ssm练习第三天

第一章：PageHelper的使用

第一节：分页的回顾

1. 分页步骤的回顾
2. 封装分页的JavaBean类

   资料中提供了PageBean的类，拷贝进来即可

3. 服务器端的分页代码的编写

```
    /**
     * 分页查询
     * @return
     */
    @RequestMapping("/findByPage")
    public ModelAndView findByPage(@RequestParam(required=true,defaultValue="1") int
pageCode,@RequestParam(required=true,defaultValue="5") int pageSize) {
        // 分页查询
        PageBean<Product> page = productService.findByPage(pageCode,pageSize);
        // 把plist存入到request域对象中
        ModelAndView mv = new ModelAndView();
        // 存入
        mv.addObject("page", page);
        // 设置跳转的JSP的页面
        mv.setViewName("product-list");
        return mv;
    }

    /**
     * 分页查询
     */
    public PageBean findByPage(int pageCode,int pageSize) {
        PageBean<Product> page = new PageBean<>();
        page.setPageCode(pageCode);
        page.setPageSize(pageSize);

        // 先查询数量
        int count = productDao.findCount();
        page.setTotalCount(count);

        // 再分页查询数据
        List<Product> list = productDao.findLimit((pageCode-1)*pageSize,pageSize);
        page.setBeanList(list);


        return page;
```

```
    }

    // 分页查询
    @Select("select * from product limit #{pageCode},#{pageSize}")
    List<Product> findLimit(@Param("pageCode") Integer pageCode, @Param("pageSize") Integer
pageSize);


    <li><a href="${ pageContext.request.contextPath }/product/findByPage?
pageCode=${page.pageCode-1}">上一页</a></li>

    <li><a href="${ pageContext.request.contextPath }/product/findByPage?
pageCode=${page.pageCode+1}">下一页</a></li>
```

第二节：PageHelper的介绍

PageHelper是国内非常优秀的一款开源的mybatis分页插件，它支持基本主流与常用的数据库，例如mysql、oracle、mariaDB、DB2、SQLite、Hsqldb等。

网址：https://pagehelper.github.io/

本项目在 github 的项目地址：https://github.com/pagehelper/Mybatis-PageHelper

本项目在 gitosc 的项目地址：http://git.oschina.net/free/Mybatis_PageHelper

第三节：PageHelper配置

1.集成

引入分页插件有下面2种方式，推荐使用 Maven 方式。

1.1. 引入 Jar 包

你可以从下面的地址中下载最新版本的 jar 包

- https://oss.sonatype.org/content/repositories/releases/com/github/pagehelper/pagehelper/
- http://repo1.maven.org/maven2/com/github/pagehelper/pagehelper/

由于使用了sql 解析工具，你还需要下载 jsqlparser.jar：

- http://repo1.maven.org/maven2/com/github/jsqlparser/jsqlparser/0.9.5/

1.2. 使用 Maven

在 pom.xml 中添加如下依赖：

```
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
    <version>最新版本</version>
</dependency>
```

2.配置

特别注意，新版拦截器是`com.github.pagehelper.PageInterceptor`。`com.github.pagehelper.PageHelper`现在是一个特殊的`dialect`实现类，是分页插件的默认实现类，提供了和以前相同的用法。

2.1. 在 MyBatis 配置 xml 中配置拦截器插件

```
<!--
    plugins在配置文件中的位置必须符合要求，否则会报错，顺序如下:
    properties?, settings?,
    typeAliases?, typeHandlers?,
    objectFactory?,objectWrapperFactory?,
    plugins?,
    environments?, databaseIdProvider?, mappers?
-->
<plugins>
    <!-- com.github.pagehelper为PageHelper类所在包名 -->
    <plugin interceptor="com.github.pagehelper.PageInterceptor">
        <!-- 使用下面的方式配置参数，后面会有所有的参数介绍 -->
        <property name="param1" value="value1"/>
    </plugin>
</plugins>
```

