# SSM框架

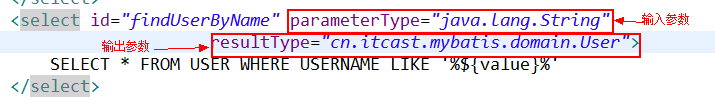
# Mybatis部分

## 概述及特点

Mybatis是一个基于持久层的框架，是对jdbc的简单封装，重点是让我们程序员灵活的去编写sql语句，所以使用mybatis对sql语句的性能优化，比较方便。

Mybaits可以通过xml和注解对statement进行配置，主要的两个特点：

1. 输出映射，将java对象映射到statement中即sql语句的输入参数。
2. 输出映射，将sql语句执行的结果映射成java对象。



## mybatis框架原理.avi

SqlMapConfig.xml(mybatis全局配置文件)

Mapper.xml（映射文件是多个）、Mapper.xml.....

SqlSessionFactory（会话工厂）

创建SqlSession会话

SqlSession（会话）

通过sqlSession接口操作数据库

Executor（底层执行器）

底层操作数据库对象

MappedStatement（mybatis底层封装对象）

封装了sql语句、输入参数类型、输出结果类型等信息

Mysql数据库

输入参数

java基本类型(int/string...)

自定义pojo

hashmap

输入映射

输出结果

java基本类型(int/string...)

自定义pojo

hashmap

输出映射

## 配置文件

SqlMapConfig.xml和\*Mapper.xml(\*.xml)

SqlMapConfig.xml中配置了

## Mapper代理开发：（4个规范）

### namespace对应的是接口的全类名

### select/insert/delete/update等语句中的id对应的是接口中的方法名

### parameterType对应的是方法中的参数类型

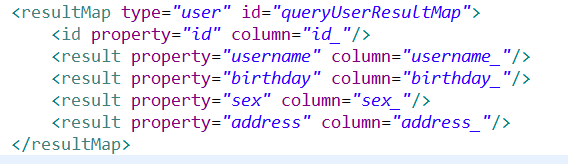
### resultType对应的是方法返回值的类型

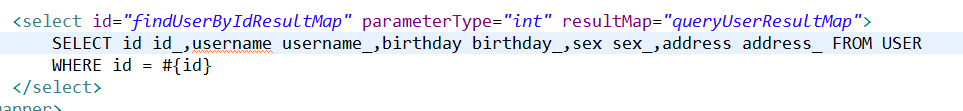
要求：mapper.java文件和mapper.xml同名且在同一个目录

## resultType和resultMap

### resultType 就是将查询到的列封装到指定的pojo（即java对象）中

### resultMap可以将封装嵌套的java对象





### resultMap关联查询

|  |  |  |
| --- | --- | --- |
| **类型** | **使用** | **举例** |
| **一对一** | < association>标签 | 在resultMap中使用<association property="user" javaType="cn.itcast.mybatis.po.User"> 吧user属性映射到一个User对象上. |
| **一对多** | <collection>标签 | **ResultMap类型可以继承自之前的一对一的resultMap这样就有了订单和用户的映射信息.然后使用collection指定把orderdetails属性映射成一个orderDetail的集合** |
| **多对多** | 联合使用2种标签 | 将查询出的column封装到一个复杂对象中，属性为list就使用<collection>标签，属性为pojo，就使用<association>标签 |

### resultMap延时加载

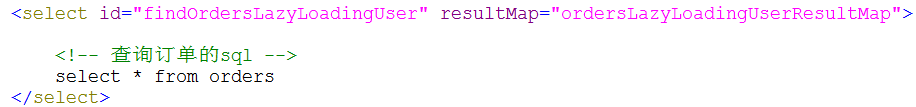
第一步：打开延迟加载的开关 ------SqlMapConfig.xml

|  |
| --- |
| <settings>  <!-- 延迟加载的总开关 -->  <setting name="lazyLoadingEnabled" value="true"/>  <!-- 设置为false实现按需求加载 -->  <setting name="aggressiveLazyLoading" value="false"/>  </settings> |

