# Ssm总结：都是用过的只说大概需要同学们自己敲，这上面的完全掌握，能驾驭一般企业级项目（一定动手实现）

1. ssm基本环境搭建:最后一次复习不能复制，手动敲一遍每个配置文件不需要完全记住但是每个里面配置了什么内容需要能说得出

数据库文件：见ssm.sql

pom.xml:

|  |
| --- |
| <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>1.9</maven.compiler.source>  <maven.compiler.target>1.9</maven.compiler.target>  <spring.version>5.0.2.RELEASE</spring.version> </properties>  <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</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.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>1.8.13</version>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.4.5</version>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>1.3.1</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.32</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.9</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok  get/set等注解 简化代码  -->  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <version>1.16.20</version>  <scope>provided</scope>  </dependency>  <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.datatype/jackson-datatype-guava  谷歌出得 功能强大  -->  <dependency>  <groupId>com.fasterxml.jackson.datatype</groupId>  <artifactId>jackson-datatype-guava</artifactId>  <version>2.9.7</version>  </dependency>  <!--logback-->  <!-- https://mvnrepository.com/artifact/ch.qos.logback/logback-core -->  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-core</artifactId>  <version>1.2.3</version>  </dependency>  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-classic</artifactId>  <version>1.2.3</version>  </dependency>  <dependency>  <groupId>org.logback-extensions</groupId>  <artifactId>logback-ext-spring</artifactId>  <version>0.1.4</version> </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>1.7.25</version>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>3.1.0</version>  <scope>provided</scope>  </dependency>   <!-- https://mvnrepository.com/artifact/commons-validator/commons-validator -->  <!--bean的校验-->  <dependency>  <groupId>commons-validator</groupId>  <artifactId>commons-validator</artifactId>  <version>1.6</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator -->  <dependency>  <groupId>org.hibernate.validator</groupId>  <artifactId>hibernate-validator</artifactId>  <version>6.0.13.Final</version>  </dependency>  <!--工具类-->  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  <version>1.3.1</version>  </dependency>  <dependency>  <groupId>commons-codec</groupId>  <artifactId>commons-codec</artifactId>  <version>1.9</version>  </dependency>  <!--jackson-->  <!-- https://mvnrepository.com/artifact/org.codehaus.jackson/jackson-core-asl -->  <dependency>  <groupId>org.codehaus.jackson</groupId>  <artifactId>jackson-core-asl</artifactId>  <version>1.9.13</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.codehaus.jackson/jackson-mapper-asl -->  <dependency>  <groupId>org.codehaus.jackson</groupId>  <artifactId>jackson-mapper-asl</artifactId>  <version>1.9.13</version>  </dependency>   <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  <version>3.5</version>  </dependency>  <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>5.1.2</version>  </dependency> </dependencies>  <build> <plugins>  <plugin>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.5.1</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat8-maven-plugin</artifactId>  <version>3.0-r1655215</version>  <configuration>  <path>/</path>  <server>tomcat8</server>  <port>8080</port>  </configuration>  </plugin> </plugins> </build>  <pluginRepositories>  <pluginRepository>  <id>alfresco-public</id>  <url>https://artifacts.alfresco.com/nexus/content/groups/public</url>  </pluginRepository>  <pluginRepository>  <id>alfresco-public-snapshots</id>  <url>https://artifacts.alfresco.com/nexus/content/groups/public-snapshots</url>  <snapshots>  <enabled>true</enabled>  <updatePolicy>daily</updatePolicy>  </snapshots>  </pluginRepository>  <pluginRepository>  <id>beardedgeeks-releases</id>  <url>http://beardedgeeks.googlecode.com/svn/repository/releases</url>  </pluginRepository> </pluginRepositories> |

Mybatis配置文件

|  |
| --- |
| <?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> </configuration> |

Mybatis-spring整合文件

applicationContext-dao.xml

|  |
| --- |
| <?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"  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">  <context:property-placeholder ignore-unresolvable="true" location="classpath:properties/jdbc.properties"></context:property-placeholder>  <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">  <property name="driverClassName" value="${jdbc.driverClass}"></property>  <property name="url" value="${jdbc.url}"></property>  <property name="username" value="${jdbc.username}"></property>  <property name="password" value="${jdbc.password}"></property>  <property name="initialSize" value="3"/>  <!--  maxIdle，最大空闲数，数据库连接的最大空闲时间。超过空闲时间，数据库连  接将被标记为不可用，然后被释放。设为0表示无限制。  MaxActive，连接池的最大数据库连接数。设为0表示无限制。  maxWait ，最大建立连接等待时间。如果超过此时间将接到异常。设为-1表示  无限制。  -->  <property name="minIdle" value="3"/>  <property name="maxActive" value="20"/>  <property name="maxWait" value="6000"/>  <property name="filters" value="stat,wall"/>  </bean>   <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  <property name="dataSource" ref="dataSource"></property>  <property name="configLocation" value="classpath:mybatis/mybatis.xml"/>  <property name="mapperLocations" value="classpath:mappers/\*.xml"/>  </bean>   <bean id="scanMapper" class="org.mybatis.spring.mapper.MapperScannerConfigurer">  <property name="basePackage" value="com.xmcc.dao"/>  <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>  </bean>  <!--druid配置-->  <bean id="stat-filter" class="com.alibaba.druid.filter.stat.StatFilter">  <!--慢查询-->  <property name="slowSqlMillis" value="3000"/>  <!--记录慢查询-->  <property name="logSlowSql" value="true"/>  <property name="mergeSql" value="true"/>  </bean>  <bean id="wall-filter" class="com.alibaba.druid.wall.WallFilter">  <property name="dbType" value="mysql"/>  </bean>  </beans> |

Jdbc.properties

|  |
| --- |
| jdbc.driverClass=com.mysql.jdbc.Driver jdbc.url=jdbc:mysql://localhost:3306/ssm?useUnicode=true&characterEncoding=utf8 jdbc.username=root jdbc.password=123456 |

applicationContext-serviceothers.xml

|  |
| --- |
| <?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"  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">  <context:component-scan base-package="com.xmcc.service"/>  </beans> |

applicationContext-tx.xml

|  |
| --- |
| <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  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/tx  http://www.springframework.org/schema/tx/spring-tx.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop.xsd">  <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  <property name="dataSource" ref="dataSource"/>  </bean>  <tx:advice id="advice" transaction-manager="transactionManager">  <tx:attributes>  <tx:method name="query\*" propagation="SUPPORTS" read-only="true"/>  <tx:method name="delete\*" propagation="REQUIRED" />  <tx:method name="update\*" propagation="REQUIRED"/>  <tx:method name="insert\*" propagation="REQUIRED" />  <tx:method name="add\*" propagation="REQUIRED"/>  </tx:attributes>  </tx:advice>  <aop:config>  <aop:pointcut id="point" expression="execution(\* com.xmcc.service.\*.\*(..))"></aop:pointcut>  <aop:advisor advice-ref="advice" pointcut-ref="point"/>  </aop:config> </beans> |

Web.xml

|  |
| --- |
| <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">    <display-name>Archetype Created Web Application</display-name>  <!--编码-->  <filter>  <filter-name>encoding</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>encoding</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>   <!--spring配置文件-->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext-\*.xml</param-value>  </context-param>    <!--springmvc配置文件-->  <servlet>  <servlet-name>dispatcherServlet</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>dispatcherServlet</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>   <!--  druid监控页面配置  -->  <servlet>  <servlet-name>druidStatServlet</servlet-name>  <servlet-class>com.alibaba.druid.support.http.StatViewServlet</servlet-class>  <init-param>  <param-name>loginUsername</param-name>  <param-value>xmcc</param-value>  </init-param>  <init-param>  <param-name>loginPassword</param-name>  <param-value>xmcc</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>druidStatServlet</servlet-name>  <url-pattern>/sys/druid/\*</url-pattern>  </servlet-mapping>  <filter>  <filter-name>druidWebStatFilter</filter-name>  <filter-class>com.alibaba.druid.support.http.WebStatFilter</filter-class>  <init-param>  <param-name>exclusions</param-name>  <param-value>\*.js,\*.css,\*.jpg,\*.png,\*.gif,\*.ico,/sys/druid/\*</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>druidWebStatFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!--日志-->  <!--<context-param>  <param-name>logbackConfigLocation</param-name>  <param-value>classpath:log/logback.xml</param-value>  </context-param>-->  <welcome-file-list>  <welcome-file>index.html</welcome-file>  </welcome-file-list>  <context-param>  <param-name>logbackConfigLocation</param-name>  <param-value>classpath:log/logback.xml</param-value> </context-param>  <listener>  <listener-class>ch.qos.logback.ext.spring.web.LogbackConfigListener</listener-class> </listener>  </web-app> |

Web.xml中配置了<servlet-name>为dispatchServlet,所以springmvc的配置文件可以直接放在web-inf下面并且以该名字开头-servlet命名,也可以用以前的配置然后通过contextParam传入

dispatchServlet-servlet.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xmlns:context="http://www.springframework.org/schema/context"  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/mvc  http://www.springframework.org/schema/mvc/spring-mvc.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd">   <context:annotation-config/>  <mvc:annotation-driven/>  <context:component-scan base-package="com.xmcc.controller"></context:component-scan>  <!--视图渲染 可以判断用什么来解析返回值-->  <bean class="org.springframework.web.servlet.view.BeanNameViewResolver"/>  <bean id="jsonView" class="org.springframework.web.servlet.view.json.MappingJackson2JsonView"/>  <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="prefix" value="/WEB-INF/pages/"></property>  <property name="suffix" value=".jsp"></property>  </bean>  </beans> |

