# SaaS-IHRM 项目-Activiti7 新特性

# 工作流引擎

传智播客-研究院

# 第1章 Activiti 与 Spring 整合开发

## 1.1Activiti 与 Spring 整合的配置

引入 pom.xml 坐标，如下

```
    <properties>
        <slf4j.version>1.6.6</slf4j.version>
        <log4j.version>1.2.12</log4j.version>
     </properties>
     <dependencies>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-engine</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>

        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-spring</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-bpmn-model</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-bpmn-converter</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-json-converter</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-bpmn-layout</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>org.activiti.cloud</groupId>
            <artifactId>activiti-cloud-services-api</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.5.4</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.40</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>5.0.7.RELEASE</version>
        </dependency>

        <!-- log start -->
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>${log4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>${slf4j.version}</version>
        </dependency>
```

```
            <dependency>
                <groupId>org.slf4j</groupId>
                <artifactId>slf4j-nop</artifactId>
                <version>${slf4j.version}</version>
            </dependency>
            <!-- log end -->

            <dependency>
                <groupId>org.mybatis</groupId>
                <artifactId>mybatis</artifactId>
                <version>3.4.5</version>
            </dependency>
            <dependency>
                <groupId>commons-dbcp</groupId>
                <artifactId>commons-dbcp</artifactId>
                <version>1.4</version>
            </dependency>
        </dependencies>
        <repositories>
            <repository>
                <id>alfresco</id>
                <name>Activiti Releases</name>

<url>https://artifacts.alfresco.com/nexus/content/repositories/activiti-releases/</
url>
                <releases>
                    <enabled>true</enabled>
                </releases>
            </repository>
        </repositories>
```

我们除了使用 StandaloneProcessEngineConfiguration 来创建 ProcessEngine 流程引擎对象。

还可以通过 **org.activiti.spring.SpringProcessEngineConfiguration 与 Spring** 整合方式来创建 ProcessEngine 对象。

创建 spring 与 activiti 的整合配置文件:
activiti-spring.xml（名称不固定）

```
    <beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd
    ">

    <!-- 数据源 -->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/activiti"/>
        <property name="username" value="root"/>
        <property name="password" value="root"/>
        <property name="maxActive" value="3"/>
        <property name="maxIdle" value="1"/>
    </bean>
    <!-- 工作流引擎配置 bean -->
    <bean                                                     id="processEngineConfiguration"
class="org.activiti.spring.SpringProcessEngineConfiguration">
        <!-- 数据源 -->
        <property name="dataSource" ref="dataSource"/>
        <!-- 使用 spring 事务管理器 -->
        <property name="transactionManager" ref="transactionManager"/>
        <!-- 数据库策略 -->
        <property name="databaseSchemaUpdate" value="drop-create"/>
    </bean>


    <!-- 流程引擎 -->
    <bean id="processEngine" class="org.activiti.spring.ProcessEngineFactoryBean">
        <property name="processEngineConfiguration" ref="processEngineConfiguration"/>
    </bean>
    <!-- 资源服务 service -->
    <bean id="repositoryService" factory-bean="processEngine"
          factory-method="getRepositoryService"/>
    <!-- 流程运行 service -->
    <bean id="runtimeService" factory-bean="processEngine"
          factory-method="getRuntimeService"/>
    <!-- 任务管理 service -->
    <bean id="taskService" factory-bean="processEngine"
          factory-method="getTaskService"/>
    <!-- 历史管理 service -->
    <bean id="historyService" factory-bean="processEngine"
          factory-method="getHistoryService"/>

    <!-- 事务管理器 -->
    <bean id="transactionManager"
          class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 通知 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <!-- 传播行为 -->
            <tx:method name="save*" propagation="REQUIRED"/>
            <tx:method name="insert*" propagation="REQUIRED"/>
            <tx:method name="delete*" propagation="REQUIRED"/>
            <tx:method name="update*" propagation="REQUIRED"/>
            <tx:method name="find*" propagation="SUPPORTS" read-only="true"/>
            <tx:method name="get*" propagation="SUPPORTS" read-only="true"/>
        </tx:attributes>
    </tx:advice>

    <!-- 切面，根据具体项目修改切点配置 -->
    <aop:config proxy-target-class="true">
        <aop:advisor advice-ref="txAdvice"
                     pointcut="execution(* com.itheima.ihrm.service.impl.*.*(..))"/>
    </aop:config>
</beans>
```