2.2. 在 Spring 配置文件中配置拦截器插件

使用 spring 的属性配置方式，可以使用`plugins`属性像下面这样配置:

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
  <!-- 注意其他配置 -->
  <property name="plugins">
    <array>
      <bean class="com.github.pagehelper.PageInterceptor">
        <property name="properties">
          <!--使用下面的方式配置参数，一行配置一个 -->
          <value>
            params=value1
          </value>
        </property>
      </bean>
    </array>
  </property>
</bean>
```

3 分页插件参数介绍

1. `helperDialect`：分页插件会自动检测当前的数据库链接，自动选择合适的分页方式。 你可以配置`helperDialect`属性来指定分页插件使用哪种方言。配置时，可以使用下面的缩写值：
   `oracle`,`mysql`,`mariadb`,`sqlite`,`hsqldb`,`postgresql`,`db2`,`sqlserver`,`informix`,`h2`,`sqlserver2012`,`derby`
   特别注意：使用 SqlServer2012 数据库时，需要手动指定为`sqlserver2012`，否则会使用 SqlServer2005 的方式进行分页。

你也可以实现 `AbstractHelperDialect`，然后配置该属性为实现类的全限定名称即可使用自定义的实现方法。

2. `offsetAsPageNum`：默认值为 `false`，该参数对使用 `RowBounds` 作为分页参数时有效。 当该参数设置为 `true` 时，会将 `RowBounds` 中的 `offset` 参数当成 `pageNum` 使用，可以用页码和页面大小两个参数进行分页。
3. `rowBoundsWithCount`：默认值为 `false`，该参数对使用 `RowBounds` 作为分页参数时有效。 当该参数设置为 `true` 时，使用 `RowBounds` 分页会进行 count 查询。
4. `pageSizeZero`：默认值为 `false`，当该参数设置为 `true` 时，如果 `pageSize=0` 或者 `RowBounds.limit = 0` 就会查询出全部的结果（相当于没有执行分页查询，但是返回结果仍然是 `Page` 类型）。
5. `reasonable`：分页合理化参数，默认值为 `false`。当该参数设置为 `true` 时，`pageNum<=0` 时会查询第一页，`pageNum>pages`（超过总数时），会查询最后一页。默认 `false` 时，直接根据参数进行查询。
6. `params`：为了支持 `startPage(Object params)` 方法，增加了该参数来配置参数映射，用于从对象中根据属性名取值， 可以配置 `pageNum,pageSize,count,pageSizeZero,reasonable`，不配置映射的用默认值， 默认值为 `pageNum=pageNum;pageSize=pageSize;count=countSql;reasonable=reasonable;pageSizeZero=pageSizeZero`。
7. `supportMethodsArguments`：支持通过 Mapper 接口参数来传递分页参数，默认值 `false`，分页插件会从查询方法的参数值中，自动根据上面 `params` 配置的字段中取值，查找到合适的值时就会自动分页。 使用方法可以参考测试代码中的 `com.github.pagehelper.test.basic` 包下的 `ArgumentsMapTest` 和 `ArgumentsObjTest`。
8. `autoRuntimeDialect`：默认值为 `false`。设置为 `true` 时，允许在运行时根据多数据源自动识别对应方言的分页 （不支持自动选择 `sqlserver2012`，只能使用 `sqlserver`），用法和注意事项参考下面的**场景五**。
9. `closeConn`：默认值为 `true`。当使用运行时动态数据源或没有设置 `helperDialect` 属性自动获取数据库类型时，会自动获取一个数据库连接， 通过该属性来设置是否关闭获取的这个连接，默认 `true` 关闭，设置为 `false` 后，不会关闭获取的连接，这个参数的设置要根据自己选择的数据源来决定。

4.基本使用

PageHelper的基本使用有6种，大家可以查看文档，最常用的有两种

4.1. RowBounds方式的调用（了解）

