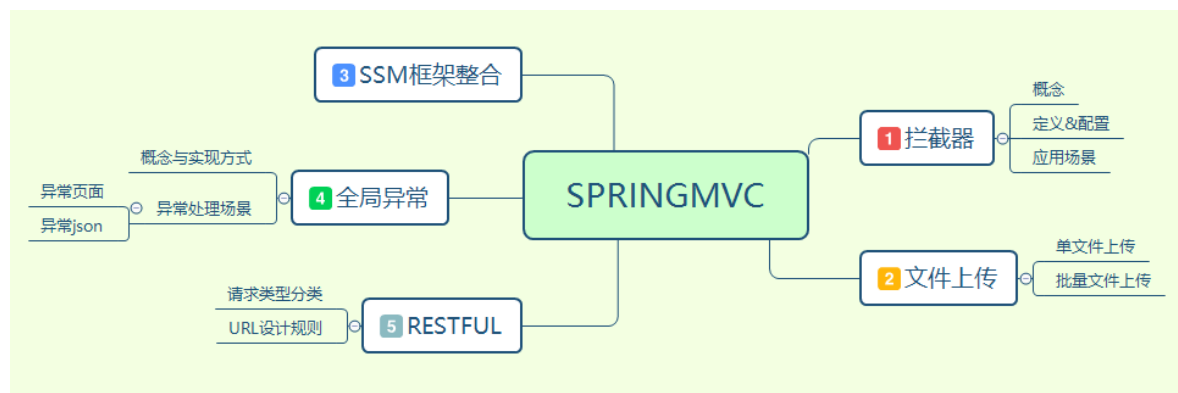


# SpringMvc-第二天

## 1. 学习目标



## 2. 拦截器

### 2.1. 基本概念

SpringMVC 中的Interceptor 拦截器也是相当重要和相当有用的，它的主要作用是拦截用户的请求并进行相应的处理。比如通过它来进行权限验证，或者是来判断用户是否登陆等操作。对于springmvc拦截器的定义方式有两种方式：

实现接口：org.springframework.web.servlet.HandlerInterceptor

继承适配器org.springframework.web.servlet.handler.HandlerInterceptorAdapter

### 2.2. 拦截器实现

#### 2.2.1. 实现HandlerInterceptor 接口

- 接口实现类

```
public class MyInterceptor1 implements HandlerInterceptor{
    /**
     * preHandle 在请求方法拦截前执行
     * 返回true 代表对当前请求进行放行处理
     */
    @Override
    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {
        System.out.println("handler方法之前执行MyInterceptor1-->preHandle方法...");
        return true; //继续执行action
    }

    /**
     * 请求执行后，生成视图前执行
     */
    @Override
    public void postHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler,
        ModelAndView modelAndView) throws Exception {
```

```

        System.out.println("handler方法之后，生成视图之前执行MyInterceptor1-->postHandle方法...");
    }

    /**
     * 在请求方法执行后进行拦截
     */
    @Override
    public void afterCompletion(HttpServletRequest request,
                               HttpServletResponse response, Object handler, Exception ex)
        throws Exception {

        System.out.println("handler方法执行完毕，生成视图后执行MyInterceptor1-->afterCompletion...");
    }
}

```

- 生效拦截器xml配置

```

<!--配置方式一-->
<mvc:interceptors>
    <!-- 使用bean定义一个Interceptor
    直接定义在mvc:interceptors根下面的Interceptor将拦截所有的请求 -->
    <bean class="com.xxxx.springmvc.interceptors.MyInterceptor1" />
</mvc:interceptors>

```

```

<!--配置方式二-->
<mvc:interceptors>
    <!-- 定义在 mvc:interceptor 下面 拦截所有test地址开头的请求-->
    <mvc:interceptor>
        <mvc:mapping path="/test/*.do" />
        <bean class="com.xxxx.springmvc.interceptors.MyInterceptor1" />
    </mvc:interceptor>
</mvc:interceptors>

```

## 2.2.2. 继承HandlerInterceptorAdapter

实际上最终还是HandlerInterceptor接口实现。

- 子类实现类

```

public class MyInterceptor2 extends HandlerInterceptorAdapter{
    /**
     * 重写preHandle 请求执行前执行
     */
    @Override
    public boolean preHandle(HttpServletRequest request,
                             HttpServletResponse response, Object handler) throws Exception {
        System.out.println("handler方法之前执行MyInterceptor2-->preHandle方法...");
        return true;
    }
}