## 1.2 测试 Activiti 与 Spring 整合

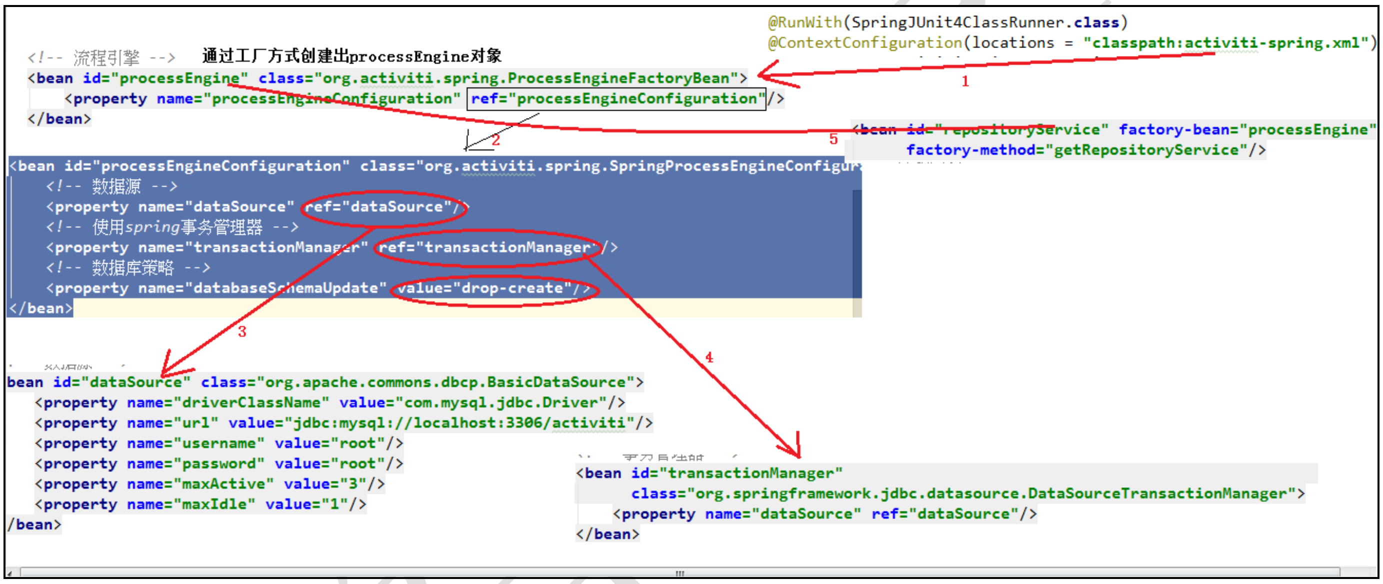
### 1.2.1 测试代码

```
/**
 *测试 activiti 与 spring 整合是否成功
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:activiti-spring.xml")
public class ActivitiTest {

    @Autowired
    private RepositoryService repositoryService;
    @Test
    public void test01(){
        System.out.println("部署对象:"+repositoryService);
    }
}
```