```
List<Country> list = sqlSession.selectList("x.y.selectIf", null, new RowBounds(1, 10));
```

使用这种调用方式时，你可以使用RowBounds参数进行分页，这种方式侵入性最小，我们可以看到，通过RowBounds方式调用只是使用了这个参数，并没有增加其他任何内容。

分页插件检测到使用了RowBounds参数时，就会对该查询进行**物理分页。**

关于这种方式的调用，有两个特殊的参数是针对 `RowBounds` 的，你可以参看上面的分页插件参数介绍

注：不只有命名空间方式可以用RowBounds，使用接口的时候也可以增加RowBounds参数，例如：

```
//这种情况下也会进行物理分页查询
List<Country> selectAll(RowBounds rowBounds);
```

注意： 由于默认情况下的 `RowBounds` 无法获取查询总数，分页插件提供了一个继承自 `RowBounds` 的 `PageRowBounds`，这个对象中增加了 `total` 属性，执行分页查询后，可以从该属性得到查询总数。

4.2. PageHelper.startPage 静态方法调用（重点）

这种方式是我们要掌握的 在你需要进行分页的 MyBatis 查询方法前调用PageHelper.startPage 静态方法即可，紧跟在这个方法后的第一个MyBatis 查询方法会被进行分页。

```
//获取第1页，10条内容，默认查询总数count
PageHelper.startPage(1, 10);
//紧跟着的第一个select方法会被分页
List<Country> list = countryMapper.selectIf(1);
```

第四节：分页代码编写

1. 服务器端代码的编写

```
    /**
     * 使用PageHelper分页的方式
     * @param pageNum
     * @param pageSize
     * @return
     */
    @RequestMapping("/findByPageHelper")
    public ModelAndView findByPageHelper(@RequestParam(required=true,defaultValue="1")int
pageNum,@RequestParam(required=true,defaultValue="5")int pageSize) {
        // 分页查询
        PageInfo page = productService.findByPageHelper(pageNum, pageSize);
        // 把plist存入到request域对象中
        ModelAndView mv = new ModelAndView();
        // 存入
        mv.addObject("page", page);
        // 设置跳转的JSP的页面
        mv.setViewName("product-list");
        return mv;
    }

    /**
     * 使用PageHelper的方式分页查询
     * 注意，不是分页的方法，是查询所有数据的方法
     */
    public PageInfo findByPageHelper(int pageNum,int pageSize) {
        // 设置第几页，每次查询的条数
        PageHelper.startPage(pageNum, pageSize);
        // 后面必须跟着查询所有数据的方法（注意，不是分页的方法，是查询所有数据的方法）
        List<Product> list = productDao.findAll();
        // 创建PageInfo类
        PageInfo<Product> page = new PageInfo<>(list);
        return page;
    }

    /**
     * 查询所有数据
     * @return

     */
```

```
    @Select("select * from product")
    List<Product> findAll();
```

2. JSP页面的代码编写

```
<div class="box-footer">
<div class="pull-left">
<div class="form-group form-inline">
总共${ page.pages } 页，共${ page.total }条数据。
每页 <select class="form-control" onchange="submitPageSize(this)" id="pageSelect">
<option value="5" <c:if test="${ page.pageSize == 5 }">selected</c:if>>5</option>
<option value="10" <c:if test="${ page.pageSize == 10 }">selected</c:if>>10</option>
</select> 条
</div>
<script type="text/javascript">
function submitPageSize(who){
// who指的是当前选中的options对象
// alert(who.value);
// 发送请求
location.href="${pageContext.request.contextPath}/product/findByPageHelper?
pageSize="+who.value;
}