```

- 生效拦截器xml配置

```

<!--配置方式二-->
<mvc:interceptors>
    <!-- 定义在 mvc:interceptor 下面 拦截所有test地址开头的请求-->
    <mvc:interceptor>
        <mvc:mapping path="/test/*.do" />
        <bean class="com.xxxx.springmvc.interceptors.MyInterceptor2" />
    </mvc:interceptor>
</mvc:interceptors>

```

### 2.2.3. 多个拦截器实现

SpringMvc 框架支持多个拦截器配置，从而构成拦截器链，对客户端请求进行多次拦截操作。

- 拦截器代码实现

这里参考MyInterceptor1、MyInterceptor2代码

- 生效拦截器xml配置

```

<mvc:interceptors>
    <mvc:interceptor>
        <!-- 拦截所有请求 -->
        <mvc:mapping path="/*" />
        <bean class="com.xxxx.springmvc.interceptors.MyInterceptor" />
    </mvc:interceptor>
    <mvc:interceptor>
        <!-- 拦截所有请求 -->
        <mvc:mapping path="/*" />
        <bean class="com.xxxx.springmvc.interceptors.MyInterceptor2" />
    </mvc:interceptor>
</mvc:interceptors>

```

## 2.3. 拦截器应用-非法请求拦截处理

使用拦截器完成用户是否登录请求验证功能

### 2.3.1. 用户控制器-UserController定义

```

/**
 *
 * @author Administrator
 * 模拟 用户操作
 */
@Controller
@RequestMapping("/user")
public class UserController {
    @RequestMapping("/userLogin")
    public ModelAndView userLogin(HttpServletRequest request){
        ModelAndView mv=new ModelAndView();
        User user=new User();
        user.setUserName("admin");
        user.setUserPwd("123456");
        request.getSession().setAttribute("user", user);
        mv.setViewName("success");
        return mv;
    }
    @RequestMapping("/addUser")

```

```

    public ModelAndView addUser(){
        System.out.println("添加用户记录。。。");
        ModelAndView mv=new ModelAndView();
        mv.setViewName("success");
        return mv;
    }
    @RequestMapping("/delUser")
    public ModelAndView delUser(){
        ModelAndView mv=new ModelAndView();
        mv.setViewName("success");
        return mv;
    }
    @RequestMapping("/updateUser")
    public ModelAndView updateUser(){
        ModelAndView mv=new ModelAndView();
        mv.setViewName("success");
        return mv;
    }
}

```

### 2.3.2. 非法请求拦截器定义

```

public class LoginInterceptor extends HandlerInterceptorAdapter{

    /**
     * 方法拦截前执行
     */
    @Override
    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {
        User user= (User) request.getSession().getAttribute("user");
        /**
         * 判断session user 是否为空
         */
        if(null==user){
            response.sendRedirect(request.getContextPath()+"/login.jsp");
            return false;
        }
        return true;
    }
}

```

### 2.3.3. 拦截器生效配置

```

<!-- 拦截所有请求 -->
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/*" />
        <!--放行用户登录请求-->
        <mvc:exclude-mapping path="/user/userLogin"/>
        <bean class="com.xxxx.springmvc.interceptors.LoginInterceptor" />
    </mvc:interceptor>
</mvc:interceptors>

```

## 3. SpringMvc文件上传

## 3.1. 环境配置

### 3.1.1. pom.xml文件修改

```
<!-- 添加commons-fileupload 依赖 -->
<dependency>
<groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.3.2</version>
</dependency>
```

### 3.1.2. servlet-context.xml修改

```
<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
  <property name="maxUploadSize">
    <value>104857600</value>
  </property>
  <property name="maxInMemorySize">
    <value>4096</value>
  </property>
</bean>
```

