## MyBatis

### MyBatis原始使用（mybatis-1）

（1）准备：映射文件





（2）通过sqlsession使用

|  |
| --- |
| *//1.读取配置文件* InputStream in = Resources.getResourceAsStream(**"SqlMapConfig.xml"**); *//2.创建SqlSessionFactory工厂* SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(in); *//3.使用工厂生产SqlSession对象* SqlSession session = sqlSessionFactory.openSession(); *//4.执行Sql语句* User user = session.selectOne(**"test.findUserById"**, 10); *//5. 打印结果* System.out.println(user); *//6.释放资源* session.close(); in.close(); |

### 基于原始DAO（mybatis-2）

通过引入dao层将SqlSession 以及相应的SQL操作进行封装而已。

1. 准备：mapper文件如上

Dao层封装如下：

|  |
| --- |
| **private** SqlSessionFactory **sqlSessionFactory**; *// 通过构造方法注入* **public** UserDaoImpl(SqlSessionFactory sqlSessionFactory){  **this**.**sqlSessionFactory** = sqlSessionFactory; } **public** User findUserById(Integer id) {  *//sqlSession是线程不安全的，所以它的最佳使用范围在方法体内* SqlSession sqlSession = **sqlSessionFactory**.openSession();  User user = sqlSession.selectOne(**"test.findUserById"**,id);  sqlSession.close();  **return** user; } |

1. 使用

|  |
| --- |
| **private** SqlSessionFactory **factory**; **private** UserDao **userdao**; **private** InputStream **in**; *// 作用：在测试方法前执行这个方法* @Before **public void** setUp() **throws** Exception{  *//1.读取配置文件* **in** = Resources.*getResourceAsStream*(**"SqlMapConfig.xml"**);  *//2.创建SqlSessionFactory工厂* **factory** = **new** SqlSessionFactoryBuilder().build(**in**); }  **public void** testFindUserById() **throws** Exception{  *// 将初始化好的工厂注入到实现类中* UserDao userdao = **new** UserDaoImpl(**factory**);  User user = userdao.findUserById(34);  System.***out***.println(user);  *//User [id=34, username=小小, sex=1, birthday=Mon Jul 15 00:00:00 CST 2019, address=西安市]* } |

### 基于代理DAO（mybatis-3）

原始DAO需要定义DAOImpl类，并且在这个类中有大量的sqlsession重复代码，通过代理DAO可以解决此问题。

1. 准备：

A：mapper文件user.xml

B：User.xml文件中的mapper文件必须是全路径名：

|  |
| --- |
| **<mapper namespace="com.hust.lw.dao.UserDao">** |

C：去掉UserDaoImp.java文件

1. 使用

只需要有user.xml这个mapper文件，以及DAO接口文件，其中DAO接口文件中的接口与Mapper文件中的statement id是通过mapper文件中的namespace的全路径名关联的，然后通过sqlsession来使用。

|  |
| --- |
| **private** SqlSessionFactory **factory**; **private** UserDao **userdao**; **private** InputStream **in**; **private** SqlSession **session**;  *// 作用：在测试方法前执行这个方法* @Before **public void** setUp() **throws** Exception {  *//1.读取配置文件* **in** = Resources.*getResourceAsStream*(**"SqlMapConfig.xml"**);  *//2.创建SqlSessionFactory工厂* SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();  *//3.创建SqlSession工厂对象* SqlSessionFactory factory = builder.build(**in**);  *//4.使用工厂生产SqlSession对象* **session** = factory.openSession();  *//5.创建Dao接口的代理对象* **userdao** = **session**.getMapper(UserDao.**class**); }  @Test **public void** testFindUserById() {  *//5.使用代理对象执行方法* User user = **userdao**.findUserById(1);  System.***out***.println(user); } |

Mybatis的基本使用，以及来龙去脉通过这三个例子得到了一个整体的理解，随后就是一些复杂sql的mapper文件编写，以及在springboot中使用mybatis。

<https://www.cnblogs.com/benjieqiang/p/11183580.html>

## SpringMVC

### 1、环境搭建及controller与jsp互传数据（SpringMVC-1）

1. 新建SpringMVC工程



1. 部署Tomcat环境





1. 缺少Spring MVC相关类库引用





完成如上的3步后，则可以将这个空的SpringMVC工程跑起来了。

参考：

IDEA建立Spring MVC Hello World 详细入门教程

<https://www.cnblogs.com/wormday/p/8435617.html>

1. 创建controller类，找不到servlet

出现了问题： javax.servlet 包找不到

解决： 将本地 Tomcat 服务器的目录下【lib】文件夹下的 servlet-api.jar 包拷贝到工程【lib】文件夹下，添加依赖

这里面的“添加依赖”则出现了误会

参考：

cannot resolve symbol 'Servlet'