</script>
</div>

<div class="box-tools pull-right">
<ul class="pagination">
<li><a href="javascript:submitPageNum(1)" aria-label="Previous">首页</a></li>
<li><a href="javascript:submitPageNum(${ page.prePage })">上一页</a></li>
<c:forEach begin="1" end="${ page.pages }" var="i">
<li><a href="javascript:submitPageNum(${ i })">${ i }</a></li>
</c:forEach>
<li><a href="javascript:submitPageNum(${ page.nextPage })">下一页</a></li>
<li><a href="javascript:submitPageNum(${ page.pages })" aria-label="Next">尾页</a></li>
</ul>
</div>
<script type="text/javascript">
// 提交当前页
function submitPageNum(pageNum){
location.href="${pageContext.request.contextPath}/product/findByPageHelper?
pageNum="+pageNum;
}
</script>
```

第二章：Spring Security安全框架

第一节：权限管理概述

1. BRAC的模型：基于角色的访问控制模型。

第二节：Security框架基本介绍

Spring Security 的前身是 Acegi Security ， 是 Spring 项目组中用来提供安全认证服务的框架。(https://projects.spring.io/spring-security/) Spring Security 为基于J2EE企业应用软件提供了全面安全服务。特别是使用领先的J2EE解决方案-Spring框架开发的企业软件项目。人们使用Spring Security有很多种原因，不过通常吸引他们的是在J2EE Servlet规范或EJB规范中找不到典型企业应用场景的解决方案。 特别要指出的是他们不能再 WAR 或 EAR 级别进行移植。这样，如果你更换服务器环境，就要，在新的目标环境进行大量的工作，对你的应用系统进行重新配 置安全。使用Spring Security 解决了这些问题，也为你提供很多有用的，完全可以指定的其他安全特性。 安全包括两个主要操作。

- “认证”，是为用户建立一个他所声明的主体。主题一般式指用户，设备或可以在你系 统中执行动作的其他系统。
- “授权”指的是一个用户能否在你的应用中执行某个操作，在到达授权判断之前，身份的主题已经由 身份验证过程建立了。

这些概念是通用的，不是Spring Security特有的。在身份验证层面，Spring Security广泛支持各种身份验证模式，这些验证模型绝大多数都由第三方提供，或则正在开发的有关标准机构提供的，例如 Internet Engineering Task Force.作为补充，Spring Security 也提供了自己的一套验证功能。

Spring Security 目前支持认证一体化如下认证技术： HTTP BASIC authentication headers (一个基于IEFT RFC 的标准) HTTP Digest authentication headers (一个基于IEFT RFC 的标准) HTTP X.509 client certificate exchange (一个基于IEFT RFC 的标准) LDAP (一个非常常见的跨平台认证需要做法，特别是在大环境) Form-based authentication (提供简单用户接口的需求) OpenID authentication Computer Associates Siteminder JA-SIG Central Authentication Service (CAS，这是一个流行的开源单点登录系统) Transparent authentication context propagation for Remote Method Invocation and HttpInvoker (一个Spring远程调用协议)

Maven依赖

```
<dependencies>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-web</artifactId>
        <version>5.0.1.RELEASE</version>
    </dependency>
</dependencies>
```

第三节：Security框架快速入门

1.maven依赖

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
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>
    <groupId>spring_security_demo</groupId>
    <artifactId>SpringSecurity_quickStart</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
    <properties>

        <spring.version>5.0.2.RELEASE</spring.version>
```

```
        <spring.security.version>5.0.1.RELEASE</spring.security.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-support</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>

            <!-- java编译插件 -->
```

```
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.2</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <configuration>
                    <!-- 指定端口 -->
                    <port>8080</port>
                    <!-- 请求路径 -->
                    <path>/</path>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

2.配置web.xml

```
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-security.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

3.Spring Security配置

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:security="http://www.springframework.org/schema/security"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd

    http://www.springframework.org/schema/security
```

```
    http://www.springframework.org/schema/security/spring-security.xsd">

    <!-- 入门代码 -->
    <!--
        配置拦截的规则
        auto-config="使用自带的页面"
        use-expressions="是否使用spel表达式"，如果使用表达式：hasRole('ROLE_USER')
    -->
    <security:http auto-config="true" use-expressions="false">
        <!-- 配置拦截的请求地址，任何请求地址都必须有ROLE_USER的权限 -->
        <security:intercept-url pattern="/**" access="ROLE_USER"/>
    </security:http>