## 3.2. 代码实现

```
@Controller
public class FileController {
    @RequestMapping("/uploadFile")
    public ModelAndView uploadFile(HttpServletRequest request){
        ModelAndView mv=new ModelAndView();
        mv.setViewName("result");
        MultipartHttpServletRequest mr=(MultipartHttpServletRequest) request;
        MultipartFile multipartFile= mr.getFile("file");
        String
path=request.getSession().getServletContext().getRealPath("upload");
        System.out.println(path);
        if(null!=multipartFile&&!multipartFile.isEmpty()){
            String fileName=multipartFile.getOriginalFilename();
            try {
                multipartFile.transferTo(new File(path,fileName));
                mv.addObject("msg", "文件上传成功!");
            } catch (Exception e) {
                mv.addObject("msg", "上传失败!");
                e.printStackTrace();
            }
        }
        return mv;
    }
}
```

## 3.3. 准备表单

```
<form action="uploadFile.do" method="post" enctype="multipart/form-data">
    <input type="file" name="file" />
    <button type="submit"> 提交</button>
</form>
```

## 4. SSM框架集成与测试

### 4.1. 环境配置

#### 4.1.1. idea 下创建maven web 工程ssm

#### 4.1.2. pom.xml 坐标添加

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<dependencies>
    <!-- junit 测试 -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>

    <!-- spring 核心jar -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.2.RELEASE</version>
    </dependency>

    <!-- spring 测试jar -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>4.3.2.RELEASE</version>
    </dependency>

    <!-- spring jdbc -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>4.3.2.RELEASE</version>
    </dependency>

    <!-- spring事物 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>4.3.2.RELEASE</version>
    </dependency>
```

```
<!-- aspectj切面编程的jar -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.9</version>
</dependency>

<!-- c3p0 连接池 -->
<dependency>
    <groupId>c3p0</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.1.2</version>
</dependency>

<!-- mybatis -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.4.1</version>
</dependency>

<!-- 添加mybatis与Spring整合的核心包 -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.0</version>
</dependency>

<!-- mysql 驱动包 -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.39</version>
</dependency>

<!-- 日志打印相关的jar -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.2</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper -->
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
    <version>5.1.10</version>
</dependency>

<!-- spring web -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
```

```

        <version>4.3.2.RELEASE</version>
    </dependency>

    <!-- spring mvc -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>4.3.2.RELEASE</version>
    </dependency>

    <!-- web servlet -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.0.1</version>
    </dependency>

    <!-- 添加json 依赖jar包 -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.7.0</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.7.0</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>2.7.0</version>
    </dependency>

    <dependency>
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
        <version>1.3.2</version>
    </dependency>

</dependencies>

<build>
    <finalName>ssm</finalName>
    <!--
        Maven 项目:如果源代码(src/main/java)存在xml、properties、tld 等文件
        Maven 默认不会自动编译该文件到输出目录,如果要编译源代码中xml properties tld
        等文件
        需要显式配置resources 标签
    -->
    <resources>
        <resource>
            <directory>src/main/resources</directory>
        </resource>
        <resource>
            <directory>src/main/java</directory>

```



```

        <includes>
            <include>**/*.xml</include>
            <include>**/*.properties</include>
            <include>**/*.tld</include>
        </includes>
        <filtering>false</filtering>
    </resource>
</resources>

<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3.2</version>
        <configuration>
            <source>1.8</source>
            <target>1.8</target>
            <encoding>UTF-8</encoding>
        </configuration>
    </plugin>

    <plugin>
        <groupId>org.mortbay.jetty</groupId>
        <artifactId>maven-jetty-plugin</artifactId>
        <version>6.1.25</version>
        <configuration>
            <scanIntervalSeconds>10</scanIntervalSeconds>
            <contextPath>/ssm</contextPath>
        </configuration>
    </plugin>
</plugins>

</build>

```

#### 4.1.3. web.xml 文件修改

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="webApp_ID" version="3.0">
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring.xml</param-value>
    </context-param>
    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
    <filter>
        <description>char encoding filter</description>
        <filter-name>encodingFilter</filter-name>
        <filter-class>
            org.springframework.web.filter.CharacterEncodingFilter</filter-
class>

```

```

        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <servlet>
        <servlet-name>springMvc</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:servlet-context.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springMvc</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>

```

#### 4.1.4. servlet-context.xml添加

src/main/resources 下创建servlet-context.xml 内容如下:

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
    <!-- 扫描com.springmvc.crm 下包 -->
    <context:component-scan base-package="com.xxxx.ssm.controller" />

    <!-- mvc 注解驱动 并添加json 支持 -->
    <mvc:annotation-driven>
        <mvc:message-converters>
            <!-- 返回信息为字符串时 处理 -->
            <bean
class="org.springframework.http.converter.StringHttpMessageConverter"/>
            <!-- 将对象转换为json 对象 -->
            <bean
class="org.springframework.http.converter.json.MappingJackson2HttpMessageConvert
er"/>
        </mvc:message-converters>
    </mvc:annotation-driven>

```

```

<!-- 静态资源文件的处理放行 -->
<mvc:default-servlet-handler />

<!--配置视图解析器 默认的视图解析器-->
<bean id="defaultViewResolver"

class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView" />
    <property name="contentType" value="text/html" />
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>

<!-- 文件上传配置 -->
<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <property name="maxUploadSize">
        <value>104857600</value>
    </property>
    <property name="maxInMemorySize">
        <value>4096</value>
    </property>
</bean>
</beans>

```

#### 4.1.5. spring.xml配置

src/main/resources 下创建spring.xml 内容如下:

```

<?xml version="1.0" encoding="UTF-8"?>
<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:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.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
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- 扫描基本包 过滤controller层 -->
    <context:component-scan base-package="com.xxxx.ssm" >
    <context:exclude-filter type="annotation"
        expression="org.springframework.stereotype.Controller" />
    </context:component-scan>

    <!-- 加载properties 配置文件 -->
    <context:property-placeholder location="classpath:db.properties" />

    <aop:aspectj-autoproxy /><!-- aop -->

```

```

<!-- 配置c3p0 数据源 -->
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="${driver}"></property>
    <property name="jdbcUrl" value="${url}"></property>
    <property name="user" value="${user}"></property>
    <property name="password" value="${password}"></property>
</bean>

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

<!-- 设置事物增强 -->
<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="add*" propagation="REQUIRED" />
        <tx:method name="insert*" propagation="REQUIRED" />
        <tx:method name="update*" propagation="REQUIRED" />
        <tx:method name="delete*" propagation="REQUIRED" />
    </tx:attributes>
</tx:advice>

<!-- aop 切面配置 -->
<aop:config>
    <aop:pointcut id="servicePointcut"
        expression="execution(* com.xxx.ssm.service..*.*(..))" />
    <aop:advisor advice-ref="txAdvice" pointcut-ref="servicePointcut" />
</aop:config>

<!-- 配置 sqlSessionFactory -->
<bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation" value="classpath:mybatis.xml" />
    <property name="mapperLocations"
value="classpath:com/xxx/ssm/mapper/*.xml" />
</bean>

<!-- 配置扫描器 -->
<bean id="mapperScanner"
class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <!-- 扫描com.springmvc.dao这个包以及它的子包下的所有映射接口类 -->
    <property name="basePackage" value="com.xxx.ssm.dao" />
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
</bean>
</beans>

```

#### 4.1.6. mybatis.xml全局文件配置

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <package name="com.xxxx.ssm.vo"/>
    </typeAliases>
    <plugins>
        <plugin interceptor="com.github.pagehelper.PageInterceptor"></plugin>
    </plugins>
</configuration>
```

#### 4.1.7. 准备db.properties 文件

src/main/resources 下创建db.properties 内容如下(mysql 创建数据库ssm):

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/ssm?useUnicode=true&characterEncoding=utf8
jdbc.username=root
jdbc.password=root
```

#### 4.1.8. 添加log4j.properties 日志打印文件

src/main/resources 下创建log4j.properties 内容如下

```
log4j.rootLogger=DEBUG, Console
#Console
log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.Console.layout.ConversionPattern=%d [%t] %-5p [%c] - %m%n
log4j.logger.java.sql.ResultSet=INFO
log4j.logger.org.apache=INFO
log4j.logger.java.sql.Connection=DEBUG
log4j.logger.java.sql.Statement=DEBUG
log4j.logger.java.sql.PreparedStatement=DEBUG
```

### 4.2. 添加源代码

src/main/java 下创建

com.xxxx.ssm.controller、com.xxxx.ssm.service、com.xxxx.ssm.mapper、com.xxxx.ssm.dao、com.xxxx.ssm.vo等包结构。

#### 4.2.1. 添加HelloController.java

com.xxxx.ssm.controller包下创建HelloController.java