Logback.xml日志配置

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <!--   scan: 当此属性设置为true时，配置文件如果发生改变，将会被重新加载，默认值为true。 scanPeriod: 设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当scan为true时， 此属性生效。默认的时间间隔为1分钟。  appender主要作用就是：①控制打印日志的地方、②打印日志的输出格式。 logger：用来设置某一个包或者具体的某一个类的日志打印级别、以及指定<appender> --> <configuration scan="true" scanPeriod="60 seconds">  <!--定义日志文件的存储地址 勿在 LogBack 的配置中使用相对路径-->  <property name="LOG\_HOME" value="F://ssmLog" />  <!--定义log的输出形式 指定输出到控制台-->  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">  <encoder>  <!--  -->  <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger - %msg%n</pattern>  </encoder>  </appender>  <!-- 按照每天生成日志文件 -->  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  <!--日志文件输出的文件名-->  <FileNamePattern>${LOG\_HOME}/ssm.log.%d{yyyy-MM-dd}.log</FileNamePattern>  <!--日志文件保留天数-->  <MaxHistory>30</MaxHistory>  </rollingPolicy>  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  <!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->  <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>  </encoder>  <!--日志文件最大的大小-->  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">  <MaxFileSize>10MB</MaxFileSize>  </triggeringPolicy>  </appender>  <!--  <appender name="permission" class="ch.qos.logback.core.rolling.RollingFileAppender">-->  <!--指定日志位置  <file>${catalina.home}/logs/permission.log</file>-->  <!--压缩规则  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">-->  <!--<FileNamePattern>${catalina.home}/logs/permission.%d{yyyy-MM-dd}.log.zip</FileNamePattern>-->  <!--</rollingPolicy>-->  <!--<layout class="ch.qos.logback.classic.PatternLayout">-->  <!--<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger - %msg%n</pattern>-->  <!--</layout>-->  <!--</appender>-->  <!---->  <!--在类中用这个名字的时候就会采用对应的appender  <logger name="xxx" level="INFO">-->  <!--<appender-ref ref="permission"/>-->  <!--</logger>-->   <!-- 指定级别 TRACE < DEBUG < INFO < WARN < ERROR -->  <root level="INFO">  <appender-ref ref="STDOUT" />  <appender-ref ref="FILE" />  </root>   </configuration> |

Mybatis反向工程见工程文件

开始测试（有问题一定自己解决）完整代码见ssm\_xmcc\_final\_01

1. mybatis篇（所有需要敲代码的，请敲代码测试）

1>什么是mybatis？

（1）mybatis是一个优秀的基于java的持久层框架，它内部封装了jdbc，使开发者只需要关注sql语句本身，而不需要花费精力去处理加载驱动、创建连接、创建statement等繁杂的过程。

（2）mybatis通过xml或注解的方式将要执行的各种statement配置起来，并通过java对象和statement中sql的动态参数进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射为java对象并返回。

（3）MyBatis 支持定制化 SQL、存储过程以及高级映射。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以使用简单的 XML 或注解来配置和映射原生信息，将接口和 Java 的 POJO映射成数据库中的记录。

2>Mybait的优点：

（1）简单易学，容易上手（相比于Hibernate） —- 基于SQL编程；

（2）JDBC相比，减少了50%以上的代码量，消除了JDBC大量冗余的代码，不需要手动开关连接；

（3）很好的与各种数据库兼容（因为MyBatis使用JDBC来连接数据库，所以只要JDBC支持的数据库MyBatis都支持，而JDBC提供了可扩展性，所以只要这个数据库有针对Java的jar包就可以就可以与MyBatis兼容），开发人员不需要考虑数据库的差异性。

（4）提供了很多第三方插件（分页插件 / 逆向工程）；

（5）能够与Spring很好的集成；

（6）MyBatis相当灵活，不会对应用程序或者数据库的现有设计强加任何影响，SQL写在XML里，从程序代码中彻底分离，解除sql与程序代码的耦合，便于统一管理和优化，并可重用。

（7）提供XML标签，支持编写动态SQL语句。

（8） 提供映射标签，支持对象与数据库的ORM字段关系映射。

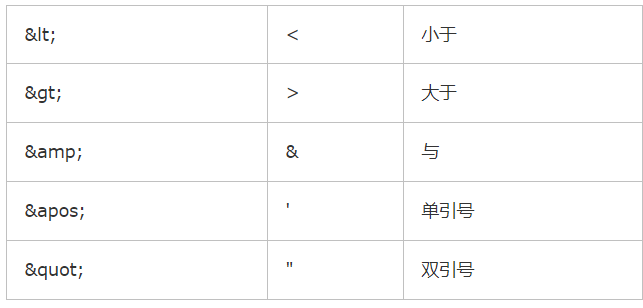
（9）提供对象关系映射标签，支持对象关系组建维护。

3>MyBatis框架的缺点：

（1）SQL语句的编写工作量较大，尤其是字段多、关联表多时，更是如此，对开发人员编写SQL语句的功底有一定要求。

（2）SQL语句依赖于数据库，导致数据库移植性差，不能随意更换数据库。

4>转义字符



当然啦， 我们也可以用另外一种，就是<![CDATA[ ]]>符号。 在mybatis中这种符号将不会解析

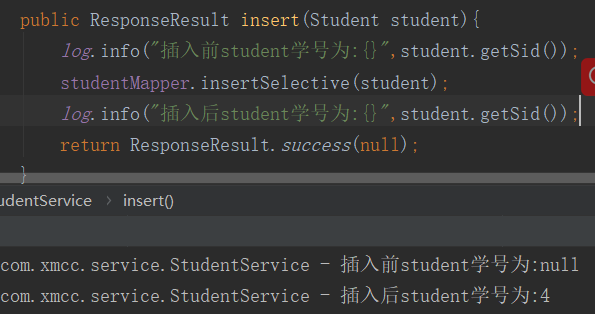
<![CDATA[ when min(starttime)<='12:00' and max(endtime)<='12:00' ]]>

5>返回主键

这儿两个表为例:

1.在mybatis自带的加入

<insert id="insertSelective" parameterType="com.xmcc.pojo.Student" useGeneratedKeys="true" keyProperty="sid">

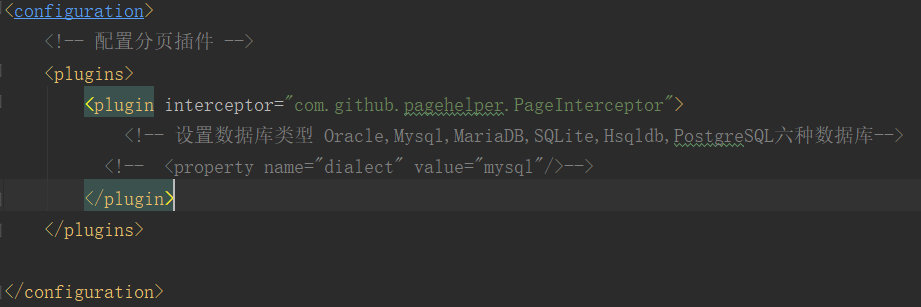


2.通过selectKey标签

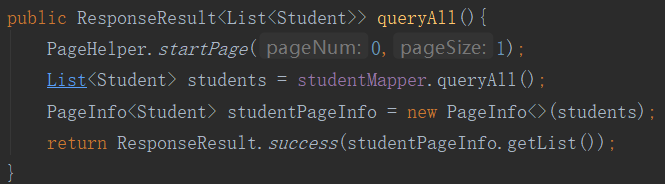
<insert id="insert" parameterType="com.xmcc.pojo.Student" >  
 <selectKey resultType="java.lang.Integer" order="AFTER" keyProperty="sid">  
 SELECT LAST\_INSERT\_ID()  
 </selectKey>  
 insert into student (sid, snmae, sbirthday,   
 tid)  
 values (#{sid,jdbcType=INTEGER}, #{snmae,jdbcType=VARCHAR}, #{sbirthday,jdbcType=DATE},   
 #{tid,jdbcType=INTEGER})  
</insert>

6>翻页插件使用

Mybatis配置文件中加入



使用:



7>mybatis多参数传递:

|  |
| --- |
| （1）第一种：  //DAO层的函数  Public UserselectUser(String name,String area);    //对应的xml,#{0}代表接收的是dao层中的第一个参数，#{1}代表dao层中第二参数，更多参数一致往后加即可。  <select id="selectUser"resultMap="BaseResultMap">  select \* fromuser\_user\_t whereuser\_name = #{0} anduser\_area=#{1}  </select>    （2）第2种： 使用 @param 注解:  public interface usermapper {  user selectuser(@param(“username”) string username,  @param(“hashedpassword”) string hashedpassword);  }  然后,就可以在xml像下面这样使用(推荐封装为一个map,作为单个参数传递给mapper):  <select id=”selectuser” resulttype=”user”>  select id, username, hashedpassword  from some\_table  where username = #{username}  and hashedpassword = #{hashedpassword}  </select> |

8>其他标签

除了常见的select|insert|updae|delete标签之外，还有哪些标签？

答：还有很多其他的标签，<resultMap>、<parameterMap>、<sql>、<include>、<selectKey>，加上动态sql的9个标签，trim|where|set|foreach|if|choose|when|otherwise|bind等，其中<sql>为sql片段标签，通过<include>标签引入sql片段，<selectKey>为不支持自增的主键生成策略标签。

9>一对一，一对多配置

自己改成我们的sql测试:

|  |
| --- |
| <mapper namespace="com.lcb.mapping.userMapper">  <!--association 一对一关联查询 -->  <select id="getClass" parameterType="int" resultMap="ClassesResultMap">  select \* from class c,teacher t where c.teacher\_id=t.t\_id and c.c\_id=#{id}  </select>    <resultMap type="com.lcb.user.Classes" id="ClassesResultMap">  <!-- 实体类的字段名和数据表的字段名映射 -->  <id property="id" column="c\_id"/>  <result property="name" column="c\_name"/>  <association property="teacher" javaType="com.lcb.user.Teacher">  <id property="id" column="t\_id"/>  <result property="name" column="t\_name"/>  </association>  </resultMap>      <!--collection 一对多关联查询 -->  <select id="getClass2" parameterType="int" resultMap="ClassesResultMap2">  select \* from class c,teacher t,student s where c.teacher\_id=t.t\_id and c.c\_id=s.class\_id and c.c\_id=#{id}  </select>    <resultMap type="com.lcb.user.Classes" id="ClassesResultMap2">  <id property="id" column="c\_id"/>  <result property="name" column="c\_name"/>  <association property="teacher" javaType="com.lcb.user.Teacher">  <id property="id" column="t\_id"/>  <result property="name" column="t\_name"/>  </association>    <collection property="student" ofType="com.lcb.user.Student">  <id property="id" column="s\_id"/>  <result property="name" column="s\_name"/>  </collection>  </resultMap>  </mapper> |

