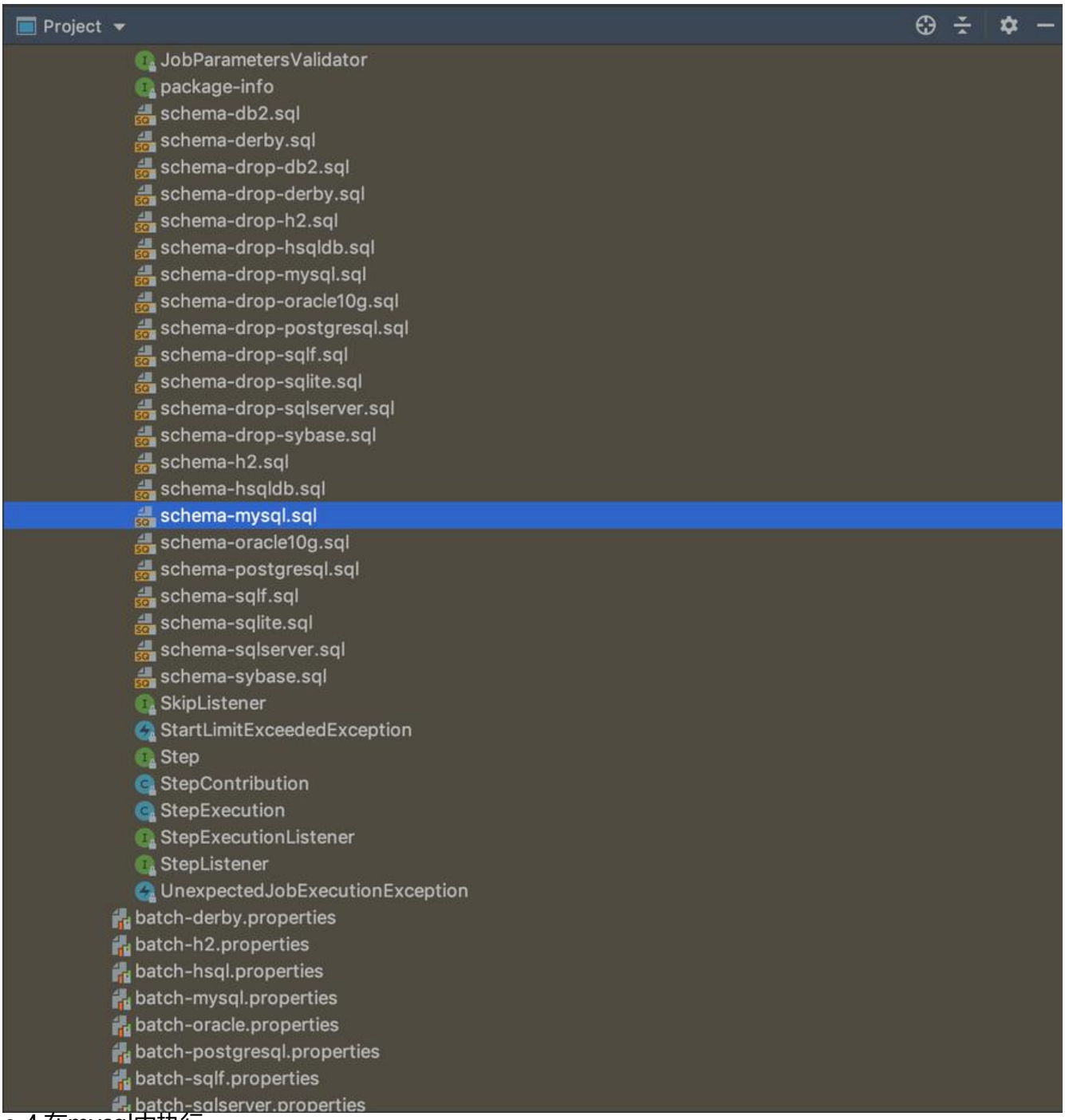
```
    @Autowired
    Job job;

    @GetMapping("test")
    public String test() throws JobParametersInvalidException, JobExecutionAlreadyRunningExc
ption, JobRestartException, JobInstanceAlreadyCompleteException {
        JobParameters jobParameters = new JobParametersBuilder().addDate("date",new Date()).
oJobParameters();
        jobLauncher.run(job,jobParameters);
        return "ok";
    }

}
```

8.创建数据库,并创建表

- 1.创建一个名称为test的数据库
- 2.找到batch核心包(org.springframework.batch:spring-batch-core:4.1.2.RELEASE)



- 3.打开包org.springframework.batch.core找到schema-mysql.sql



- 4.在mysql中执行

9.在resources文件夹下创建两个文件user-1-account.csv,user-1-account-result.csv,并在user-1-account.csv写入如下内容

小张,2019,50
小张,2018,100
小李,2016,20

测试

1.启动项目

2.请求连接:http://localhost:8080/test 输出如下

```
2019-08-30 16:20:42.139  INFO 57596 --- [nio-8080-exec-1] o.s.b.c.l.support.SimpleJobLaunc
er      : Job: [SimpleJob: [name=job]] launched with the following parameters: [{date=1567153
42078}]
JobExecutionListener.beforeJob
2019-08-30 16:20:42.153  INFO 57596 --- [nio-8080-exec-1] o.s.batch.core.job.SimpleStepHa
dler     : Executing step: [reconciliation]
Account{name='小张', date=2018, money=100}
Account{name='小张', date=2018, money=100}
Account{name='小李', date=2016, money=20}
JobExecutionListener.afterJob
2019-08-30 16:20:42.412  INFO 57596 --- [nio-8080-exec-1] o.s.b.c.l.support.SimpleJobLaunc
er      : Job: [SimpleJob: [name=job]] completed with the following parameters: [{date=15671
3242078}] and the following status: [COMPLETED]
```

3.查看 项目>target/classes/user-1-account-result.csv

```
小张,2018,100
小张,2018,100
小李,2016,20
```

4.查看数据库,发现有记录数据

Name	Rows	Data Length	Engine	Created Date	Modified Date	Collation	Comm...
BATCH_JOB_EXECUTION	2	16.00 KB	InnoDB	2019-09-04 06:35:10		utf8mb4_general_ci	
BATCH_JOB_EXECUTION_CONTEXT	2	16.00 KB	InnoDB	2019-09-04 06:35:10		utf8mb4_general_ci	
BATCH_JOB_EXECUTION_PARAMS	2	16.00 KB	InnoDB	2019-09-04 06:35:10		utf8mb4_general_ci	
BATCH_JOB_EXECUTION_SEQ	0	16.00 KB	InnoDB	2019-09-04 06:35:10		utf8mb4_general_ci	
BATCH_JOB_INSTANCE	2	16.00 KB	InnoDB	2019-09-04 06:35:10		utf8mb4_general_ci	
BATCH_JOB_SEQ	0	16.00 KB	InnoDB	2019-09-04 06:35:10		utf8mb4_general_ci	
BATCH_STEP_EXECUTION	2	16.00 KB	InnoDB	2019-09-04 06:35:10		utf8mb4_general_ci	
BATCH_STEP_EXECUTION_CONTEXT	2	16.00 KB	InnoDB	2019-09-04 06:35:10		utf8mb4_general_ci	
BATCH_STEP_EXECUTION_SEQ	0	16.00 KB	InnoDB	2019-09-04 06:35:10		utf8mb4_general_ci	

相关资源

1.demo地址(内存模式 master分支)

1.demo地址(数据库模式 db分支)