```
@Controller
public class HelloController {
    /**
     * 注入service 层userService 接口
     */
    @Autowired
    private UserService userService;
```

```

@RequestMapping("/hello")
public ModelAndView hello(){
    ModelAndView mv=new ModelAndView();
    /**
     * 调用service 层查询方法
     */
    User user=userService.queryUserById(7);
    System.out.println(user);
    mv.addObject("user", user);
    mv.setViewName("hello");
    return mv;
}
}

```

#### 4.2.2. 添加UserService.java,提供用户详情查询方法

com.xxxx.ssm.service 包下创建UserService.java 文件

```

@Service
public class UserService{











    @Autowired
    private UserMapper userMapper;

    public User queryUserByUserId(Integer userId){
        return userMapper.queryUserByUserId(userId);
    }
}

```

#### 4.2.3. 添加User.java 文件

com.xxxx.ssm.vo 包下创建JavaBean文件 User.java(数据库字段对应如下)

t_user	
	id int(11) (auto increment)
	user_name varchar(20)
	user_pwd varchar(100)
	true_name varchar(20)
	email varchar(30)
	phone varchar(20)
	is_valid int(4)
	create_date datetime
	update_date datetime
	PRIMARY (id)

```

public class User {
    private Integer id;

    private String userName;

    private String userPwd;

    private String trueName;
}

```

```

private String email;

private String phone;

private Integer isValid;

private Date createDate;

private Date updateDate;

/**
 * set get 方法省略
 */
}

```

#### 4.2.4. 添加UserMapper.java 接口文件 ,提供用户详情查询方法

com.xxxx.ssm.dao 包下创建UserMapper.java 文件

```

public interface UserMapper {

    User queryUserByUserId(Integer userId);

}

```

#### 4.2.5. 添加UserMapper.xml 映射文件， 提供select 查询标签配置

com.xxxx.ssm.mapper 包下创建UserMapper.xml 文件

```

<select id="queryUserByUserId" parameterType="integer"
resultType="com.xxxx.ssm.vo.User">
    select id, user_name
    from t_user where id=#{userId}
</select>

```

#### 4.2.6. 添加视图文件hello.jsp,展示查询的用户信息

在src/main/webapp/WEB-INF 创建jsp 目录，并在该目下创建hello.jsp

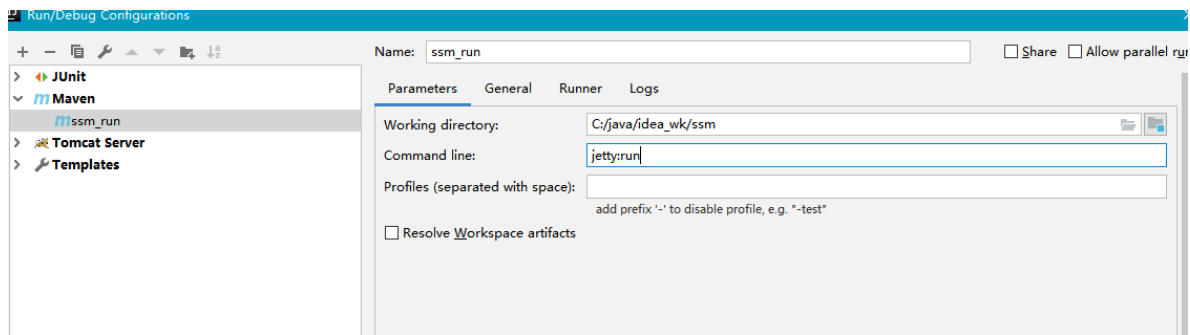
```