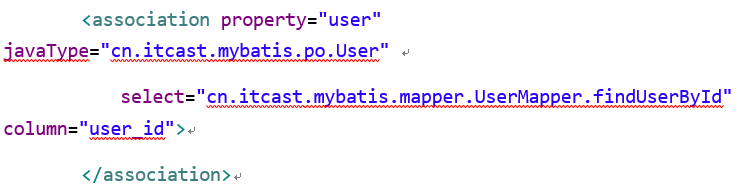
第二步：在Mapper.xml中配置resultMap



对应的statement为：



要延时加载的对象进行如下配置



**要点总结：**

**开启开关，在order对象的user属性使用select指向另一个statement然后使用column给这个statement设置参数。**

### 打开mybatis二级缓存

第一步：全局配配文件------SqlMapConfig.xml

|  |
| --- |
| <setting name="cacheEnabled" value="true"/> |

第二步：Mapper中开启二级缓存

|  |
| --- |
| <cache /> |

第三步：对象序列化

|  |
| --- |
| **public class User implements Serializable {** |

PS:

## paramType

### #{} 和 ${} 区别

${}是拼接符,不能防止sql注入，输入参数为基本类型时{}中间只能写value,对象时是对象的属性名.乱写会报错

#{}是占位符,能防止sql注入,基本类型时括号中间的名子随便写,对象是为对象属性,为对象时乱写会报错

尽量使用 #{}

## 动态sql

Sql拼接及条件查询，<sql><where><if><foreach>等标签的使用



# Mybatis配置

## 加载SqlMapConfig.xml

### 编码实现---为了获得sqlSessionFactory对象

SqlSessionFactoryBuilder().build(inputStream)



## SqlMapConfig.xml配置（单独运行）

**创建mybatis的核心配置文件(数据源、事务管理、Mapper)**

使用mybatis-3-config.dtd约束xml

|  |
| --- |
| <?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>  <!-- 和spring整合后 environments配置将废除-->  <environments default="development">  <environment id="development">  <!-- 使用jdbc事务管理-->  <transactionManager type="JDBC" />  <!-- 数据库连接池-->  <dataSource type="POOLED">  <property name="driver" value="com.mysql.jdbc.Driver" />  <property name="url"  value="jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8" />  <property name="username" value="root" />  <property name="password" value="mysql" />  </dataSource>  </environment>  </environments>  <!-- 加载mapper.xml -->  <!-- <mappers></mappers> -->  </configuration> |

可以有多个environment节点表示多个不同的数据库，如果在创建SqlSessionFactory的时候不指定要使用的environment将使用 default属性指定的environment做为默认。

## 加载mapper.xml

SqlMapConfig.xml中配置,整合spring时，不使用该方法，使用

|  |
| --- |
| <mappers>  <mapper resource="sqlmap/User.xml"/>  <!-- <mapper class="cn.itcast.mybatis.mapper.UserMapper"/> -->  <package name="cn.itcast.mybatis.mapper"/>  </mappers> |

## spring创建mapper代理对象（spring管理）

<!-- 使用mapper扫描器创建mapper代理对象

扫描器把自动将包下边的mapper扫描出来创建代理对象在spring容器注册，bean的id为类名（首字母小写）

-->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<!-- 指定要扫描的包路径，如果要扫描多个包，中间使用半角逗号分隔

注意：如果使用扫描器，不需要在sqlMapConfig.xml中加载mapper，要将mapper.xml和mapper.java放在同一个目录且同名

-->

<property name=*"basePackage"* value=*"cn.itcast.ssm.mapper"*/>

<property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactory"*></property>