10>mybatis延迟加载与缓存:

|  |
| --- |
| Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？  答：Mybatis仅支持association关联对象和collection关联集合对象的延迟加载，association指的就是一对一，collection指的就是一对多查询。在Mybatis配置文件中，可以配置是否启用延迟加载lazyLoadingEnabled=true|false。  它的原理是，使用CGLIB创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用a.getB().getName()，拦截器invoke()方法发现a.getB()是null值，那么就会单独发送事先保存好的查询关联B对象的sql，把B查询上来，然后调用a.setB(b)，于是a的对象b属性就有值了，接着完成a.getB().getName()方法的调用。这就是延迟加载的基本原理。  当然了，不光是Mybatis，几乎所有的包括Hibernate，支持延迟加载的原理都是一样的。   Mybatis的一级、二级缓存?一级就是session、二级就是sqlsessionfactory  1）一级缓存: 基于 PerpetualCache 的 HashMap 本地缓存，其存储作用域为 Session，当 Session flush 或 close 之后，该 Session 中的所有 Cache 就将清空，默认打开一级缓存。  2）二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap 存储，不同在于其存储作用域为 Mapper(Namespace)，并且可自定义存储源，如 Ehcache。默认不打开二级缓存，要开启二级缓存，使用二级缓存属性类需要实现Serializable序列化接口(可用来保存对象的状态),可在它的映射文件中配置<cache/> ；  3）对于缓存数据更新机制，当某一个作用域(一级缓存 Session/二级缓存Namespaces)的进行了C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear。 |

11>#{}和${}的区别是什么？

#{}是预编译处理，${}是字符串替换。

Mybatis在处理#{}时，会将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值；

Mybatis在处理${}时，就是把${}替换成变量的值。

使用#{}可以有效的防止SQL注入，提高系统安全性。

12>dao简单实现原理

通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？

        Dao接口，就是人们常说的Mapper接口，接口的全限名，就是映射文件中的namespace的值，接口的方法名，就是映射文件中MappedStatement的id值，接口方法内的参数，就是传递给sql的参数。Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可唯一定位一个MappedStatement，举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到namespace为com.mybatis3.mappers.StudentDao下面id = findStudentById的MappedStatement。在Mybatis中，每一个<select>、<insert>、<update>、<delete>标签，都会被解析为一个MappedStatement对象。

        Dao接口里的方法，是不能重载的，因为是全限名+方法名的保存和寻找策略。

        Dao接口的工作原理是JDK动态代理，Mybatis运行时会使用JDK动态代理为Dao接口生成代理proxy对象，代理对象proxy会拦截接口方法，转而执行MappedStatement所代表的sql，然后将sql执行结果返回。

13>简述Mybatis的插件运行原理，以及如何编写一个插件。

答：Mybatis仅可以编写针对ParameterHandler、ResultSetHandler、StatementHandler、Executor这4种接口的插件，Mybatis使用JDK的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这4种接口对象的方法时，就会进入拦截方法，具体就是InvocationHandler的invoke()方法，当然，只会拦截那些你指定需要拦截的方法。

实现Mybatis的Interceptor接口并复写intercept()方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件。

14>Mybatis动态sql是做什么的？都有哪些动态sql？能简述一下动态sql的执行原理不？(如果模糊的同学请自己练习一下各个标签)

答：Mybatis动态sql可以让我们在Xml映射文件内，以标签的形式编写动态sql，完成逻辑判断和动态拼接sql的功能，Mybatis提供了9种动态sql标签trim|where|set|foreach|if|choose|when|otherwise|bind。

其执行原理为，使用OGNL从sql参数对象中计算表达式的值，根据表达式的值动态拼接sql，以此来完成动态sql的功能

15>mybatis调用存储过程（现在存储过程越用越少，能看懂就可以了）https://www.cnblogs.com/xdp-gacl/p/4270352.html

注：能有这么多够了如果问到其他的，无外乎就是动态代理加反射与解析xml，学会自己猜测，大胆的告诉面试官不知道，但是觉得应该是什么什么样的

1. spring篇

1>spring中用到哪些设计模式：后面设计模式在来谈论具体的

简单工厂、工厂方法、单例模式、[适配器](https://www.baidu.com/s?wd=%E9%80%82%E9%85%8D%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)、包装器、代理、观察者、策略、模板方法

2>spring是单例还是多例

通过scope属性或者@Scope注解来指定：

默认是单例，所以线程不安全

Scope的取值有：

（1）Singleton：一个Spring容器中只有一个Bean的实例，此为Spring的默认配置，全容器共享一个实例。

　　（2）Prototype：每次调用新建一个Bean实例。

　　（3）Request：Web项目中，给每一个 http request 新建一个Bean实例。

　　（4）Session：Web项目中，给每一个 http session 新建一个Bean实例。

　　（5）GlobalSession：这个只在portal应用中有用，给每一个 global http session 新建一个Bean实例。

3>spring IOC

IOC是inverse of control的缩写，控制反转。

意思把依赖对象时自己创建对象的控制权转移到spring容器

1. 降低了与引用对象之间的耦合，提高了各层代码的健壮性
2. 利用容器管理，方便了对对象的创建销毁的控制，也能合适的时候提高对象的重用

没有详细跟踪源码，自己理解的，先通过dom4j技术解析配置的xml文件拿到需要扫描的包路径，然后递归扫描所有的类，通过反射判断类中是否有注解来指定交给spring管理，如果有通过反射生成对象，放在一个currentMap中，key为类名首字母小写或者用户指定的名字或者类型，然后在通过反射获得交给spring管理类的属性，对属性中有autowire注解的对象通过key在map中取到对应的对象注入。

4>spring AOP

Aop为面向切面编程，应用广泛，例如springmvc中的拦截器，mybatis中的插件等

在spring中主要应用在事务上面，

Spring事务的实现主要时通过动态代理（主要对方法进行增强，判断是否抛出异常，如果抛出异常就在catch中回滚，没有就commit，当然异常也是可以在配置文件中指定的）与threadlocal（主要是一个请求一个线程副本来保证该次请求中用到的都是同一个connection）来完成的

Threadlocal:线程那儿讲

动态代理：jdk动态代理与cglib代理，设计模式那儿讲

5>spring生命周期(希望能大概记得几个，然后说：吧，这个只是死记的没有去跟踪过，也能比一个都答不出的好)

    1、实例化一个Bean－－也就是我们常说的new；

    2、按照Spring上下文对实例化的Bean进行配置－－也就是IOC注入；

    3、如果这个Bean已经实现了BeanNameAware接口，会调用它实现的setBeanName(String)方法，此处传递的就是Spring配置文件中Bean的id值

    4、如果这个Bean已经实现了BeanFactoryAware接口，会调用它实现的setBeanFactory(setBeanFactory(BeanFactory)传递的是Spring工厂自身（可以用这个方式来获取其它Bean，只需在Spring配置文件中配置一个普通的Bean就可以）；

    5、如果这个Bean已经实现了ApplicationContextAware接口，会调用setApplicationContext(ApplicationContext)方法，传入Spring上下文（同样这个方式也可以实现步骤4的内容，但比4更好，因为ApplicationContext是BeanFactory的子接口，有更多的实现方法）；

    6、如果这个Bean关联了BeanPostProcessor接口，将会调用postProcessBeforeInitialization(Object obj, String s)方法，BeanPostProcessor经常被用作是Bean内容的更改，并且由于这个是在Bean初始化结束时调用那个的方法，也可以被应用于内存或缓存技术；

    7、如果Bean在Spring配置文件中配置了init-method属性会自动调用其配置的初始化方法。

    8、如果这个Bean关联了BeanPostProcessor接口，将会调用postProcessAfterInitialization(Object obj, String s)方法、；

    注：以上工作完成以后就可以应用这个Bean了，那这个Bean是一个Singleton的，所以一般情况下我们调用同一个id的Bean会是在内容地址相同的实例，当然在Spring配置文件中也可以配置非Singleton，这里我们不做赘述。

    9、当Bean不再需要时，会经过清理阶段，如果Bean实现了DisposableBean这个接口，会调用那个其实现的destroy()方法；

    10、最后，如果这个Bean的Spring配置中配置了destroy-method属性，会自动调用其配置的销毁方法。

以上10步骤可以作为面试或者笔试的模板，另外我们这里描述的是应用Spring上下文Bean的生命周期，如果应用Spring的工厂也就是BeanFactory的话去掉第5步就Ok了

注：重点就这几个大家一定要把这个博客看一看掌握其中百分之90：https://blog.csdn.net/liuhouli923914981/article/details/79763814

1. springmvc篇
2. 拦截器

Springmvc拦截器基于动态代理的aop实现，用到责任链模式(设计模式中讲解)

第一种是实现HandlerInterceptor接口(或者继承实现了该接口的子类)

第二种是实现WebRequestInterceptor接口(或者继承实现了该接口的子类)

不同点

1.WebRequestInterceptor的入参WebRequest是包装了HttpServletRequest 和HttpServletResponse的，通过WebRequest获取Request中的信息更简便。  
2.WebRequestInterceptor的preHandle是没有返回值的，说明该方法中的逻辑并不影响后续的方法执行，所以这个接口实现就是为了获取Request中的信息，或者预设一些参数供后续流程使用。  
3.HandlerInterceptor的功能更强大也更基础，可以在preHandle方法中就直接拒绝请求进入controller方法

都有三个方法:

**preHandle：**预处理回调方法，实现处理器的预处理（如登录检查），第三个参数为响应的处理器（如我们上一章的Controller实现）；

     返回值：true表示继续流程（如调用下一个拦截器或处理器）；

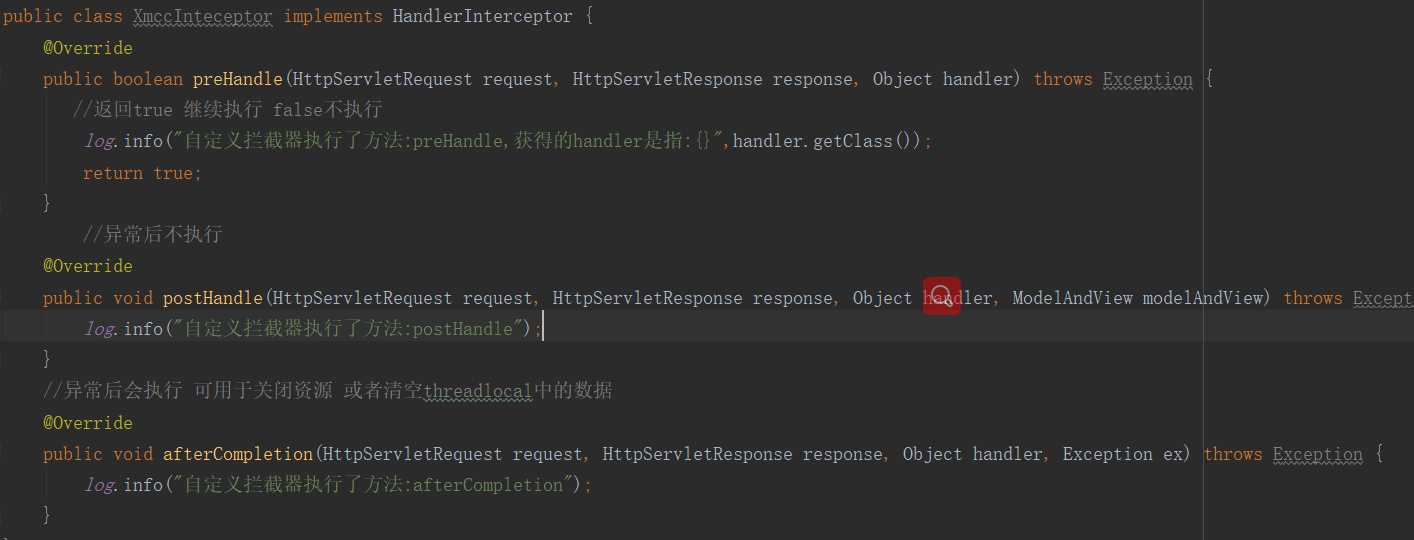
             false表示流程中断（如登录检查失败），不会继续调用其他的拦截器或处理器，此时我们需要通过response来产生响应；