    <!-- 在内存中临时提供用户名和密码的数据 -->
    <security:authentication-manager>
        <security:authentication-provider>
            <security:user-service>
                <security:user name="admin" password="{noop}admin" authorities="ROLE_USER"/>
            </security:user-service>
        </security:authentication-provider>
    </security:authentication-manager>

</beans>
```

4.测试

我们在webapp下创建一个index.html页面，在页面中任意写些内容。

当我们访问index.html页面时发现会弹出登录窗口，可能你会奇怪，我们没有建立下面的登录页面，为什么Spring Security会跳到上面的登录页面呢？这是我们设置http的auto-config=”true”时Spring Security自动为我们生成的。

5.使用自定义页面

5.1 spring-security.xml配置

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:security="http://www.springframework.org/schema/security"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security.xsd">

    <!-- 入门代码 -->

    <!-- 把登陆页面不拦截 -->
    <security:http pattern="/login.html" security="none"/>
    <security:http pattern="/error.html" security="none"/>

    <!--
```

```
        配置拦截的规则
        auto-config="使用自带的页面"
        use-expressions="是否使用spel表达式"，如果使用表达式：hasRole('ROLE_USER')
    -->
    <security:http auto-config="true" use-expressions="false">
        <!-- 配置拦截的请求地址，任何请求地址都必须有ROLE_USER的权限 -->
        <security:intercept-url pattern="/**" access="ROLE_USER"/>
        <!-- 配置具体的页面跳转 -->
        <security:form-login
            login-page="/login.html"
            login-processing-url="/login"
            default-target-url="/success.html"
            authentication-failure-url="/error.html"
        />

        <!-- 关闭跨越请求 -->
        <security:csrf disabled="true"/>

        <!-- 退出 -->
        <security:logout invalidate-session="true" logout-url="/logout" logout-success-
url="/login.html"/>

    </security:http>

    <!-- 在内存中临时提供用户名和密码的数据 -->
    <security:authentication-manager>
        <security:authentication-provider>
            <security:user-service>
                <security:user name="admin" password="{noop}admin" authorities="ROLE_USER"/>
                <security:user name="user" password="{noop}user" authorities="ROLE_ADMIN"/>
            </security:user-service>
        </security:authentication-provider>
    </security:authentication-manager>

</beans>
```

5.2 login.html

```
  <!DOCTYPE html>
  <html>
  <head>
  <meta charset="UTF-8">
  <title>Insert title here</title>
  </head>
  <body>
  <form action="login" method="post">
    <table>
        <tr>
            <td>用户名：</td>
            <td><input type="text" name="username" /></td>
        </tr>
        <tr>
```

```
            <td>密码: </td>
            <td><input type="password" name="password" /></td>
        </tr>
        <tr>
            <td colspan="2" align="center"><input type="submit" value="登录" />
                <input type="reset" value="重置" /></td>
        </tr>
    </table>
  </form>
  </body>
  </html>
```

5.3 success.html

```
  <!DOCTYPE html>
  <html>
  <head>
  <meta charset="UTF-8">
  <title>Insert title here</title>
  </head>
  <body>
    success html<br>
    <a href="logout">退出</a>
  </body>
  </html>
```

5.4 failer.html

```
  <!DOCTYPE html>
  <html>
  <head>
  <meta charset="UTF-8">
  <title>Insert title here</title>
  </head>
  <body>登录失败
  </body>
  </html>
```

第四节：在项目中引入Spring Security框架

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:security="http://www.springframework.org/schema/security"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security.xsd">

    <!-- 不拦截静态资源 -->
```

```
    <security:http pattern="/css/**" security="none"/>
    <security:http pattern="/img/**" security="none"/>
    <security:http pattern="/plugins/**" security="none"/>