<body>
    欢迎你, ${user.userName}
</body>

```

### 4.3. 执行测试

#### 4.3.1. Idea下配置jetty启动命令



4.3.2. 启动jetty 浏览器访问<http://localhost:8080/ssm/hello.do> 查看结果

## 5. RestFul URL

### 5.1. 基本概念

模型-视图-控制器（MVC）是一个众所周知的以设计界面应用程序为基础的设计思想。

Restful风格的API是一种软件架构风格，设计风格而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务端交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。

在Restful风格中，用户请求的url使用同一个url而用请求方式：get, post, delete, put...等方式对请求的处理方法进行区分，这样可以在前后台分离式的开发中使得前端开发人员不会对请求的资源地址产生混淆和大量的检查方法名的麻烦，形成一个统一的接口。

在Restful风格中，现有规定如下：

- GET (SELECT)：从服务器查询，可以在服务器通过请求的参数区分查询的方式。
- POST (CREATE)：在服务器端新建一个资源，调用insert操作。
- PUT (UPDATE)：在服务器端更新资源，调用update操作。
- PATCH (UPDATE)：在服务器端更新资源（客户端提供改变的属性）。(目前jdk7未实现，tomcat7不支持)。
- DELETE (DELETE)：从服务器端删除资源，调用delete语句。

### 5.2. Spring Mvc 支持RestFul URL 风格设计

案例：如何在java构造没有扩展名的RESTful url,如 /forms/1?

SpringMvc是通过@RequestMapping 及@PathVariable annotation提供的,通过如  
@RequestMapping(value="/blog/{id}",method=RequestMethod.DELETE)即可处理/blog/1 的delete  
请求。

### 5.3. RestFul Url 映射地址配置实现

#### 5.3.1. Get 请求配置



```

/**
 *restful-->get 请求 执行查询操作
 * @param id
 * @return
 */
@GetMapping("account/{id}")
@ResponseBody
public Account queryAccountById(@PathVariable Integer id){
    return accountService.selectById(id);
}

```

### 5.3.2. Post请求配置

```

/* restful-->post请求执行添加操作
 * @return
 */
@PostMapping("account")
@ResponseBody
public Map<String,Object> saveAccount(@RequestBody Account account){
    int result = accountService.saveAccount(account);
    Map<String,Object> map=new HashMap<String,Object>();
    map.put("msg","success");
    map.put("code",200);
    if(result==0){
        map.put("msg","error");
        map.put("code",500);
    }
    return map;
}

```

### 5.3.3. Put请求配置

```

/* restful-->put 请求执行更新操作
 * @param id
 * @param account
 * @return
 */
@PutMapping("account")
@ResponseBody
public Map<String,Object> updateAccount(@RequestBody Account account){
    int result = accountService.updateAccount(account);
    Map<String,Object> map=new HashMap<String,Object>();
    map.put("msg","success");
    map.put("code",200);
    if(result==0){
        map.put("msg","error");
        map.put("code",500);
    }
    return map;
}

```

### 5.3.4. Delete请求配置

```

/* restful-->delete 请求 执行删除操作

```

```

    * @param id
    * @return
    */
    @DeleteMapping("account/{id}")
    @ResponseBody
    public Map<String, Object> deleteAccount(@PathVariable Integer id){
        int result = accountService.delAccount(id);
        Map<String, Object> map=new HashMap<String, Object>();
        map.put("msg", "success");
        map.put("code", 200);
        if(result==0){
            map.put("msg", "error");
            map.put("code", 500);
        }
        return map;
    }
}

```

## 6. SpringMVC 全局异常统一处理

### 6.1. 全局异常概念

在JavaEE 项目的开发中，不管是对底层的数据库操作过程，还是业务层的处理过程，还是控制层的处理过程，都不可避免会遇到各种可预知的、不可预知的异常需要处理。每个过程都单独处理异常，系统的代码耦合度高，工作量大且不好统一，维护的工作量也很大。

SpringMvc 对于异常处理这块提供了支持，通过 SpringMvc 提供的全局异常处理机制，能够将所有类型的异常处理从各处理过程解耦出来，这样既保证了相关处理过程的功能较单一，也实现了异常信息的统一处理和维。

全局异常实现方式Spring MVC 处理异常有 3 种方式

1. 使用 Spring MVC 提供的简单异常处理器 SimpleMappingExceptionHandler;
2. 实现 Spring 的异常处理接口 HandlerExceptionHandler 自定义自己的异常处理器;
3. 使用@ExceptionHandler 注解实现异常处理;

### 6.2. 异常处理实现

#### 6.2.1. 全局异常处理方式一

配置 SimpleMappingExceptionHandler 对象