<https://blog.csdn.net/ycfn97/article/details/101114915>

1. 其他步骤，参考：

Spring MVC【入门】就这一篇！

<https://www.jianshu.com/p/91a2d0a1e45a>

说明：

A、 指定controller的方法，要么特定指定，要么泛指

|  |
| --- |
| *<bean id="simpleUrlHandlerMapping"  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">  <property name="mappings">  <props>   <!-- /hello 路径的请求交给 id 为 helloController 的控制器处理-->    <prop key="/hello">helloController</prop>  </props>  </property> </bean> <bean id="helloController" class="com.hust.lw.controller.HelloController"></bean>* |

|  |
| --- |
| *<!-- 扫描ccom.hust.lw.controller下的组件 -->* <**context:component-scan base-package="com.hust.lw.controller"**/> |

如上只能选一个

B、在用模型传参时，模型中的set方法实现中，必须this.userName和this.password这样的格式

C、 jsp向后台controller传参一共有如下几种方法：

jnhs-SpringMVC jsp页面向controller传递参数的五种方式

<https://www.cnblogs.com/jnhs/p/9961589.html>

D、 Controller向前端jsp回传数据的方法：原生API完成、使用modelandview

、使用model、使用modelAttribute

# [知识整理一：idea搭建maven 的springmvc项目（将maven项目转变为web项目）](https://www.cnblogs.com/dayu007/p/11170999.html)

### 史上最全最强SpringMVC详细示例（SpringMVC-2）

[https://www.cnblogs.com/sunniest/p/4555801.html#commentform](https://www.cnblogs.com/sunniest/p/4555801.html" \l "commentform)

1. JSP到controller格式转换

由于前台传到controller里的值是String类型的，当往Model里Set这个值的时候，如果set的这个属性是个对象，Spring就会去找到对应的editor进行转换，然后再SET进去。

有两种：只在当前controller中生效的；在全局controller中生效的。

|  |
| --- |
| */\*\*  \* 使用InitBinder来处理Date类型的参数  \* the parameter was coverted in initBinder  \** ***@url:*** *http://localhost:8080/spring-mvc-action-tutorial/mvc/date?date=2016-05-17  \* http://localhost:8080/spring-mvc-action-tutorial/mvc/date?date=2016-05  \* http://localhost:8080/spring-mvc-action-tutorial/mvc/date?date=2016-05-17 15:00:00  \*   \** ***@param date*** *\** ***@return*** *\*/* @RequestMapping(**"/date"**) **public** String date(Date date) {  System.***out***.println(date);  **return "hello"**; }  */\*\*  \* At the time of initialization,* ***convert the type "String" to type "date"***  *\*   \** ***@param binder*** *\*/* @InitBinder **public void** initBinder(ServletRequestDataBinder binder) {  System.***out***.println(**"InitBinder of date : convert the type of String to type date."**);  binder.registerCustomEditor(Date.**class**, **new** CustomDateEditor(**new** SimpleDateFormat(**"yyyy-MM-dd"**), **true**)); } |

参考：

SpringMVC之自定义类型转换WebBindingInitializer@initBinder

<https://blog.csdn.net/zmx729618/article/details/54133830>

spring中@InitBinder和WebBindingInitializer的使用

<https://blog.csdn.net/yu870646595/article/details/51191188/>

1. 前端ajax调用到后台controller，然后返回数据到ajax

通过PrintWriter返回

|  |
| --- |
| */\*\*  \* 使用Ajax调用: pass the parameters to front-en using ajax  \* url: http://localhost:8080/spring-mvc-action-tutorial/mvc/page/ajax  \*   \** ***@param name*** *\** ***@param pw*** *\*/* @RequestMapping(value = **"/getPerson"**, method = RequestMethod.***POST***) **public void** getPerson(String name, PrintWriter pw) {  *// 也可以使用HttpServletResponse.getWriter()来获取PrintWriter* pw.write(**"hello, "** + name); } |

参考：

前后台数据交换，printwriter、jsonobject、jsonarray、ajax请求，数据交换

<https://www.hellojava.com/a/69745.html>

1. JSON在前后台之间传递

“客户端请求的数据是json字符串,springmvc不能直接解析，加@requestBody解析成对象，客户端请求的是key/value的拼接不需要加@requestBody，@responseBody返回json字符串，如果是ajax请求，我们想到得到json对象的话(为了方便根据key取value)”



对于request是json串，则必须要有@requestbody，如下图所示：



对于表单数据，则可以不需要@RequestBody：



Form-data、x-www-form-unlencoded、raw/json三者区别：







参考：

@RequestBody和@ResponseBody的用法以及Stringify()的作用

<https://blog.csdn.net/linlinlinfeng/article/details/82705772>

springmvc实现json交互 -requestBody和responseBody

<https://www.cnblogs.com/cnblog-long/p/6547380.html>

浅谈@RequestMapping @ResponseBody 和 @RequestBody 注解的用法与区别

<https://blog.csdn.net/ff906317011/article/details/78552426>

关于postman，参考如下两篇：

postman Params和Body的区别

<https://blog.csdn.net/Loren_SUN/article/details/86063374>

Postman使用详解

<https://www.cnblogs.com/xiaoxi-3-/p/7839278.html>

1. spring异常统一处理

Spring MVC异常统一处理的三种方式

<https://www.cnblogs.com/junzi2099/p/7840294.html>

（5）