## 1.2.2　执行流程分析

下面我们一起来分析 Activiti 与 Spring 整合加载的过程。如下图所示：



```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:activiti-spring.xml")
```

```
<!-- 流程引擎 -->   通过工厂方式创建出processEngine对象
<bean id="processEngine" class="org.activiti.spring.ProcessEngineFactoryBean">
    <property name="processEngineConfiguration" ref="processEngineConfiguration"/>
</bean>

<bean id="repositoryService" factory-bean="processEngine"
      factory-method="getRepositoryService"/>

<bean id="processEngineConfiguration" class="org.activiti.spring.SpringProcessEngineConfigur
    <!-- 数据源 -->
    <property name="dataSource" ref="dataSource"/>
    <!-- 使用spring事务管理器 -->
    <property name="transactionManager" ref="transactionManager"/>
    <!-- 数据库策略 -->
    <property name="databaseSchemaUpdate" value="drop-create"/>
</bean>

bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
   <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
   <property name="url" value="jdbc:mysql://localhost:3306/activiti"/>
   <property name="username" value="root"/>
   <property name="password" value="root"/>
   <property name="maxActive" value="3"/>
   <property name="maxIdle" value="1"/>
/bean>

<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

从图中我们可以看出，首先我们加载 activiti-spring.xml 配置文件，而该配置文件中又会加载 SpringProcessEngineConfiguration 对象，这个对象它需要依赖注入 dataSource 对象和 transactionManager 对象。其次会加载 ProcessEngineFactoryBean 工厂来创建 ProcessEngine 对象，而 ProcessEngineFactoryBean 工厂又需要依赖注入 processEngineConfiguration 对象。最后由 processEngine 对象来负责创建我们的 Service 对象，从而简化 Activiti 的开发过程。在程序代码中我们可以根据实际需求来决定使用的 Service 对象。

# 第2章 Activiti7 现状分析

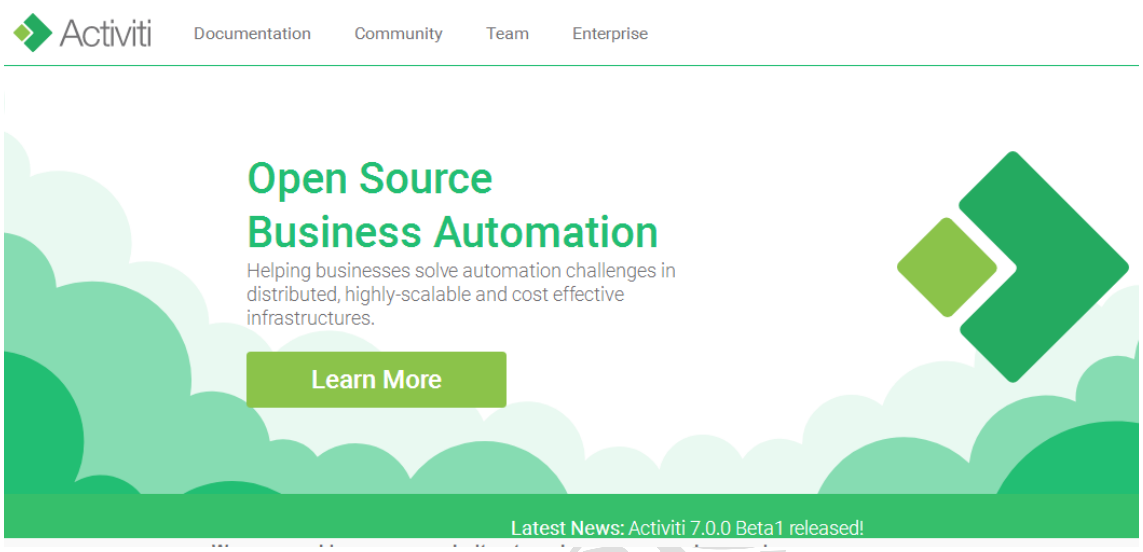
## 2.1Activiti7 基本介绍

Alfresco 软件在 2010 年 5 月 17 日宣布 Activiti 业务流程管理（BPM）开源项目的正式启动，其首席架构师由业务流程管理 BPM 的专家 Tom Baeyens 担任，Tom Baeyens 就是原来 jbpm 的架构师，而 jbpm 是一个非常有名的工作流引擎，当然 activiti 也是一个工作流引擎。

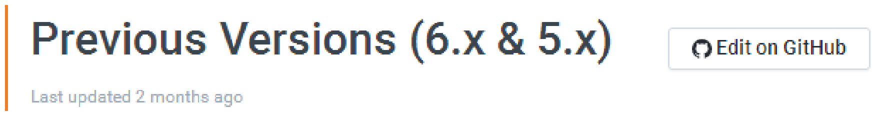
Activiti 是一个工作流引擎， activiti 可以将业务系统中复杂的业务流程抽取出来，使用专门的

建模语言（BPMN2.0）进行定义，业务系统按照预先定义的流程进行执行，实现了业务系统的业务流程由 activiti 进行管理，减少业务系统由于流程变更进行系统升级改造的工作量，从而提高系统的健壮性，同时也减少了系统开发维护成本。

官方网站：https://www.activiti.org/


Activiti
Documentation
Community
Team
Enterprise
Open Source
Business Automation
Helping businesses solve automation challenges in distributed, highly-scalable and cost effective infrastructures.
Learn More
Latest News: Activiti 7.0.0 Beta1 released!

经历的版本:



目前最新版本：Activiti7.0.0.Beta

## 2.2 下载官网资源

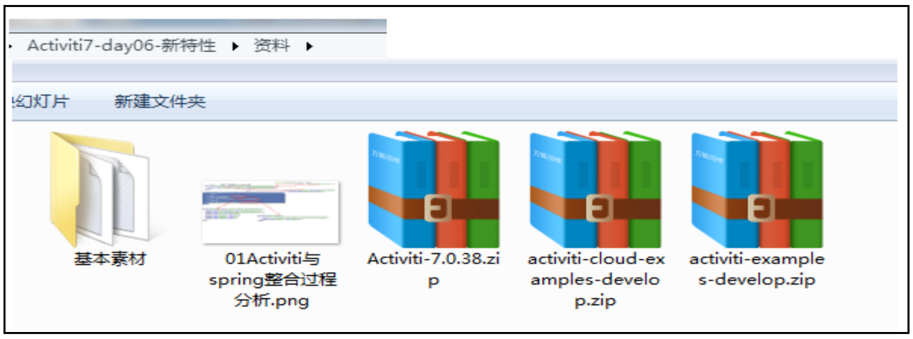
1.进入 Activiti-Core 学习向导页面

https://github.com/Activiti/activiti-7-developers-guide/blob/51a1681c0e4bb5e2f96a6dea73516c9fd53d8521/getting-started/getting-started-activiti-core.md