```

<bean
    class="org.springframework.web.servlet.handler.SimpleMappingExceptionHandler">
        <property name="defaultErrorView" value="error"></property>
        <property name="exceptionAttribute" value="ex"></property>
        <property name="exceptionMappings">
            <props>
                <prop key="com.xxxx.ssm.exception.BusinessException">error</prop>
                <prop key="com.xxxx.ssm.exception.ParamsException">error</prop>
            </props>
        </property>
    </bean>

```

使用 SimpleMappingExceptionHandler 进行异常处理，具有集成简单、有良好的扩展性、对已有代码没有入侵性等优点，但该方法仅能获取到异常信息，若在出现异常时，对需要获取除异常以外的数据的情况不适用。

### 6.2.2. 全局异常处理方式二(推荐)

实现 HandlerExceptionResolver 接口

```
public class MyExceptionHandler implements HandlerExceptionResolver {
    @Override
    public ModelAndView resolveException(HttpServletRequest request,
        HttpServletResponse response, Object handler, Exception ex) {
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("ex", ex);
        ModelAndView mv = null;
        if (ex instanceof ParamsException) {
            return new ModelAndView("error_param", map);
        }
        if (ex instanceof BusinessException) {
            return new ModelAndView("error_business", map);
        }
        return new ModelAndView("error", map);
    }
}
```

使用实现 HandlerExceptionResolver 接口的异常处理器进行异常处理，具有集成简单、有良好的扩展性、对已有代码没有入侵性等优点，同时，在异常处理时能获取导致出现异常的对象，有利于提供更详细的异常处理信息。

### 6.2.3. 全局异常处理方式三

页面处理器继承 BaseController

```
public class BaseController {
    @ExceptionHandler
    public String exc(HttpServletRequest request, HttpServletResponse
        response, Exception ex) {
        request.setAttribute("ex", ex);
        if (ex instanceof ParamsException) {
            return "error_param";
        }
        if (ex instanceof BusinessException) {
            return "error_business";
        }
        return "error";
    }
}
```

使用 @ExceptionHandler 注解实现异常处理，具有集成简单、有扩展性好（只需要将要异常处理的 Controller 类继承于 BaseController 即可）、不需要附加 Spring 配置等优点，但该方法对已有代码存在入侵性(需要修改已有代码，使相关类继承于 BaseController)，在异常处理时不能获取除异常以外的数据。

## 6.3. 未捕获异常的处理

对于 Unchecked Exception 而言，由于代码不强制捕获，往往被忽略，如果运行期产生了 Unchecked Exception，而代码中又没有进行相应的捕获和处理，则我们可能不得不面对尴尬的 404、500.....等服务器内部错误提示页面。

此时需要一个全面而有效的异常处理机制。目前大多数服务器也都支持在 Web.xml 中通过 (Websphere/Weblogic)或者(Tomcat)节点配置特定异常情况的显示页面。修改 web.xml 文件，增加以下内容：

```
<!-- 出错页面定义 -->
<error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/500.jsp</location>
</error-page>
<error-page>
    <error-code>500</error-code>
    <location>/500.jsp</location>
</error-page>
<error-page>
    <error-code>404</error-code>
    <location>/404.jsp</location>
</error-page>
```

## 7. 课程总结

本章内容主要讲解了SpringMvc拦截器概念、实现配置与应用场景,借助SpringMvc来实现应用资源的拦截操作,借助SpringMvc框架文件上传环境实现了单文件与多文件上传功能,同时为了以后大家使用ssm框架开发企业应用,本章节完成了ssm框架环境整合操作，以后对与SpringMvc应用的开发基本都是在整合环境下进行,这里要求大家能够在整合环境下实现功能代码开发操作。

最后两块主要介绍Restful 概念,在SpringMvc环境下Restful URL 地址设计规范与账户模块代码实现，同时对于应用程序的异常通过使用SpringMvc 全局异常来进行统一处理，从而降低异常处理代码的耦合度。

通过两天内容的学习，这里基本上将SpringMvc 的核心知识点讲解完毕，并完成了SSM三大框架的整合操作，后续课程会在该环境下加入具体业务功能来帮助大家熟悉框架的使用，希望大家再接再厉。