**postHandle：**后处理回调方法，实现处理器的后处理（但在渲染视图之前），此时我们可以通过modelAndView（模型和视图对象）对模型数据进行处理或对视图进行处理，modelAndView也可能为null。

**afterCompletion：**整个请求处理完毕回调方法，即在视图渲染完毕时回调，如性能监控中我们可以在此记录结束时间并输出消耗时间，还可以进行一些资源清理，类似于try-catch-finally中的finally，但仅调用处理器执行链中**preHandle返回true的拦截器的afterCompletion**

实现步骤(以HandlerInterceptor为例)

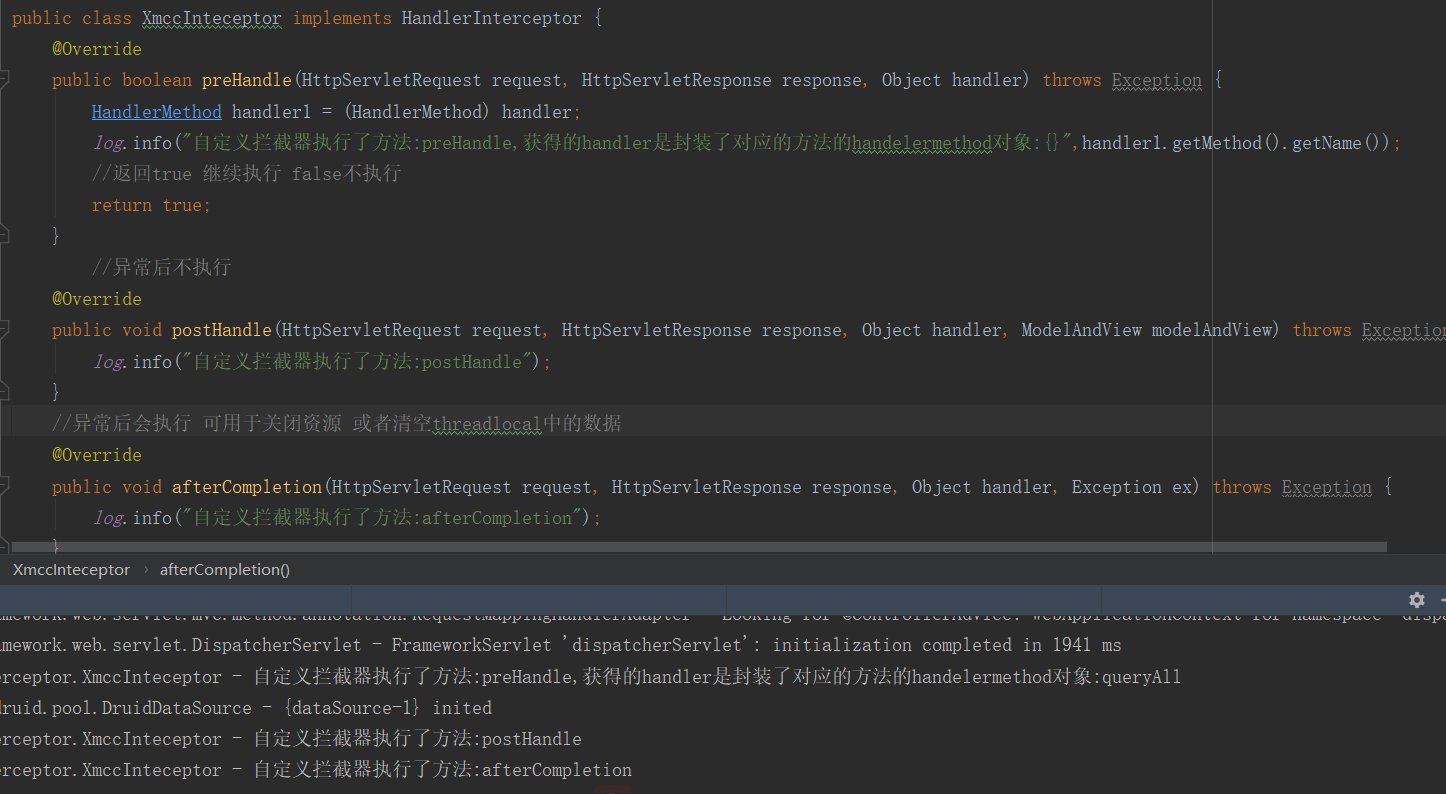
1. 自定义类实现HandlerInterceptor接口



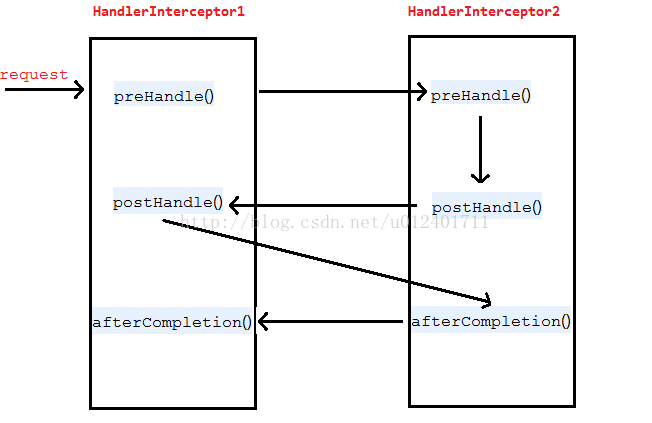
1. 配置拦截器生效



1. 执行



多拦截器就在配置那儿加<mvc:interceptor>，执行顺序如下，责任链模式



1. Json返回处理

配置方法不统一：我们这儿配置是:

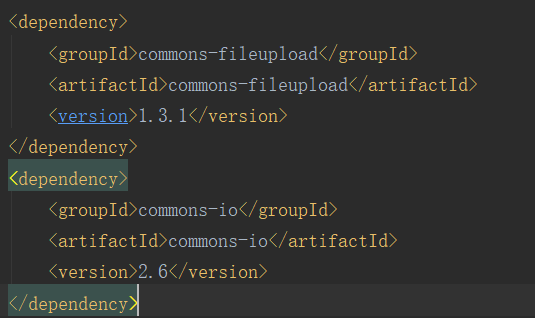
<!--  
<mvc:annotation-driven /> 会自动注册DefaultAnnotationHandlerMapping(注解映射器)与AnnotationMethodHandlerAdapter （注解适配器）  
-->  
<mvc:annotation-driven/>  
<!--视图渲染 可以判断用什么来解析返回值-->  
<bean class="org.springframework.web.servlet.view.BeanNameViewResolver"/>  
<bean id="jsonView" class="org.springframework.web.servlet.view.json.MappingJackson2JsonView"/>

然后@responseBody或者@restController就可以了

3>文件上传下载

一、文件上传:

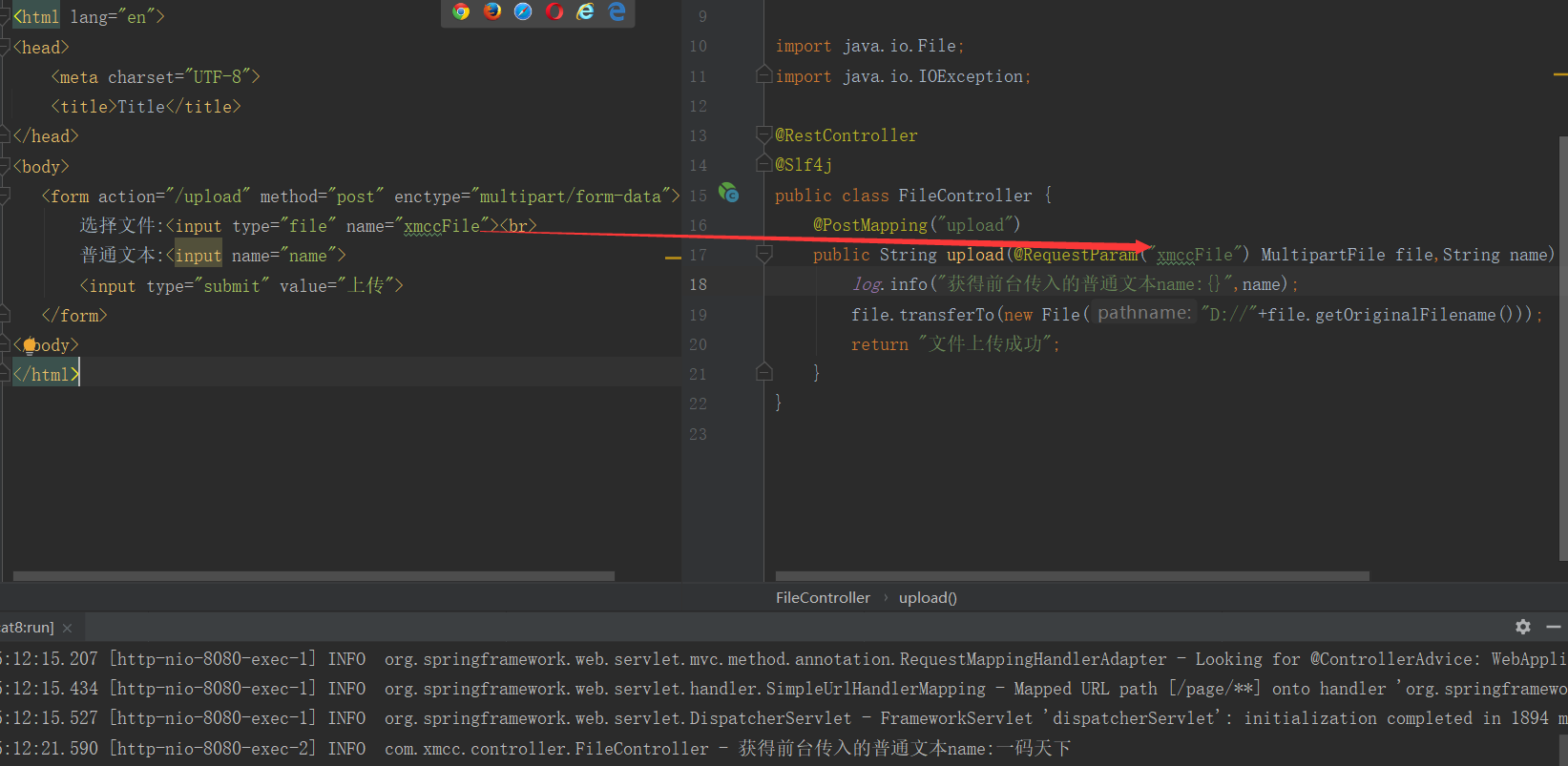
1. 导入jar



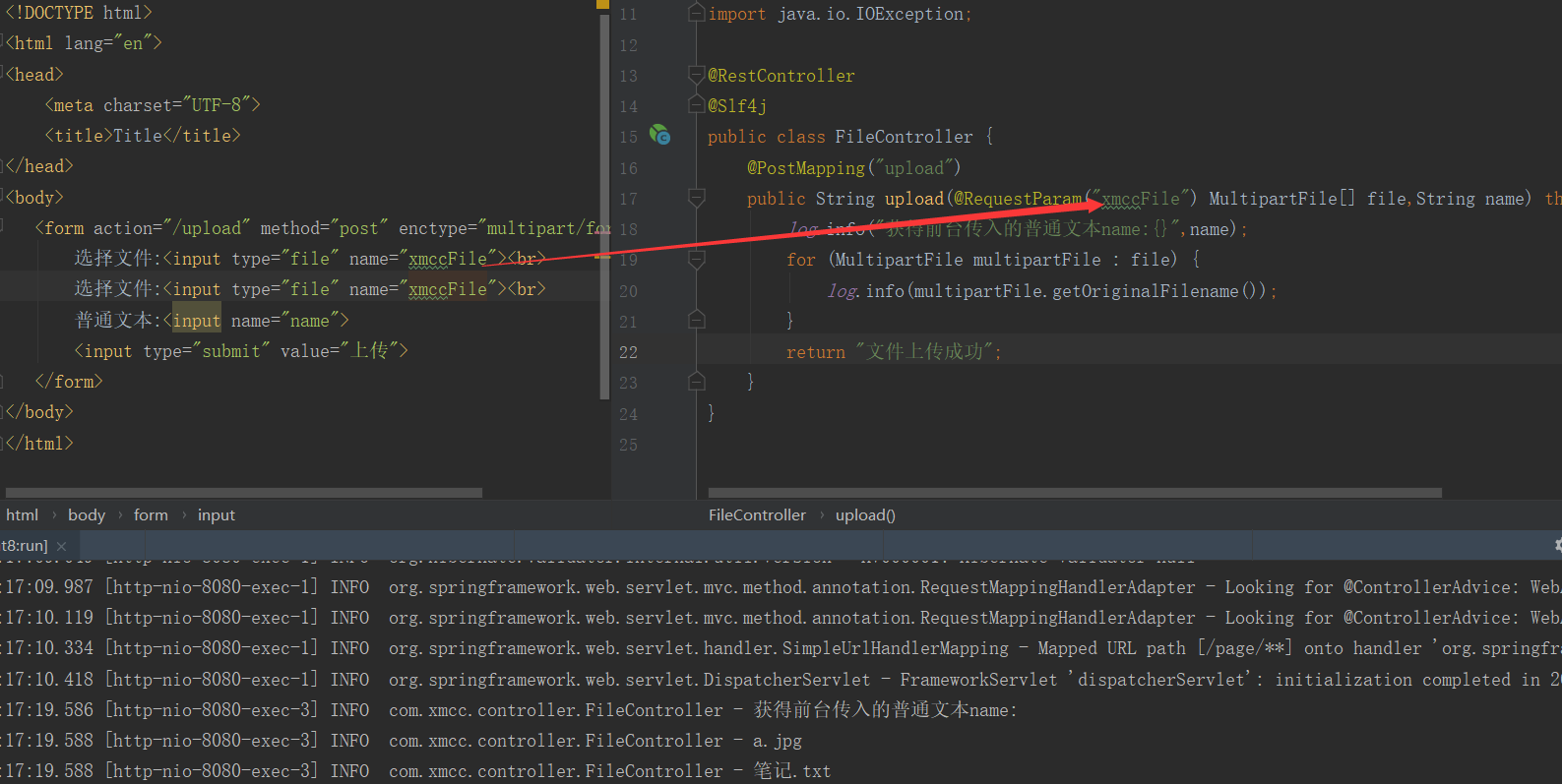
1. 配置文件解析器

<bean id="multipartResolver"  
 class="org.springframework.web.multipart.commons.CommonsMultipartResolver">  
 <!--上传文件的最大大小-->  
 <property name="maxUploadSize" value="17367648787"></property>  
 <!--上传文件的编码 -->  
 <property name="defaultEncoding" value="UTF-8"></property>  
</bean>

1. 前端form表单与后台:



1. 多文件



5>后台需要封装这儿就不封装了 文件名字需要不重复的字符串大家应该都知道怎么处理，如果需要存储到其他服务器(如fastdfs)则可以通过muiltpartFile获得流

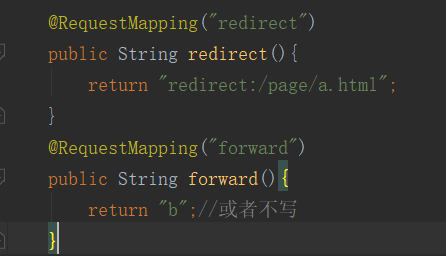
二、文件下载



还有一种是ResponseEntity 但是有局限，暂时不需要掌握 如果上班别人用这个也很简单,

文件上传下载都需要封装，这儿没有封装

4>重定向与跳转



也可以用原生的request

1. 静态资源放过
2. web.xml配置

|  |
| --- |
| <servlet-mapping>  <servlet-name>default</servlet-name>  <url-pattern>\*.jpg</url-pattern>  </servlet-mapping>  <servlet-mapping>  <servlet-name>default</servlet-name>  <url-pattern>\*.js</url-pattern>  </servlet-mapping>  <servlet-mapping>  <servlet-name>default</servlet-name>  <url-pattern>\*.css</url-pattern>  </servlet-mapping> |

1. mvc配置文件配置

<mvc:resources mapping="/images/\*\*" location="/images/" />

3. <mvc:default-servlet-handler/>

1. 转换器

这儿以日期转换器的全局转换器为例：（其他的了解一下@InitBinder

）

1.自定义全局转换器

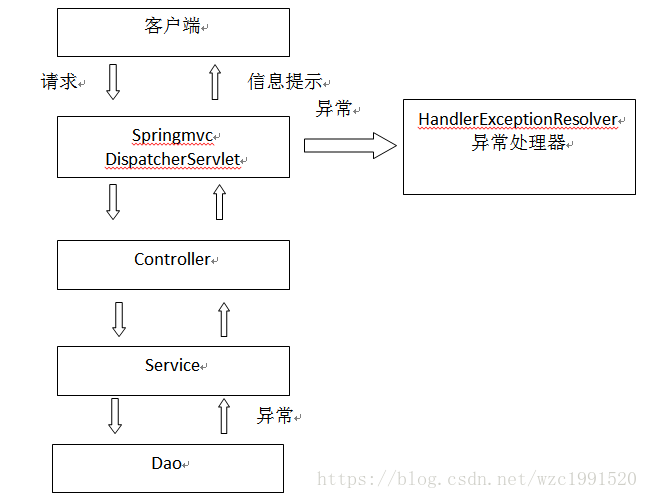
|  |
| --- |
| public class DateConverter implements Converter<String, Date> {  @Override  public Date convert(String source) {  SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");  dateFormat.setLenient(false);  try {  return dateFormat.parse(source);  } catch (ParseException e) {  e.printStackTrace();  }  return null;  } |

2.配置全局转换器

|  |
| --- |
| <bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">  <property name="converters">  <list>  <bean class="com.doje.XXX.web.DateConverter" />  </list>  </property>  </bean> |

|  |
| --- |
| <mvc:annotation-driven conversion-service="conversionService" /> 对应上面的id |

1. 自定义异常处理方案（这个需要多思考）



org.springframework.web.servlet.HandlerExceptionResolver（处理器异常解析器）只是一个接口，需要自己继承并实现

步骤：

1.自定义异常类

public class CustomException extends Exception {

    /\*\* serialVersionUID\*/

    private static final long serialVersionUID = -5212079010855161498L;

    public CustomException(String message){

        super(message);

        this.message = message;

    }

    //异常信息

    private String message;

    public String getMessage() {

        return message;

    }

    public void setMessage(String message) {

        this.message = message;

    }

}

2.    自定义异常处理器

public class CustomExceptionResolver implements HandlerExceptionResolver {

    @Override

    public ModelAndView resolveException(HttpServletRequest request,

            HttpServletResponse response, Object handler, Exception ex) {

        ex.printStackTrace();

        CustomException customException = null;

        //如果抛出的是系统自定义异常则直接转换

        if(ex instanceof CustomException){

            customException = (CustomException)ex;

        }else{

            //如果抛出的不是系统自定义异常则重新构造一个系统错误异常。

            customException = new CustomException("系统错误，请与系统管理 员联系！");

        }

        ModelAndView modelAndView = new ModelAndView();

        modelAndView.addObject("message", customException.getMessage());

        modelAndView.setViewName("error");

        return modelAndView;