2.下载 Activiti-Core 的 Example 示例

https://github.com/Activiti/activiti-examples

3.已提供的资料所在位置

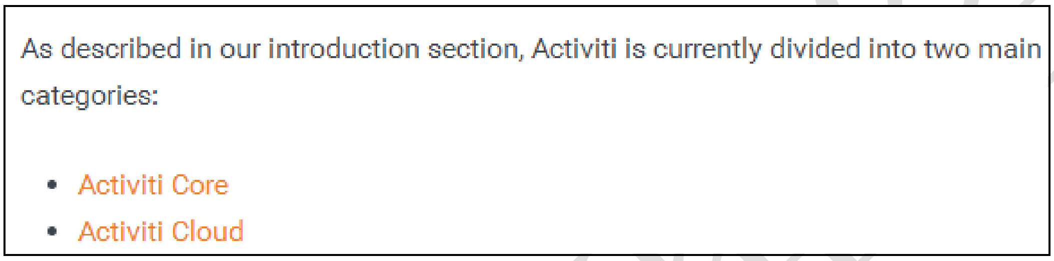

Activiti7-day06-新特性 ▸ 资料 ▸
幻灯片
新建文件夹
基本素材
01Activiti与spring整合过程分析.png
Activiti-7.0.38.zip
activiti-cloud-examples-develop.zip
activiti-examples-develop.zip

我们可以通过官方文档和官方示例来学习 Activiti7，目前国内的关于 Activiti7 的参考资料还很少，我们本次将结合这些资料来学习 Activiti7 课程。

### 2.3Activiti7 面临的挑战

Tom Baeyens 离职，并带着原班 Activiti 开发者又开发了一版新的工作流引擎，叫 flowable。这对于 Activiti 阵营而言是一个不少的挑战。我们现在所学的 Activiti7 是另一个架构师在原有的 Activiti6 基础上再进行一层封装，所以 API 有着很大的改变。

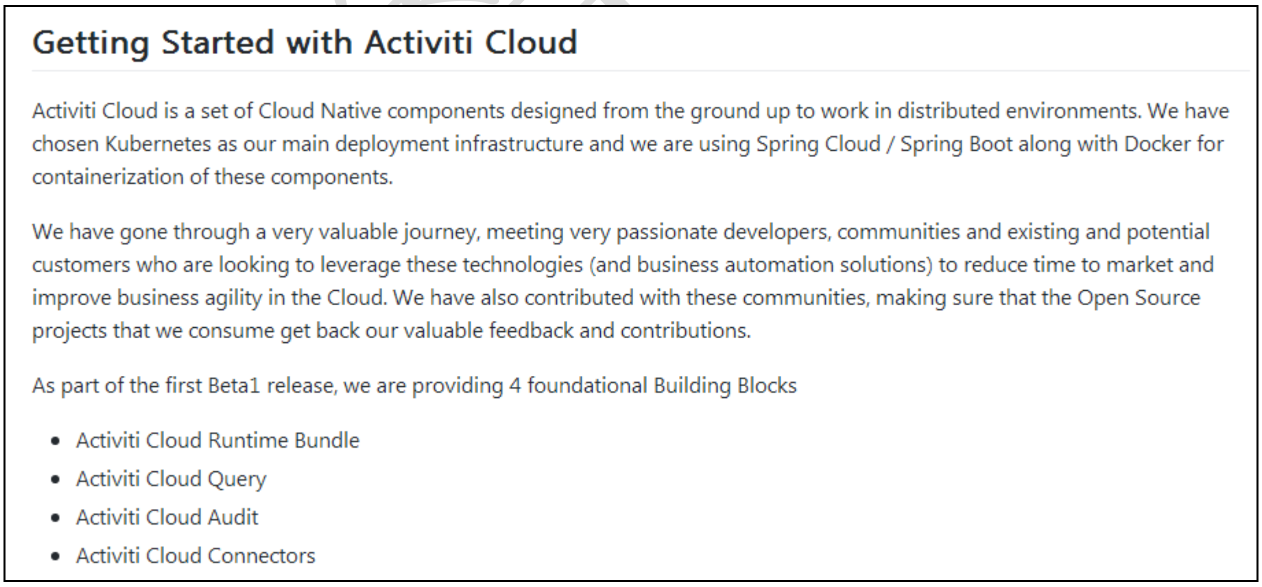
### 2.4Activiti7 的组成部分


As described in our introduction section, Activiti is currently divided into two main categories:

- Activiti Core
- Activiti Cloud

从这个部分我们可以看到 Activiti 现在已经分成两个不同的部分，Activiti Core 作为 Activiti 的核心部分，本次课程我们重点解决。另一部分 Activiti Cloud 主要是利用云服务来实现分布式业务流程开发。

Activiti Cloud 简单介绍


Getting Started with Activiti Cloud

Activiti Cloud is a set of Cloud Native components designed from the ground up to work in distributed environments. We have chosen Kubernetes as our main deployment infrastructure and we are using Spring Cloud / Spring Boot along with Docker for containerization of these components.

We have gone through a very valuable journey, meeting very passionate developers, communities and existing and potential customers who are looking to leverage these technologies (and business automation solutions) to reduce time to market and improve business agility in the Cloud. We have also contributed with these communities, making sure that the Open Source projects that we consume get back our valuable feedback and contributions.

As part of the first Beta1 release, we are providing 4 foundational Building Blocks

- Activiti Cloud Runtime Bundle
- Activiti Cloud Query
- Activiti Cloud Audit
- Activiti Cloud Connectors

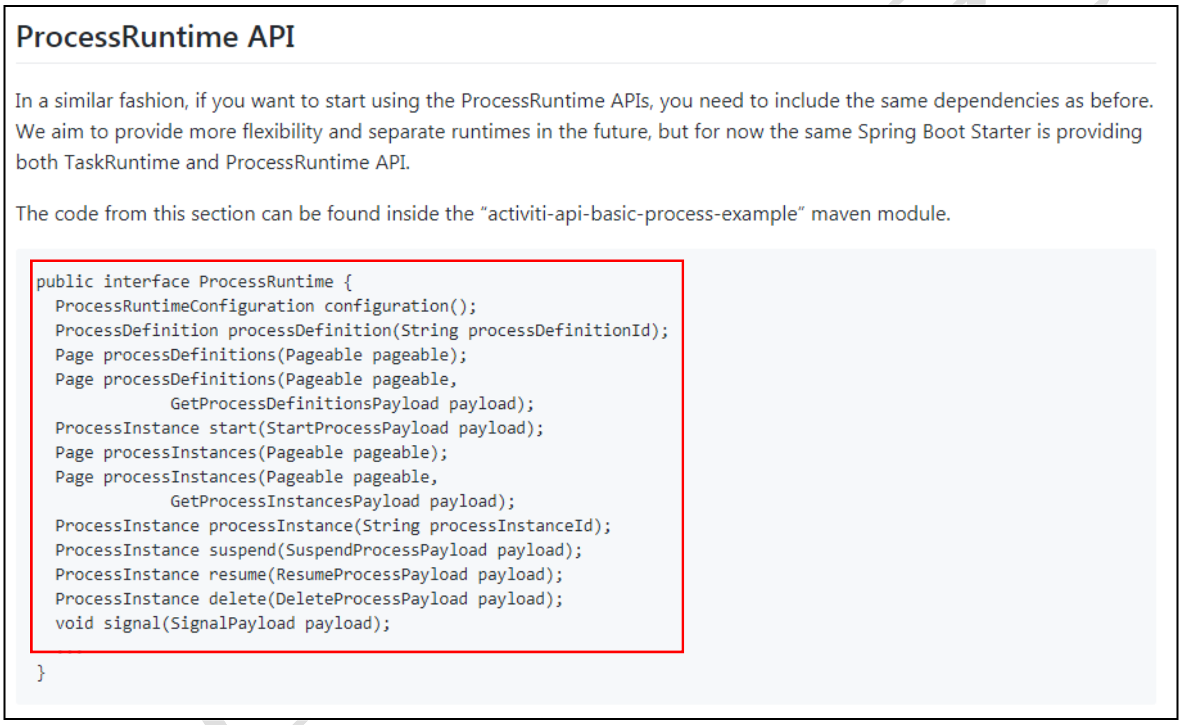
## 2.5 Activiti 新的 API 介绍

Activiti7 为了简化对工作流的操作，特别在原有 API 的基础上再次进行封闭，这样我们原来所学习的 Activiti 基本 API 就被封闭起来了。具体要学习的包括：

ProcessRuntime 接口

TaskRuntime 接口

### 2.5.1 ProcessRuntime 接口

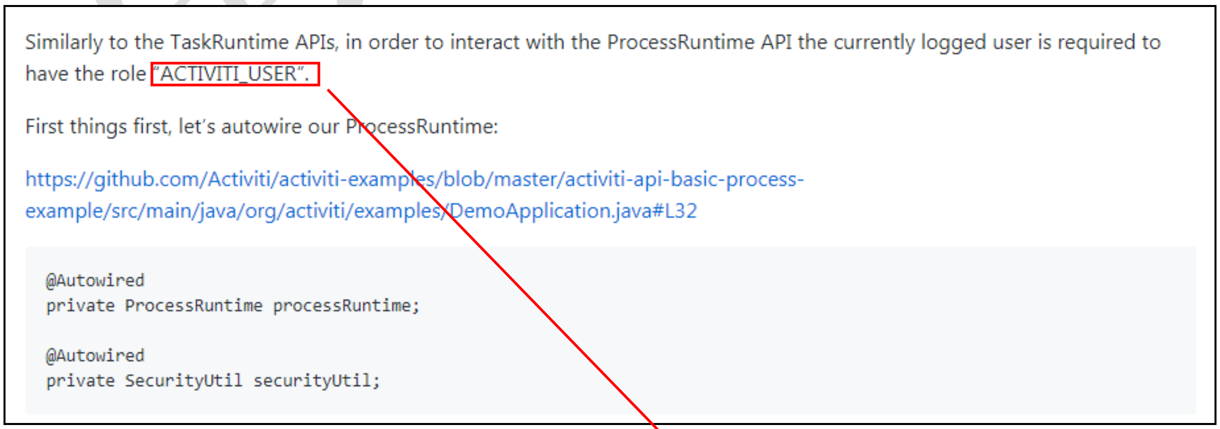


ProcessRuntime API

In a similar fashion, if you want to start using the ProcessRuntime APIs, you need to include the same dependencies as before. We aim to provide more flexibility and separate runtimes in the future, but for now the same Spring Boot Starter is providing both TaskRuntime and ProcessRuntime API.

The code from this section can be found inside the "activiti-api-basic-process-example" maven module.

