Intellij IDEA入门到精通

(高级篇)

黄文毅 编著

1.1 课程介绍

1.1.1 课程介绍

**开发环境：**

window10 + Intellij IDEA 2018 + JDK1.8

彻底掌握Intellij IDEA（实战篇）： https://edu.csdn.net/course/detail/7751

**适用人群：**

1.编程人员

2.初级工程师、中级工程师

3.在线学生，研究生

4.Intellij IDEA爱好者

**联系方式：**

QQ：[1512079179@qq.com](mailto:1512079179@qq.com)

微信：huangwenyi2010

邮箱地址：huangwenyi10@163.com

**重要式：**

1）工欲善其事必先利其器，磨刀不误砍柴工

2）提高工作效率

3）减少错误

**课程资料：**

购买正版课程后可联系老师获取课程资料。

**课程服务：**

购买正版视频，提供购买订单号，可以享受以下服务：

1）学员老师一对一交流服务

2）课后作业和思考题服务

3）咨询服务（职业规划、经验交流）

4）购买【盗版视频】无法获得所有的源代码、课件及服务。

**课程目标：**

帮助学员快速掌握Intellij IDEA

**课程目录：**

1.1 课程介绍

1.2 块选择

1.3 隐藏.idea文件

1.4 代码自动补全live templates

1.5.1 自定义代码补全live templates

1.5.2 Postfix Completion 的使用

1.6 查看完整类继承关系图

1.7 Intellij IDEA启动缓慢优化

1.8 解决控制台输出乱码问题

1.9 Inject language快速编写Json字符串

1.10 深入剖析IntelliJ IDEA的目录结构

2.1.1 搭建web项目

2.1.2 web项目集成Spring框架

2.1.3 web项目集成Spring MVC框架

2.1.4 web项目集成MyBatis框架

2.1.5 web项目集成Log4J框架

2.1.6 web项目集成JUnit框架

3.1 IntelliJ IDEA快速搭建Spring Boot