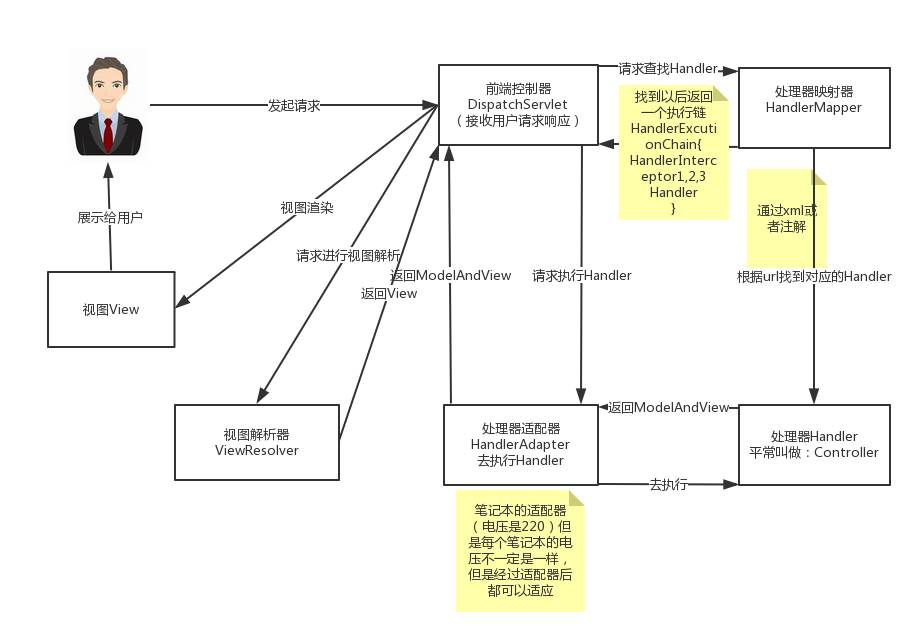
    }

}

3.在springmvc.xml的配置

<bean id="handlerExceptionResolver" class="xxx.CustomExceptionResolver"/>

1. Springmvc流程及核心类的作用



第一步:用户发起请求到前端控制器（DispatcherServlet）

第二步：前端控制器请求处理器映射器（HandlerMappering）去查找处理器（Handle）：通过xml配置或者注解进行查找

第三步：找到以后处理器映射器（HandlerMappering）像前端控制器返回执行链（HandlerExecutionChain）

第四步：前端控制器（DispatcherServlet）调用处理器适配器（HandlerAdapter）去执行处理器（Handler）

第五步：处理器适配器去执行Handler

第六步：Handler执行完给处理器适配器返回ModelAndView

第七步：处理器适配器向前端控制器返回ModelAndView

第八步：前端控制器请求视图解析器（ViewResolver）去进行视图解析

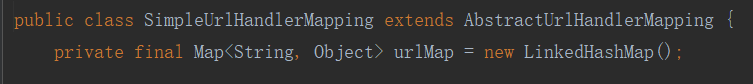
第九步：视图解析器像前端控制器返回View

第十步：前端控制器对视图进行渲染

第十一步：前端控制器向用户响应结果

其中handemapping作用：根据请求的url查找Handler（包含controller与拦截器链）

例如



Key就是url路径 value就是对应的控制器

1. BeanNameUrlHandlerMapping 根据指定的名字来匹配



1. SimpleUrlHandlerMapping:注解的时候就是将requestMapping中的url作为key(建议使用)

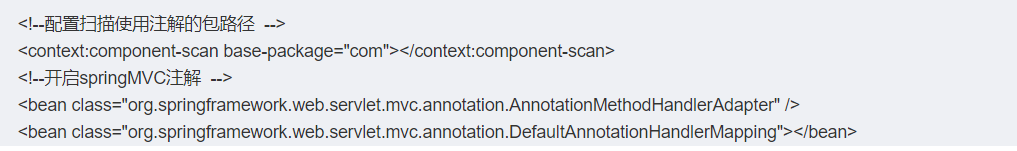


1. ControllerClassNameHandlerMapping



4.使用注解来映射寻找controller：

DefaultAnnotationHandlerMapping和RequestMappingHandlerMapping都是处理requestMapping的注解，  
前者以废弃，不建议使用，后者用来代替前者



使用<mvc:annotation-driven/> 默认就开启了

HandlerAdapter：HandlerAdapter用于调用处理器方法，并且为处理器方法提供参数解析、返回值处理等适配工作，使使用者专心于业务逻辑的实现

1. 非注解处理适配器：

1.第一个是 SimpleControllerHandlerAdapter，翻译过来就是简单的控制器处理器适配器，它支持所有实现了 Controller 接口的 Handler 控制器，如果开发中编写了实现 Controller 接口的控制器，则 SimpleControllerHandlerAdapter 适配器就会去执行 Controller 的具体方法

2. 第二个是 HttpRequestHandlerAdapter，翻译过来就是 http 请求处理器适配器，它要求编写的 Handler 需要实现 HttpRequestHandler 接口。使用这种 Handler 的开发方式，方便开发者获取 request 的相关 http 请求信息，以及设置返回对象 response 的一些参数

2. 注解处理适配器：

 RequestMappingHandlerAdapter：

开启方式:1.

|  |
| --- |
| <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/> |

开启方式2:

<mvc:annotation-driven></mvc:annotation-driven>

1. 整合jquery完成ajax与文件上传

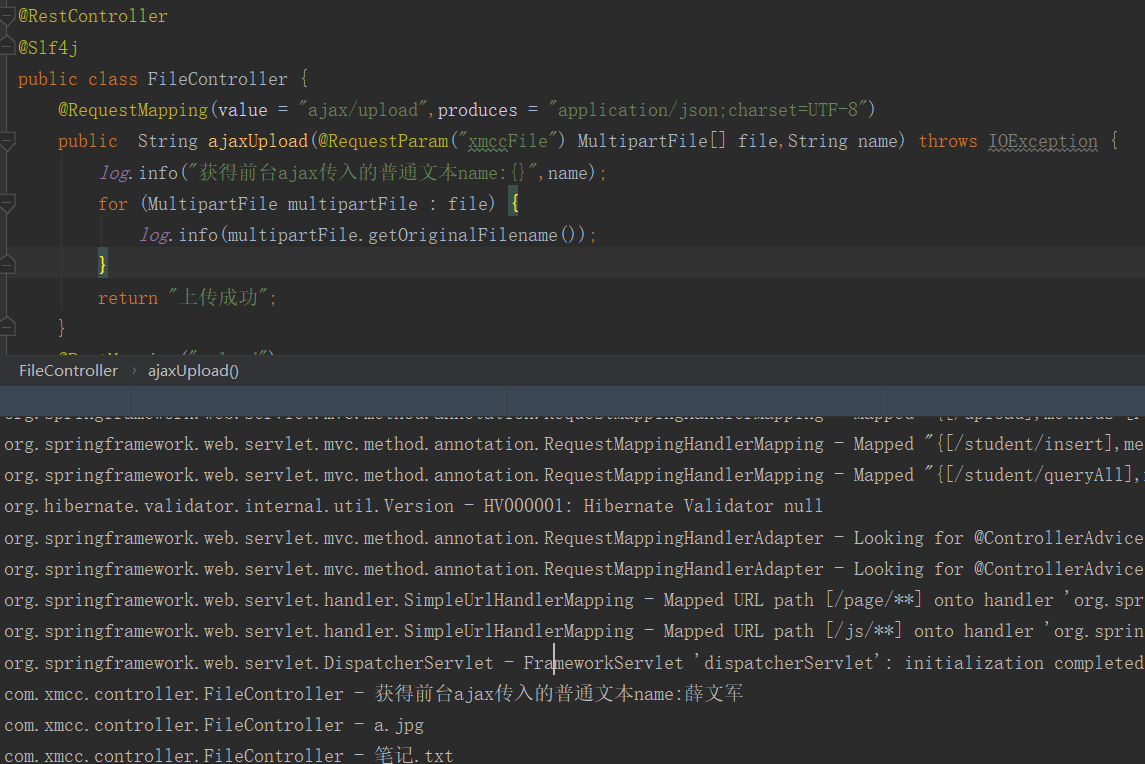
请自己写一下顺便复习下jquery的ajax，后台可以获得普通与文件，忽略的上传

1.引入juqery: <script src="../js/jquery-1.8.3.min.js"></script>

2.前台：

|  |
| --- |
| <!DOCTYPE html> <html lang="en"> <head>  <meta charset="UTF-8">  <title>Title</title>  <script src="../js/jquery-1.8.3.min.js"></script>  <script>  $(**function** () {  $("#upload").click(**function** () {  **var** formData= **new** FormData($("#uploadForm")[0]);  $.ajax({  type: "post",  url: "/ajax/upload",  dataType: "text",  data: formData,  */\*\*  \*必须false才会自动加上正确的Content-Type  \*/* contentType : **false**,  */\*\*  \* 必须false才会避开jQuery对 formdata 的默认处理  \* XMLHttpRequest会对 formdata 进行正确的处理  \*/* processData : **false**,  success: **function**(data){//从后端返回数据进行处理  alert(data);  },  error: **function**(err) {//提交出错  //$("#msg").html(JSON.stringify(err));//打出响应信息  alert("服务器无响应");  }  })  })  })  </script> </head> <body>  <form action="/upload" method="post" enctype="multipart/form-data" id="uploadForm">  选择文件:<input type="file" name="xmccFile"><br>  选择文件:<input type="file" name="xmccFile"><br>  普通文本:<input name="name">  <input type="submit" value="上传">  </form>  <button id="upload">ajax上传文件</button> <a href="/download">文件下载</a> </body> </html> |

3.后台：



1. 常用注解（需要记住百分之90以上，不用慌springboot会用得更多）

1、@Controller

2、@RequestMapping

RequestMapping是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。

RequestMapping注解有六个属性，下面我们把她分成三类进行说明（下面有相应示例）。

**1> value， method；**

value：     指定请求的实际地址，指定的地址可以是URI Template 模式（后面将会说明）；

method：  指定请求的method类型， GET、POST、PUT、DELETE等；

2>consumes，produces

consumes： 指定处理请求的提交内容类型（Content-Type），例如application/json, text/html;

produces:    指定返回的内容类型，仅当request请求头中的(Accept)类型中包含该指定类型才返回；

**3>params，headers**

params： 指定request中必须包含某些参数值是，才让该方法处理。

headers： 指定request中必须包含某些指定的header值，才能让该方法处理请求。

3、@Autowired

4、@ModelAttribute和 @SessionAttributes

代表的是：该Controller的所有方法在调用前，先执行此@ModelAttribute方法，可用于注解和方法参数中，可以把这个@ModelAttribute特性，应用在BaseController当中，所有的Controller继承BaseController，即可实现在调用Controller时，先执行@ModelAttribute方法。

 @SessionAttributes即将值放到session作用域中，写在class上面

5、@PathVariable(restful风格需要用到)

用于将请求URL中的模板变量映射到功能处理方法的参数上，即取出uri模板中的变量作为参数

6、@requestParam

@requestParam主要用于在SpringMVC后台控制层获取参数，类似一种是request.getParameter("name")，它有三个常用参数：defaultValue = "0", required = false, value = "isApp"；defaultValue 表示设置默认值，required 铜过boolean设置是否是必须要传入的参数，value 值表示接受的传入的参数类型