```
public interface ProcessRuntime {
  ProcessRuntimeConfiguration configuration();
  ProcessDefinition processDefinition(String processDefinitionId);
  Page processDefinitions(Pageable pageable);
  Page processDefinitions(Pageable pageable,
            GetProcessDefinitionsPayload payload);
  ProcessInstance start(StartProcessPayload payload);
  Page processInstances(Pageable pageable);
  Page processInstances(Pageable pageable,
            GetProcessInstancesPayload payload);
  ProcessInstance processInstance(String processInstanceId);
  ProcessInstance suspend(SuspendProcessPayload payload);
  ProcessInstance resume(ResumeProcessPayload payload);
  ProcessInstance delete(DeleteProcessPayload payload);
  void signal(SignalPayload payload);
  ...
}
```



Similarly to the TaskRuntime APIs, in order to interact with the ProcessRuntime API the currently logged user is required to have the role "ACTIVITI_USER".

First things first, let's autowire our ProcessRuntime:

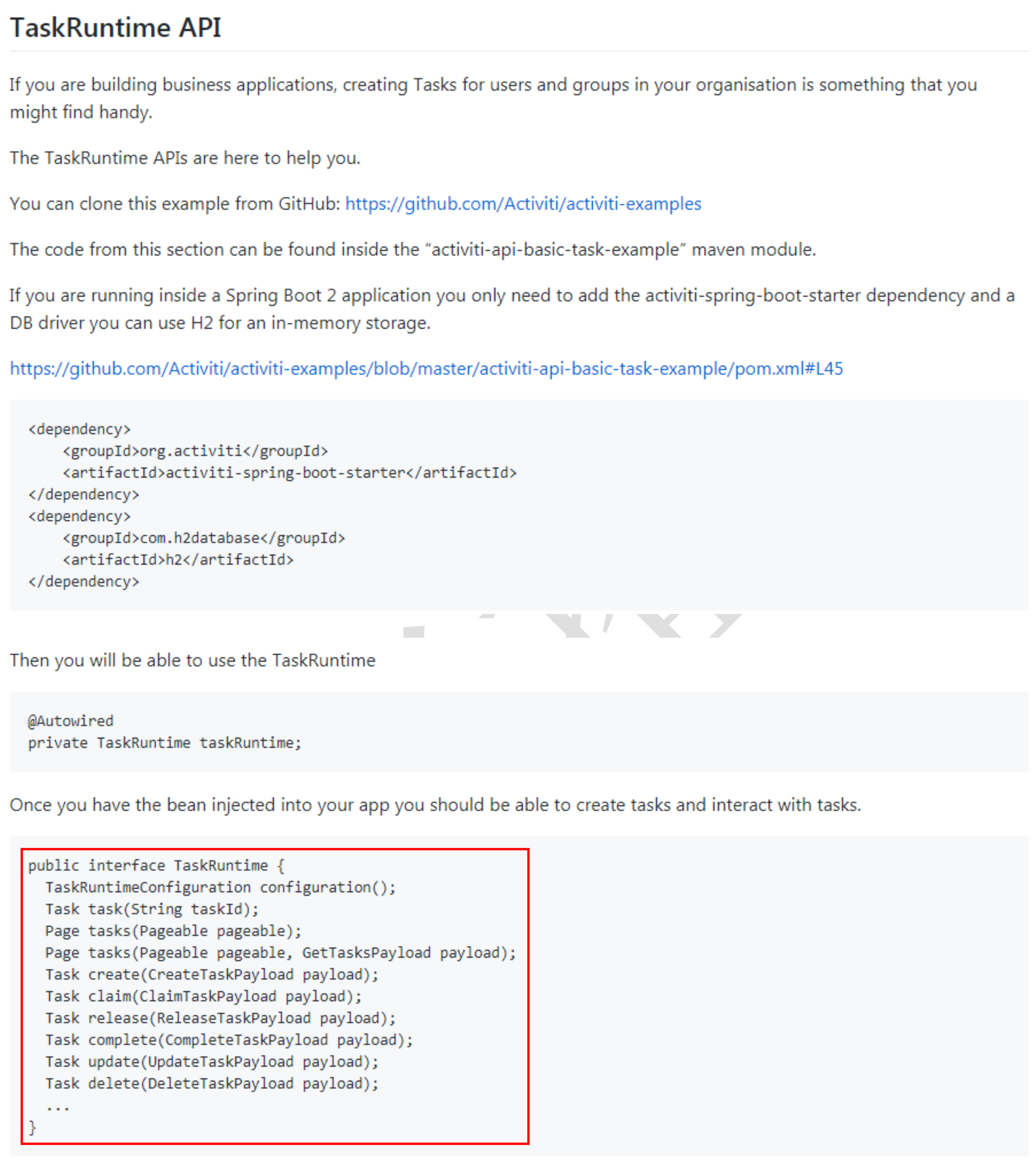
https://github.com/Activiti/activiti-examples/blob/master/activiti-api-basic-process-example/src/main/java/org/activiti/examples/DemoApplication.java#L32