</bean>

## 配置sqlSessionFactory

<!-- 配置sqlSessionFactory -->

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!-- 配置数据源 -->

<property name=*"dataSource"* ref=*"dataSource"*/>

<!-- 加载mybatis的配置文件 -->

<property name=*"configLocation"* value=*"classpath:mybatis/SqlMapConfig.xml"*/>

</bean>

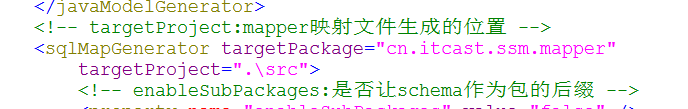
## 逆向工程生成Mapper.java、Mapper.xml

使用mybatis文件夹下的逆向工程生成即可

修改数据库信息



修改生成代码的指定包路径



# Springmvc部分

## 架构图



## 架构流程

1. 用户发送请求至前端控制器DispatcherServlet
2. DispatcherServlet收到请求调用HandlerMapping处理器映射器查找Handler。
3. 处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。
4. DispatcherServlet通过HandlerAdapter处理器适配器调用处理器
5. HandlerAdapter调用处理器Handler
6. Handler执行完成返回ModelAndView
7. HandlerAdapter将Handler执行结果ModelAndView返回给DispatcherServlet
8. DispatcherServlet将ModelAndView传给ViewReslover视图解析器，ViewReslover根据逻辑视图名解析View
9. ViewReslover返回View
10. DispatcherServlet对View进行渲染视图（即将模型数据填充至request域）。
11. DispatcherServlet响应用户

## 理解

HandleMapping作用：根据映射找到handler

HandlerAdapter作用：根据controller（也就是action）实现的接口，去执行handler

# Springmvc配置

## Springmvc头信息

<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"

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-3.2.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.2.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-3.2.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-3.2.xsd ">

</beans>

## 在springmvc.xml中配置注解处理器映射器和适配器

|  |
| --- |
| <!—下边的注解处理器映射器和适配器配置可以使用下边mvc:annotation-driven代替 -->  <mvc:annotation-driven></mvc:annotation-driven> |

等价于

|  |
| --- |
| <!-- 注解处理器映射器 3.1版本以后 -->  <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>  <!-- 注解处理器适配器 -->  <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"/> |

## 注解部分

|  |  |  |
| --- | --- | --- |
| 类型 | 位置 | 作用 |
| @Controller | 类上 | 指定类为Handler |
| @RequestMapping("/itemsList3") | 类上、方法上 | 指定url访问路径 |
| @requestParam(“ids”)Integer id | 方法的形参上 | 用来匹配请求参数名 |
|  |  |  |

## 配置扫描指定包下的handler

|  |
| --- |
| <!-- 使用组件扫描，将@controller注解的类扫描出来在容器中进行注册  组件扫描可以扫描标记有@controller、@service、@repository、@component的bean进行扫描 -->  <context:component-scan basepackage="cn.itcast.springmvc.controller"/> |

## 参数绑定问题

springmvc接收请求的key/value串（比如：id=2&type=101），经过类型转换，将转换后的值赋值给controller方法的形参，这个过程就叫参数绑定。

|  |  |
| --- | --- |
| 参数绑定类型 | 实现 |
| 默认支持类型 | HttpServletRequest、HttpServletResponse、HttpSession、Model/ModelMap |
| 基本数据类型 | **基本类型的参数绑定只要url中参数名和方法的形参一致即可绑定成功，如果名子不一致可以使用注解@RequestParam()**  **的value参数指定指定的参数名如:@RequestParam(value=“ids”，required=”true”,defaultValue=””) id 。** |
| 简单pojo对象 | **当页面参数名称和controller方法的pojo的参数的属性名一致，可以将请求的参数绑定到pojo形参的属性中。** |
| 包装pojo对象 | **因为包装POJO中的属性又是一个POJO，所以我们要给对象的属性的属性赋值。只需要写pojo属性名.属性名即可。** |
| 数组 | **Handler方法的形参名子和表单中的多个元素名子一致可以绑定。** |
| 集合 | **页面表单元素的名子写成.itemsList[n].name，保持 一致即可。**  <td><input type="text" name="itemsList[${status.index}].name" value="${item.name }"/></td> |