    <!-- 把登陆页面不拦截 -->
    <security:http pattern="/login.jsp" security="none"/>
    <security:http pattern="/failer.jsp" security="none"/>

    <!--
        配置拦截的规则
        auto-config="使用自带的页面"
        use-expressions="是否使用spel表达式"，如果使用表达式：hasRole('ROLE_USER')
    -->
    <security:http auto-config="true" use-expressions="false">
        <!-- 配置拦截的请求地址，任何请求地址都必须有ROLE_USER的权限 -->
        <security:intercept-url pattern="/**" access="ROLE_USER"/>
        <!-- 配置具体的页面跳转 -->
        <security:form-login
            login-page="/login.jsp"
            login-processing-url="/login"
            default-target-url="/index.jsp"
            authentication-failure-url="/failer.jsp"
        />

        <!-- 关闭跨越请求 -->
        <security:csrf disabled="true"/>

        <!-- 退出 -->
        <security:logout invalidate-session="true" logout-url="/logout" logout-success-
url="/login.html"/>

    </security:http>

    <!-- 在内存中临时提供用户名和密码的数据 -->
    <security:authentication-manager>
        <security:authentication-provider>
            <security:user-service>
                <security:user name="admin" password="{noop}admin" authorities="ROLE_USER"/>
                <security:user name="user" password="{noop}user" authorities="ROLE_ADMIN"/>
            </security:user-service>
        </security:authentication-provider>
    </security:authentication-manager>

</beans>
```

第五节：搭建用户的开发环境

1. 创建表结构

```
CREATE TABLE sys_user(
  id NUMBER PRIMARY KEY ,
  username VARCHAR2(50),
  email VARCHAR2(50) ,
  PASSWORD VARCHAR2(80),
  phoneNum VARCHAR2(20),
  STATUS NUMBER(1)
);
```

2. 搭建用户模块的开发环境

```
public class SysUser implements Serializable{

    private static final long serialVersionUID = 6047270150238909794L;

    private Long id;
    private String username;
    private String email;
    private String password;
    private String phoneNum;
    private int status;
    // get/set
}
```

3. 省略步骤

第六节：自定义认证入门程序

1. 修改spring-security.xml配置文件

```
<!-- 在内存中临时提供用户名和密码的数据 -->
<security:authentication-manager>
    <!-- 提供服务类，去数据库查询用户名和密码 -->
    <security:authentication-provider user-service-ref="userService"/>
</security:authentication-manager>
```

2. service的代码如下

```
service的接口
public interface UserService extends UserDetailsService {

}

实现类代码
@Service("userService")
public class UserServiceImpl implements UserService {

    @Autowired
    private UserDao userDao;
```

```
    /**
     * 该方法是认证的方法
     * 先编写一个默认的认证代码
     * 参数就是表单提交的用户名
     */
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
{
        // 先设置假的权限
        List<GrantedAuthority> authorities = new ArrayList<>();
        // 传入角色
        authorities.add(new SimpleGrantedAuthority("ROLE_USER"));
        // 创建用户
        User user = new User(username, "{noop}123", authorities) ;
        return user;
    }

}
```

第七节：登录认证功能

1. 代码

```
@Service("userService")
public class UserServiceImpl implements UserService {

    @Autowired
    private UserDao userDao;

    /**
     * 该方法是认证的方法
     * 先编写一个默认的认证代码
     * 参数就是表单提交的用户名
     */
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
{
        // 先设置假的权限
        List<GrantedAuthority> authorities = new ArrayList<>();
        // 传入角色
        authorities.add(new SimpleGrantedAuthority("ROLE_USER"));

        // 通过用户名查询密码
        SysUser sysUser = userDao.findByUsername(username);

        // 创建用户
        User user = new User(username, "{noop}"+sysUser.getPassword(), authorities) ;
        return user;
    }

}
```

```
@Repository
public interface UserDao {

    @Select("select * from sys_user where username = #{username}")
    SysUser findByUsername(String username);

}
```