```
@Autowired
private ProcessRuntime processRuntime;

@Autowired
private SecurityUtil securityUtil;
```

通过上面的分析，我们发现使用 Activiti7 开发时，只要注入 ProcessRuntime 的实现对象，就可以实现流程定义信息的操作。当然这个过程中因为 Activiti7 与 SpringSecurity 的强耦合，引导我们也必须将 SpringSecurity 加入进来。

### 2.5.2 TaskRuntime 接口



TaskRuntime API

If you are building business applications, creating Tasks for users and groups in your organisation is something that you might find handy.

The TaskRuntime APIs are here to help you.

You can clone this example from GitHub: https://github.com/Activiti/activiti-examples

The code from this section can be found inside the "activiti-api-basic-task-example" maven module.

If you are running inside a Spring Boot 2 application you only need to add the activiti-spring-boot-starter dependency and a DB driver you can use H2 for an in-memory storage.

https://github.com/Activiti/activiti-examples/blob/master/activiti-api-basic-task-example/pom.xml#L45

```
<dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-spring-boot-starter</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

Then you will be able to use the TaskRuntime

```
@Autowired
private TaskRuntime taskRuntime;
```

Once you have the bean injected into your app you should be able to create tasks and interact with tasks.

```
public interface TaskRuntime {
  TaskRuntimeConfiguration configuration();
  Task task(String taskId);
  Page tasks(Pageable pageable);
  Page tasks(Pageable pageable, GetTasksPayload payload);
  Task create(CreateTaskPayload payload);
  Task claim(ClaimTaskPayload payload);
  Task release(ReleaseTaskPayload payload);
  Task complete(CompleteTaskPayload payload);
  Task update(UpdateTaskPayload payload);
  Task delete(DeleteTaskPayload payload);
  ...
}
```

上面部分给我们介绍了如何引入 Activiti Core 所需的坐标，同时介绍了如何添加 TaskRuntime 实现对象，源码介绍等。我们会发现 TaskRuntime 本身就是对于 TaskService 的一个封装。

# 第3章 Activiti7 与 SpringBoot 整合开发

Activiti7 发布正式版之后，它与 SpringBoot2.x 已经完全支持整合开发。我们可以将 Activiti7 与 SpringBoot 整合开发的坐标引入到工程中，从而达到 SpringBoot 支持 Activti7 整合。

SpringBoot 整合 Activti7 的具体步骤如下：
1.添加 SpringBoot 整合 Activti7 的坐标

2.添加 SpringSecurity 安全框架的整合配置信息

3.使用 Activti7 新支持的类来实现工作流开发
  ProcessRuntime 接口
  TaskRuntime 接口

4.使用新的 API 实现工作流开发，主要包括
  流程定义查询
  启动流程实例
  任务的查询
  任务的完成

## 3.1 SpringBoot 整合 Activiti7 的配置

为了能够实现 SpringBoot 与 Activiti7 整合开发，首先我们要引入相关的依赖支持。
所以，我们在工程的 pom.xml 文件中引入相关的依赖，具体依赖如下所示：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.0.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
</properties>