7、@ResponseBody

8、restful风格/ @getMapping @postMapping @deleteMapping等等

9 、@Bean 注解在方法上，声明当前方法的返回值为一个bean，替代xml中的方式（方法上）

10、@Configuration 声明当前类为配置类，其中内部组合了@Component注解，表明这个类是一个bean（类上）

11、@ComponentScan 用于对Component进行扫描，相当于xml中的（类上）

12、@WishlyConfiguration 为@Configuration与@ComponentScan的组合注解，可以替代这两个

13.@Enable\*注解说明

这些注解主要用来开启对xxx的支持。

@EnableAspectJAutoProxy 开启对AspectJ自动代理的支持

@EnableAsync 开启异步方法的支持

@EnableScheduling 开启计划任务的支持

@EnableWebMvc 开启Web MVC的配置支持

@EnableConfigurationProperties 开启对@ConfigurationProperties注解配置Bean的支持

@EnableJpaRepositories 开启对SpringData JPA Repository的支持

@EnableTransactionManagement 开启注解式事务的支持

@EnableTransactionManagement 开启注解式事务的支持

@EnableCaching 开启注解式的缓存支持

14.测试相关注解

@RunWith 运行器，Spring中通常用于对JUnit的支持

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration 用来加载配置ApplicationContext，其中classes属性用来加载配置类

@ContextConfiguration(classes={TestConfig.class})

1. 整合poi完成excel导入导出

这里只是简单入门，因为上班叫你自己封装poi工具概率不大，但是只要入门了，封装也就是查api的事情，掌握几个注意点，

面试的时候会问到你poi导入需要注意什么点？

1. 版本问题

2003版本是以xls结尾，每个sheet只能存储65535条数据，用的是hssfworkbook

2007版本是以xlsx结尾，能存储近104万条,用的是xssfworkbook

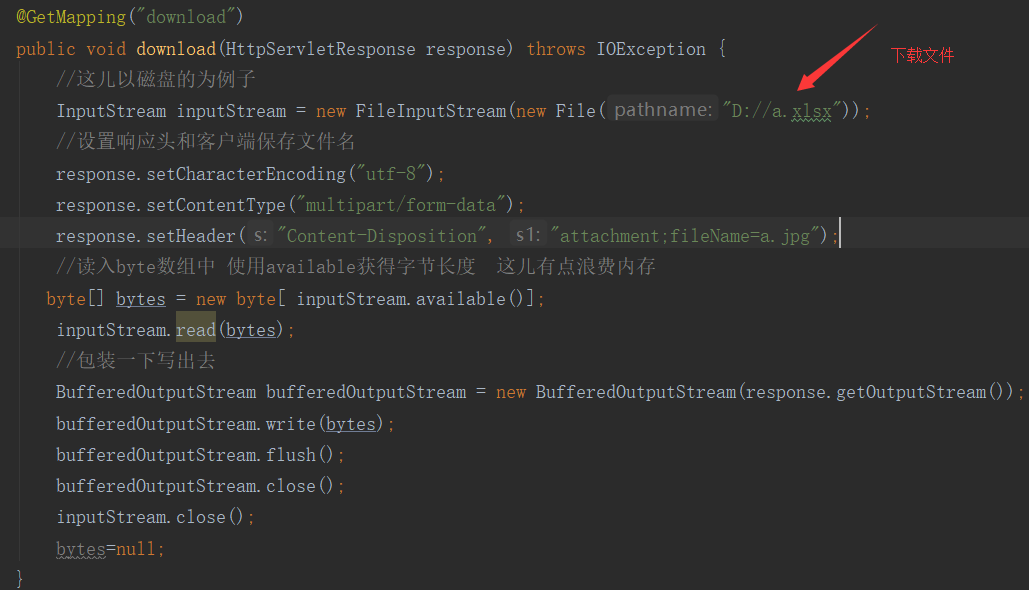
1. 格式问题，要求用户下载模板
2. 大数据量问题，百万数据咋导入导出

https://blog.csdn.net/happyljw/article/details/52809244

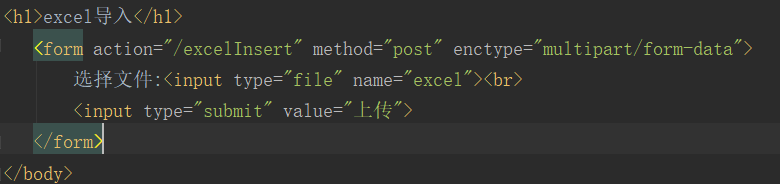
1. 导入jar包

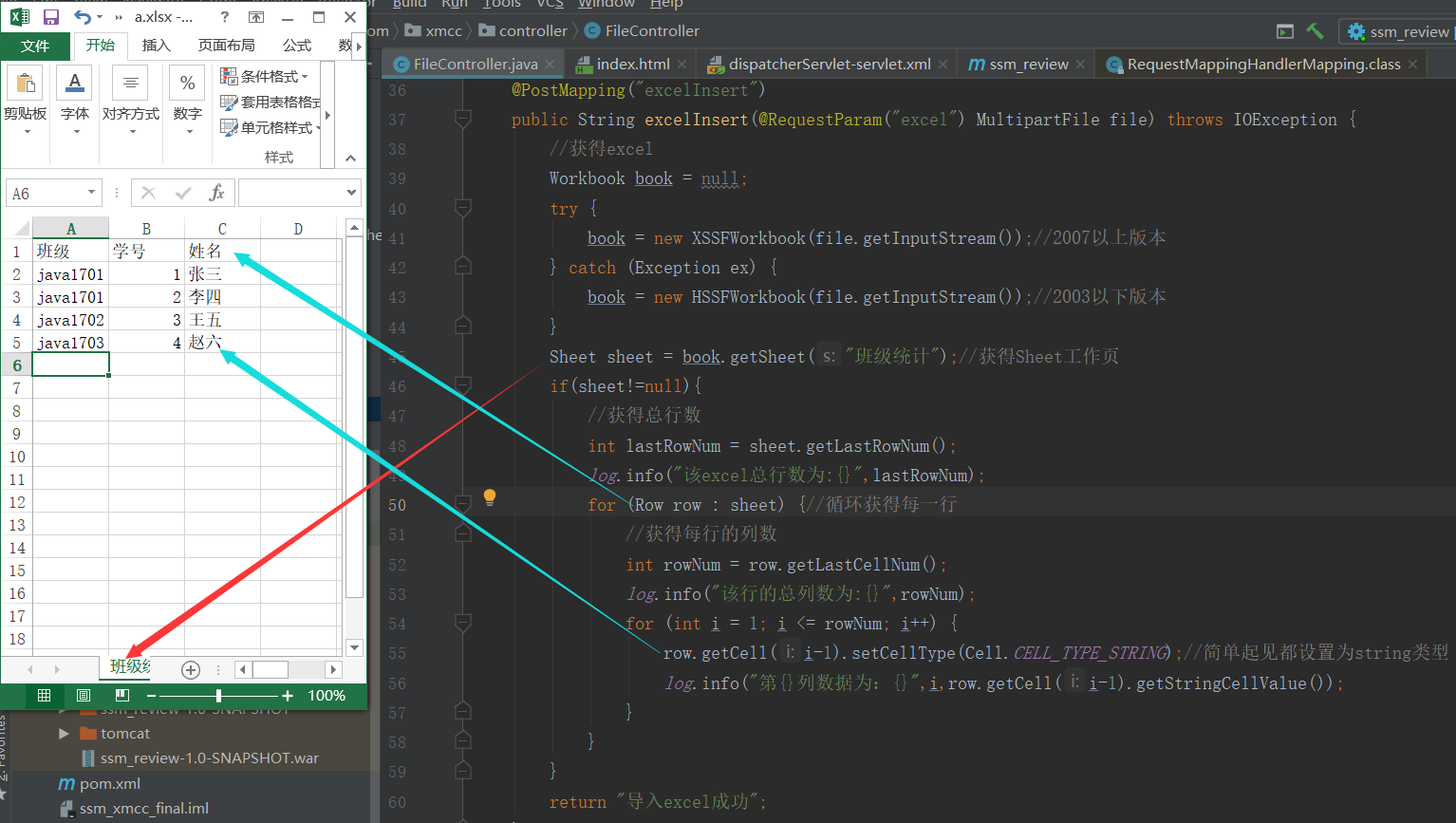
|  |
| --- |
| <dependency>  <groupId>org.apache.poi</groupId>  <artifactId>poi-ooxml</artifactId>  <version>3.5-FINAL</version> </dependency> |
|  |

1. 书写模板，所有得excel导入导出必须有模板的，用户在使用前需要在我们这里下载模板，写好以后用4.3的技术下载excel



1. 下载好后，由工作人员录入数据，然后导入excel文件（没有写入数据库，既然都能拿到数据，写入数据库同学们没问题吧？）





4.导出数据库数据到excel

1.从数据库查出数据

2.写入excel 下载文件就是

|  |
| --- |
| @GetMapping("exportExcel") public void exportExcel(HttpServletResponse response) throws IOException {  //从数据库查询数据  List<ExcelPojo> list = Lists.*newArrayList*();  list.add(new ExcelPojo("张三",10,new Date()));  list.add(new ExcelPojo("李四",10,new Date()));  list.add(new ExcelPojo("王五",10,new Date()));  Workbook book = null;  try {  book = new XSSFWorkbook();//2007以上版本  } catch (Exception ex) {  book = new HSSFWorkbook();//2003以下版本  }  Sheet sheet = book.createSheet("excel导出测试");//创建页sheet  //创建标题行  Row row = sheet.createRow(0);  row.createCell(0).setCellValue("姓名");  row.createCell(1).setCellValue("年龄");  row.createCell(2).setCellValue("生日");  //获得需要插入的数据  int size = list.size();  for (int i = 0; i < size; i++) {  Row row1 = sheet.createRow(i + 1);  row1.createCell(0).setCellValue(list.get(i).getName());//因为知道是三行 可以直接写  row1.createCell(1).setCellValue(list.get(i).getAge());  row1.createCell(2).setCellValue(list.get(i).getBirthday());  }  //设置响应头和客户端保存文件名  response.setCharacterEncoding("utf-8");  response.setContentType("multipart/form-data");  response.setHeader("Content-Disposition", "attachment;fileName=excel.xlsx");  OutputStream outputStream = response.getOutputStream();  book.write(outputStream);  outputStream.flush();  outputStream.close(); } |