3.2 Maven Helper插件分析依赖

3.3 GsonFormat插件转化json

3.4 Markdown文本编辑器插件

3.5 FindBugs插件快速排查bug

3.6 ECTranslation插件翻译英文

3.7 CheckStyle-IDEA插件检查代码风格

3.8 MyBatisCodeHelper插件生成代码

3.9 Actibpm插件绘制流程图

3.10 Intellij IDEA使用SSH连接服务器

3.11 Rest Client工具使用

3.12 背景图片和代码小视图插件

3.13 GrepConsole自定义控制台颜色

3.14 大小写转化和自动生成序列版本号插件

3.15 IdeaVim文本编辑器插件

3.16 JRebel插件实现热部署

3.17 SonarLint代码质量管理插件

4.1 IntelliJ IDEA构建多模块项目

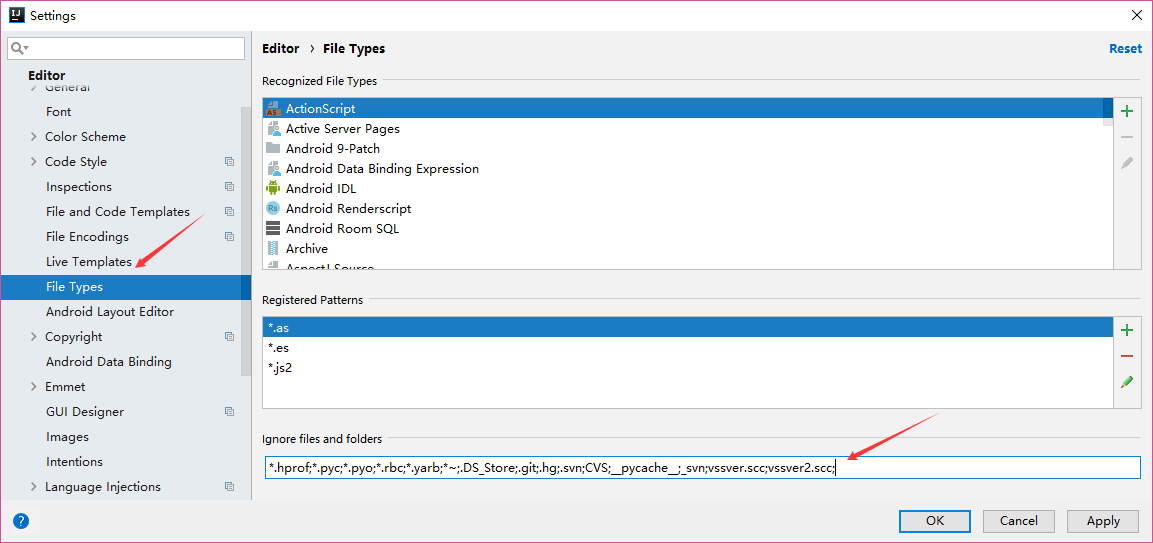
4.2 作业

1.2 块选择

快捷键：Alt + 选择（在Intellij IDEA 不分文件类型）

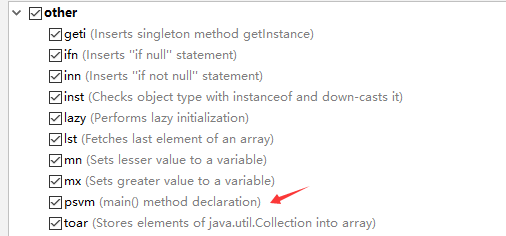
1. 删除作用；
2. 全部替换作用；

1.3 隐藏.idea文件

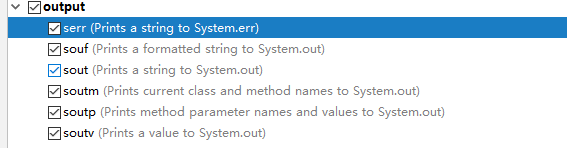


1.4 代码自动补全live templates(上)

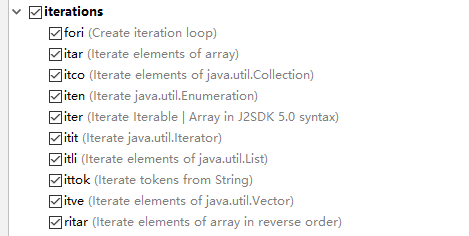
1. psvm



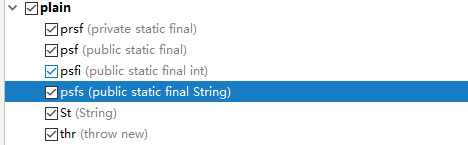
1. sout系列



1. fori系列



4、psfs系列

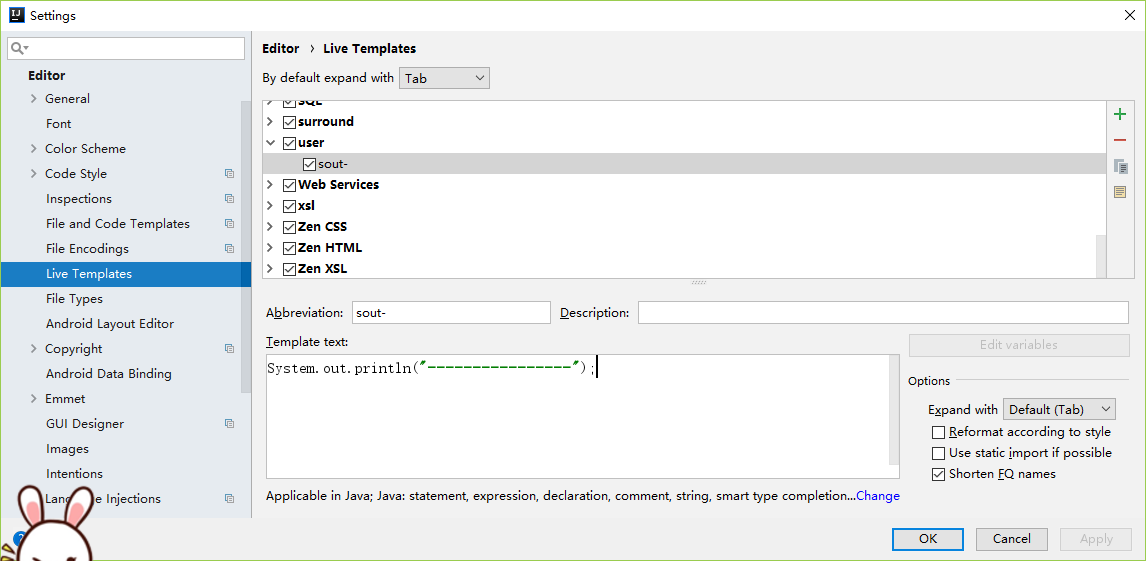


作业：其他没讲述到的快捷键，希望大家自己动手实践。

1.5 自定义代码补全live templates

**1、不带变量代码补全**

新增自定义模板，首先需要填写触发单词（即 Abbreviation），描述是可选的，然后定义模板的上下文，点击define选择Java，这样在编辑 Java 的时候就会触发当前模板，定义完上下文之后，就可以填写模板了。