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

```xml
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.activiti</groupId>
        <artifactId>activiti-spring-boot-starter</artifactId>
        <version>7.0.0.Beta2</version>
    </dependency>

    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
         <version>3.4.5</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.27</version>
    </dependency>
</dependencies>

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

通过该 pom.xml 文件所导入的坐标，我们就可以实现 activiti7 与 Springboot 整合。

## 3.2 SpringBoot 的 application.yml 文件配置

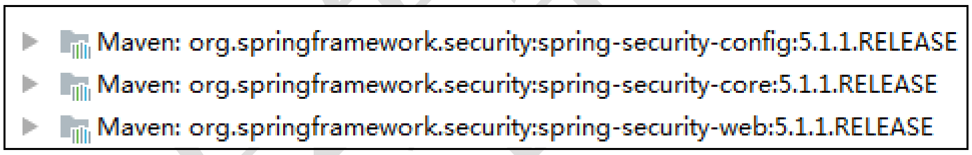
为了能够实现 Activiti7 生成的表放到 Mysql 数据库中，需要在 springboot 的配置文件 application.yml 中添加相关的配置

```
spring:
  datasource:
    url:
jdbc:mysql://localhost:3306/activiti?useUnicode=true&characterEncoding=utf8&serverT
imezone=GMT
    username : root
    password : root
    driver-class-name: com.mysql.jdbc.Driver
```

## 3.3 添加 SpringSecurity 安全框架整合配置

因为 Activiti7 与 SpringBoot 整合后，默认情况下，集成了 SpringSecurity 安全框架，这样我们就要去准备 SpringSecurity 整合进来的相关用户权限配置信息。

可以查看一下整合 SpringBoot 的依赖包，发现同时也将 SpringSecurity 的依赖包也添加进项目中了，如下：


```
Maven: org.springframework.security:spring-security-config:5.1.1.RELEASE
Maven: org.springframework.security:spring-security-core:5.1.1.RELEASE
Maven: org.springframework.security:spring-security-web:5.1.1.RELEASE
```

### 3.3.1 添加 SecurityUtil 类

**添加** SecurityUtil **类**。

为了能够快速实现 SpringSecurity 安全框架的配置，所添加的一个组件。

```
@Component
public class SecurityUtil {

    private Logger logger = LoggerFactory.getLogger(SecurityUtil.class);
    @Autowired
    @Qualifier("myUserDetailsService")
    private UserDetailsService userDetailsService;

    public void logInAs(String username) {

        UserDetails user = userDetailsService.loadUserByUsername(username);
        if (user == null) {
            throw new IllegalStateException("User " + username + " doesn't exist, please
provide a valid user");
        }
        logger.info("> Logged in as: " + username);
        SecurityContextHolder.setContext(new SecurityContextImpl(new Authentication() {
            @Override
            public Collection<? extends GrantedAuthority> getAuthorities() {
                return user.getAuthorities();
            }

            @Override
            public Object getCredentials() {
                return user.getPassword();
            }

            @Override
            public Object getDetails() {
                return user;
            }

            @Override
            public Object getPrincipal() {
                return user;
            }

            @Override
            public boolean isAuthenticated() {
                return true;
            }

            @Override
            public void setAuthenticated(boolean isAuthenticated) throws
IllegalArgumentException {

            }

            @Override
            public String getName() {
                return user.getUsername();
            }
        }));

org.activiti.engine.impl.identity.Authentication.setAuthenticatedUserId(username);
    }
}
```

这个类可以从我们下载的 Activiti7 官方提供的 Example 中找到。

## 3.3.2 添加 DemoApplicationConfig 类

在 Activiti7 官方下载的 Example 中找到 DemoApplicationConfig 类，它的作用是为了实现 SpringSecurity 框架的用户权限的配置，这样我们就可以在系统中使用用户权限信息。本次项目中基本是在文件中定义出来的用户信息，当然也可以是数据库中查询的用户权限信息。

```
@Configuration
public class DemoApplicationConfiguration {

    private Logger logger =
LoggerFactory.getLogger(DemoApplicationConfiguration.class);

    @Bean
    public UserDetailsService myUserDetailsService() {
        InMemoryUserDetailsManager inMemoryUserDetailsManager = new
InMemoryUserDetailsManager();

        String[][] usersGroupsAndRoles = {
                {"salaboy", "password", "ROLE_ACTIVITI_USER", "GROUP_activitiTeam"},
                {"ryandawsonuk", "password", "ROLE_ACTIVITI_USER", "GROUP_activitiTeam"},
                {"erdemedeiros", "password", "ROLE_ACTIVITI_USER", "GROUP_activitiTeam"},
                {"other", "password", "ROLE_ACTIVITI_USER", "GROUP_otherTeam"},
                {"system", "password", "ROLE_ACTIVITI_USER"},
                {"admin", "password", "ROLE_ACTIVITI_ADMIN"},
        };

        for (String[] user : usersGroupsAndRoles) {
            List<String> authoritiesStrings = Arrays.asList(Arrays.copyOfRange(user, 2,
user.length));
            logger.info("> Registering new user: " + user[0] + " with the following
Authorities[" + authoritiesStrings + "]");
            inMemoryUserDetailsManager.createUser(new User(user[0],
passwordEncoder().encode(user[1]),
                    authoritiesStrings.stream().map(s -> new
SimpleGrantedAuthority(s)).collect(Collectors.toList())));
        }

        return inMemoryUserDetailsManager;
    }


    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

}
```

## 3.4 使用 SpringBoot 整合 Junit 方式测试新特性

### 3.4.1 创建测试类

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class Actviti7DemoApplicationTests {
    @Autowired
    private ProcessRuntime processRuntime;
    @Autowired
    private TaskRuntime taskRuntime;
    @Autowired
    private SecurityUtil securityUtil;

}
```