Poi导入导出结束了，当然只是简单的demo，同学们只需要掌握大流程，就没问题了，上班需要的时候再查询api，以这个为基础增加就可以了，大胆的说自己用过

1. 整合echarts完成图表制作

这个比较简单就是去官网找你需要的图，然后数据从后台传来就可以了，官网：

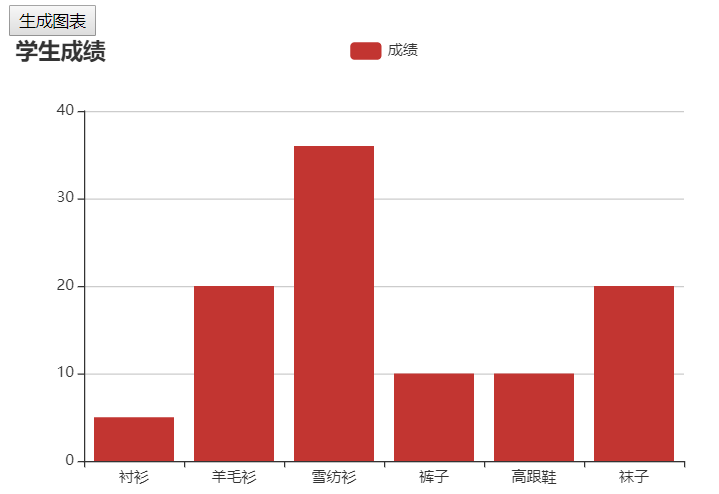
<https://www.echartsjs.com/tutorial.html#5%20%E5%88%86%E9%92%9F%E4%B8%8A%E6%89%8B%20ECharts>

简单完成一个案例：其他案例照着做就行了 只需要关注数据点，所有的图表在官网都有

一共有两种方式1、在后台完成图表的制作 2、是在前台 面试就说前台做的只需要传递数据

|  |
| --- |
| <!DOCTYPE html> <html lang="en"> <head>  <meta charset="UTF-8">  <title>Title</title>  <script src="../js/echarts.common.min.js"></script>  <script src="../js/jquery-1.8.3.min.js"></script>  <script>  $(**function** () {  $("#echarts").click(**function** () {  $.ajax({  type:"GET",  url:"/getEcharts",  dataType:"json",  success:**function** (data) {  // 基于准备好的dom，初始化echarts实例  **var** myChart = echarts.init(document.getElementById('main'));  // 指定图表的配置项和数据  **var** option = {  title: {  text: '学生成绩'  },  tooltip: {},  legend: {  data:['成绩']  },  xAxis: {  data: data.course  },  yAxis: {},  series: [{  name: '成绩',  type: 'bar',//指定图表类型  data: data.score  }]  };   // 使用刚指定的配置项和数据显示图表。  myChart.setOption(option);  }  })  })  })   </script> </head> <body> <button id="echarts">生成图表</button> <!-- 为 ECharts 准备一个具备大小（宽高）的 DOM --> <div id="main" style="width: 600px;height:400px;"></div> </body> </html> |

|  |
| --- |
| package com.xmcc.controller;  import com.google.common.collect.Lists; import com.google.common.collect.Maps; import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RestController;  import java.util.List; import java.util.Map;  @RestController public class EchartsController {  @GetMapping("getEcharts")  public Map<String,Object> getEcharts(){//这儿就用map测试了  List<String> courseList = Lists.*newArrayList*("衬衫","羊毛衫","雪纺衫","裤子","高跟鞋","袜子");  List<Integer> scoreList = Lists.*newArrayList*(5, 20, 36, 10, 10, 20);  Map<String,Object> map = Maps.*newHashMap*();  map.put("course",courseList);  map.put("score",scoreList);  return map;  } } |



1. 整合过滤器完成登录校验

核心思想是，在过滤器中拦截除登录注册以外的所有请求，当然也可以根据项目业务自行放过所有请求，因为登录成功后会把用户数据放在session中，所以在filter中拦截的时候只需要判断session中是否有当前用户的登录信息就可以了。下面是代码:

Request获得路径：

|  |
| --- |
| 示例： 当前url：http://localhost:8080/CarsiLogCenter\_new/idpstat.jsp?action=idp.sptopn  request.getRequestURL() ：http://localhost:8080/CarsiLogCenter\_new/idpstat.jsp  request.getRequestURI() ：/CarsiLogCenter\_new/idpstat.jsp  request.getContextPath()：/CarsiLogCenter\_new  request.getServletPath()： /idpstat.jsp  request.getQueryString()：action=idp.sptopn |

最基本的一个登录拦截: 以后复杂的也是在这上面扩展,controller层就不写了，就是调用数据库查询

|  |
| --- |
| @WebFilter("/\*") public class LoginFilter implements Filter {  List<String> ignoreUrl = Lists.*newArrayList*();  @Override  public void init(FilterConfig filterConfig) throws ServletException {  //在这里可以拿到web.xml中配置的参数  //初始化需要放过的路径  ignoreUrl.add("login");  ignoreUrl.add("register");  }   @Override  public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {  HttpServletRequest request = (HttpServletRequest) servletRequest;  HttpServletResponse response = (HttpServletResponse) servletResponse;  String requestURI = request.getRequestURI();  //去掉项目需要放过的请求，如登录注册等 可以将这些路径放在一个集合中 在init方法中初始化  if(requestURI!=null){  boolean flag = false;  for (String s : ignoreUrl) {  if(requestURI.endsWith(s)){  flag = true;  break;  }  }  if(!flag){//说明不是直接放过的路径  HttpSession session = request.getSession();  User user = (User) session.getAttribute("user");//这儿应该是与controller层登录时候放入session的key相同的一个常量值  if(user==null){//说明没有登录  //获得当前请求的地址 登录成功后直接跳过去 增加用户体验度  StringBuffer requestURL = request.getRequestURL();  String queryString = request.getQueryString();//获得参数  if(!StringUtils.*isBlank*(queryString)){  requestURL.append("?").append(queryString);  }  //记得在登录成功后 跳转回登录前请求地址的时候清除session的这个值  session.setAttribute("lastRequestUrl",requestURL.toString());  response.sendRedirect("/page/login.html");  return;  }  }  //其他情况就放过  filterChain.doFilter(request,response);  }   }   @Override  public void destroy() {   } } |

1. 整合cookie完成自动登录与记住密码功能

只需要在controller用户输入的账户密码验证通过后判断前台的，复选框是否勾选，如果勾选了，将用户的账户密码写入cookie然后写到前台，这里只写出关键的代码

Response.addCookie(new Cookie(“username”,”password”))

设置cookie的存活时间setMaxAge

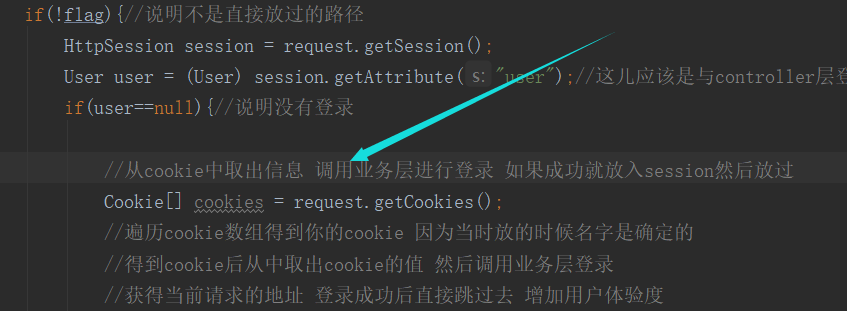
设置cookie的域:domain

设置cookie的path:

 1.domain表示的是cookie所在的域，默认为请求的地址，如网址为www.test.com/test/test.aspx，那么domain默认为www.test.com。而跨域访问，如域A为t1.test.com，域B为t2.test.com，那么在域A生产一个令域A和域B都能访问的cookie就要将该cookie的domain设置为.test.com；如果要在域A生产一个令域A不能访问而域B能访问的cookie就要将该cookie的domain设置为t2.test.com。

  2.path表示cookie所在的目录，asp.net默认为/，就是根目录。在同一个服务器上有目录如下：/test/,/test/cd/,/test/dd/，现设一个cookie1的path为/test/，cookie2的path为/test/cd/，那么test下的所有页面都可以访问到cookie1，而/test/和/test/dd/的子页面不能访问cookie2。这是因为cookie能让其path路径下的页面访问。

然后在filter中加入:



自己完成这个完整的功能 有问题就问，最好自己能完成 不然很**尴尬**

1. 几个项目中常用的工具类

1>Json工具类

|  |
| --- |
| package com.springmvc.util;  import lombok.extern.slf4j.Slf4j;  import org.codehaus.jackson.map.DeserializationConfig;  import org.codehaus.jackson.map.ObjectMapper;  import org.codehaus.jackson.map.SerializationConfig;  import org.codehaus.jackson.map.annotate.JsonSerialize;  import org.codehaus.jackson.map.ser.impl.SimpleFilterProvider;  import org.codehaus.jackson.type.TypeReference;  import java.io.IOException;  /\*\*  \* 转换json  \*/  @Slf4j  public class JsonUtil {  private final static ObjectMapper objectMapper = new ObjectMapper();  static{  //所有字段参与序列化  objectMapper.setSerializationInclusion(JsonSerialize.Inclusion.ALWAYS);  //序列化对象为null时不报错  objectMapper.configure(SerializationConfig.Feature.FAIL\_ON\_EMPTY\_BEANS,false);  //序列化时 防止date类型转换为TIMESTAMPS类型  objectMapper.configure(SerializationConfig.Feature.WRITE\_DATES\_AS\_TIMESTAMPS,false);  //自定义转换格式  objectMapper.setDateFormat(new SimpleDateFormat("yyyy-MM-dd hh:mm:ss"));  //反序列化时 忽略不匹配的字段  objectMapper.configure(DeserializationConfig.Feature.FAIL\_ON\_UNKNOWN\_PROPERTIES,false);  }  // json 转字符串：  public static <T> String object2string(T t){  if (t==null){  return null;  }  try {  return t instanceof String? (String) t :objectMapper.writeValueAsString(t);  } catch (Exception e) {  log.info("JsonUtil: object2string exception,Object:{},exception:{}",t,e);  return null;  }  }  //字符串转对象 TypeReference<List<User>> tTypeReference：  public static <T> T string2object(String str,TypeReference<T> tTypeReference){  if (str==null){  return null;  }  try {  return tTypeReference.getType().equals(String.class)?(T)str:objectMapper.readValue(str,tTypeReference);  } catch (IOException e) {  log.info("JsonUtil: string2object exception,String:{},tTypeReference:{},exception:{}",str,tTypeReference,e);  return null;  }  }  } |

<https://blog.csdn.net/qq_22075041/article/details/77431915>

这里面有常用的 没测试过 应该问题不大