## 配置自定义参数绑定器

第一步：编写参数绑定器

|  |
| --- |
| //Converter<String, Date>String就是原始类型即key/value串，Date目标转换类型  **public** **class** CustomDateConverter **implements** Converter<String, Date>{  @Override  **public** Date convert(String source) {  //将字符串转成日期类型  SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  **try** {  **return** simpleDateFormat.parse(source);  } **catch** (ParseException e) {  e.printStackTrace();  //如果 转换失败返回null  **return** **null**;  }  }  } |

第二步：配置绑定器

|  |
| --- |
| <!-- 上边的注解处理器映射器和适配器配置可以使用下边mvc:annotation-driven代替 -->  <mvc:annotation-driven conversion-service="conversionService"></mvc:annotation-driven>  <!-- 自定义转换器 -->  <bean id="conversionService"  class="org.springframework.format.support.FormattingConversionServiceFactoryBean">  <!-- 转换器 -->  <property name="converters">  <list>  <bean class="cn.itcast.ssm.controller.converter.CustomDateConverter"/>  </list>  </property>  </bean> |

# Spring部分

## Spring是什么？

Spring是一个基于JAVA的轻量级J2EE应用框架

Spring是一个超级的“黏合平台”，将很多技术黏合在一起，形成一个整体，使每个组件发挥其最大功效。

(IOC、AOP、Template模板、s2sh整合)

## 具体功能

表现层：SpringMVC

逻辑层：Bean管理、AOP、事务管理

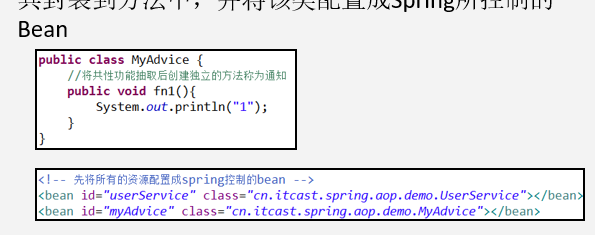
数据层：Spring JDBCTemplate

简答说：

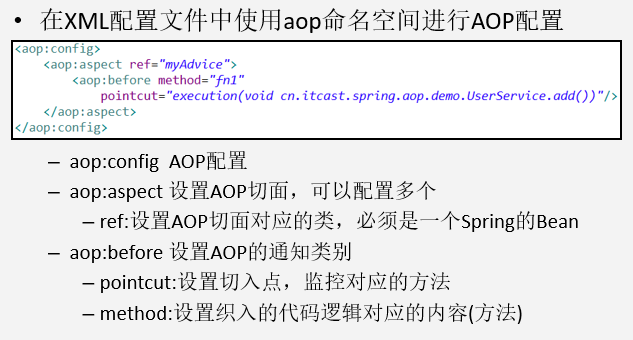
IOC: 为了让spring为我们创建类的对象

AOP:为了抽取代码

## aop







# Spring配置

## Spring头文件(applicationContext.xml)

<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"*

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-3.2.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.2.xsd*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop-3.2.xsd*

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx-3.2.xsd "*>

*</beans>*

## 配置数据库连接池

<!-- 配置数据源dataSource -->

<!-- 加载配置文件 -->



<context:property-placeholder location=*"classpath:db.properties"*/>

<!-- 数据库连接池 -->

<bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"* destroy-method=*"close"*>

<property name=*"driverClassName"* value=*"${jdbc.driver}"*/>

<property name=*"url"* value=*"${jdbc.url}"*/>

<property name=*"username"* value=*"${jdbc.username}"*/>

<property name=*"password"* value=*"${jdbc.password}"*/>

<property name=*"maxActive"* value=*"10"*/>

<property name=*"maxIdle"* value=*"5"*/>

</bean>