通过上面的类，我们可以实现 SpringBoot 与 Junit 整合测试 Activiti7 所需的各种 API，之后我们就可以实现各种测试了。

### 3.4.2 查看流程定义信息

```
 /**
   * 查看流程定义
   */
   @Test
    public void contextLoads() {
        securityUtil.logInAs("system");
        Page<ProcessDefinition>                processDefinitionPage              =
processRuntime.processDefinitions(Pageable.of(0, 10));
        System.out.println("  可  用  的  流  程  定  义  数  量 ：  "  +
processDefinitionPage.getTotalItems());
        for         (org.activiti.api.process.model.ProcessDefinition        pd        :
processDefinitionPage.getContent()) {
            System.out.println("流程定义: " + pd);
        }
    }
```

通过 ProcessRuntime 的方法，我们可以查看已经部署的流程定义信息。通过加入 Pageable.of()方法可以实现分页查询效果。

## 3.4.3 启动流程实例

```
     /**
     * 启动流程实例
     */
    @Test
    public void testStartProcess() {
        securityUtil.logInAs("system");
        ProcessInstance                                     pi                                     =
processRuntime.start(ProcessPayloadBuilder.start().withProcessDefinitionKey("myProc
ess").build());
        System.out.println("流程实例 ID: " + pi.getId());
    }
```

现在要启动流程实例，我们可以使用 ProcessRuntime 的 start()方法就可以实现流程实例的启动

## 3.4.4 查询并完成任务

```
/**
 * 查询任务，并完成自己的任务
 */
@Test
public void testTask() {
    securityUtil.logInAs("ryandawsonuk");
    Page<Task> taskPage=taskRuntime.tasks(Pageable.of(0,10));
    if (taskPage.getTotalItems()>0){
        for (Task task:taskPage.getContent()){

taskRuntime.claim(TaskPayloadBuilder.claim().withTaskId(task.getId()).build());
            System.out.println("任务: "+task);

```

```java
taskRuntime.complete(TaskPayloadBuilder.complete().withTaskId(task.getId()).build()
);
        }
    }
    Page<Task> taskPage2=taskRuntime.tasks(Pageable.of(0,10));
    if (taskPage2.getTotalItems()>0){
        System.out.println("任务: "+taskPage2.getContent());
    }
}
```

使用 TaskRuntime 接口的 tasks()方法实现任务的查询。
使用 TaskRuntime 接口的 claim()方法实现任务拾取。
使用 TaskRuntime 接口的 complete()方法实现任务的完成

# 第4章 Activiti7+ SpringBoot+SpringMVC 开发

## 4.1 添加 Controller 类

```java
@RestController
public class TestController {
    private Logger logger = LoggerFactory.getLogger(TestController.class);
    @Autowired
    private ProcessRuntime processRuntime;
    @Autowired
    private TaskRuntime taskRuntime;
    @Autowired
    private SecurityUtil securityUtil;
}
```

## 4.2 实现任务完成

```java
@RequestMapping(value = "/hello")
public void hello() {
    //首先，取出项目中的最多 10 个流程定义
    Page<ProcessDefinition>                    processDefinitionPage                    =
processRuntime.processDefinitions(Pageable.of(0, 10));
```

```
    if (processDefinitionPage.getTotalItems() > 0) {
        //然后，对取出的流程进行启动
        for (ProcessDefinition definition : processDefinitionPage.getContent()) {
            logger.info("流程定义信息： " + definition);

processRuntime.start(ProcessPayloadBuilder.start().withProcessDefinitionId(definiti
on.getId()).build());
        }
    }
    //完成流程启动后，由于当前项目中只有 other.bpmn 一个流程，且该流程在设计时，已分配给
activitiTeam 组
    //因此我们登录一个 activitiTeam 组成员,该账号信息会被设置到 security 上下文中，activiti 会对其
信息进行读取
    //获取当前用户任务，最多 10 个
    Page<Task> taskPage = taskRuntime.tasks(Pageable.of(0, 10));
    //由于目前只有一个流程，两个任务，我们尝试一下完成一个，看看会发生什么变化
    if (taskPage.getTotalItems() > 0) {
        for (Task task : taskPage.getContent()) {
            logger.info("任务信息： " + task);
            //注意，完成任务前必须先拾取

taskRuntime.claim(TaskPayloadBuilder.claim().withTaskId(task.getId()).build());
            //完成任务

taskRuntime.complete(TaskPayloadBuilder.complete().withTaskId(task.getId()).build()
);
        }
    }
    //上一轮任务完成，再看一下，现在流程是否走到了 second?
    Page<Task> taskPage2 = taskRuntime.tasks(Pageable.of(0, 10));
    if (taskPage2.getTotalItems() > 0) {
        logger.info("任务信息： " + taskPage2.getContent());
    }
}
```

## 4.3 编写 SpringBoot 启动类

```
@SpringBootApplication(exclude = SecurityAutoConfiguration.class)
public class Actviti7DemoApplication {
  private Logger logger = LoggerFactory.getLogger(Actviti7DemoApplication.class);

  public static void main(String[] args) {
        SpringApplication.run(Actviti7DemoApplication.class, args);
    }
    @Bean
    public Connector testConnector() {
        return integrationContext -> {
            logger.info("以前叫代理，现在叫连接器被调用啦~~");
            return integrationContext;
        };
    }
}
```