**2、带变量的自动代码补全功能**

实例一：

System.out.println("-------------");

@Test

public void test() {

}

实例二：

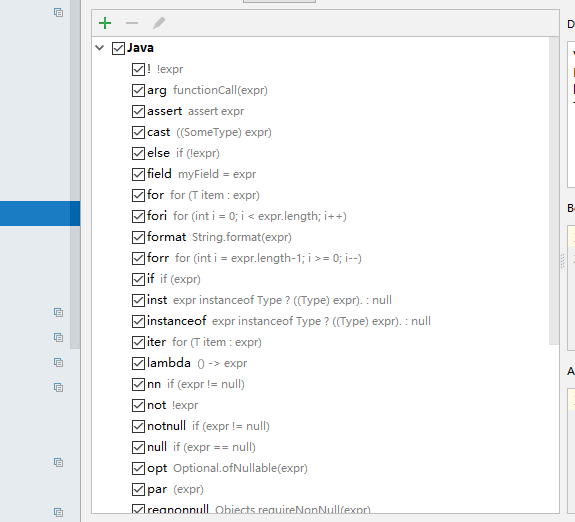
private static final Logger LOGGER = LoggerFactory.getLogger($CLASS$.class);

作业：groovyScript()使用，动手实现自己想要的代码模板。

QQ：[1512079179@qq.com](mailto:1512079179@qq.com)

1.5.2 Postfix Completion 的使用

Postfix Completion 功能本质上也是代码模板，只是它比 Live Templates 来得更加便捷一点点而已。



1.6 查看完整类继承关系图

1. 查看类继承关系图

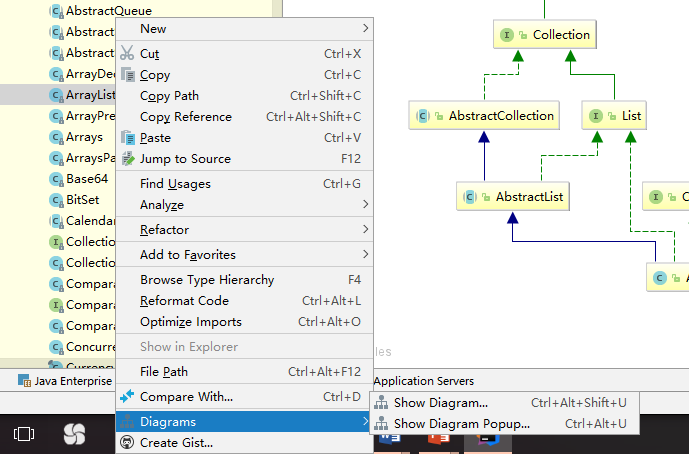
1）删除不需要的类；

2）显示类上的方法；

3）对显示方法再进行过滤；

4）图片太小，如何放大；（按住Alt）

2、查看类所在的模块



作业：动手学习显示图表的其他按钮操作。

1.7 Intellij IDEA启动缓慢优化

1、修改idea64.exe.vmoptions 配置，修改前请记得备份（重要）



1. 去掉不需要使用的插件

作业：动手查找资料，搜索让Intellij IDEA启动加快的其它方法。

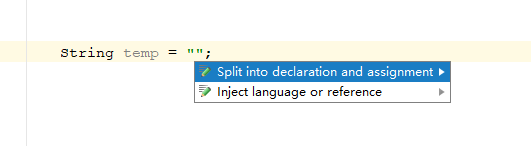
1.8 解决控制台输出乱码问题

1、修改idea64.exe.vmoptions 配置，追加配置： -Dfile.encoding=UTF-8

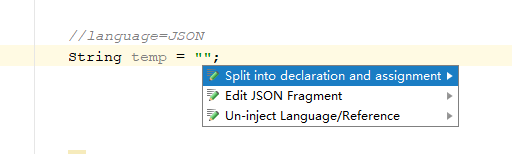
2、修改虚拟机参数“ VM options ”，内容与在文件idea64.exe.vmoptions中追加的内容相同，皆为-Dfile.encoding=UTF-8

1.9 Inject language快速编写Json字符串

1. ALT + Enter

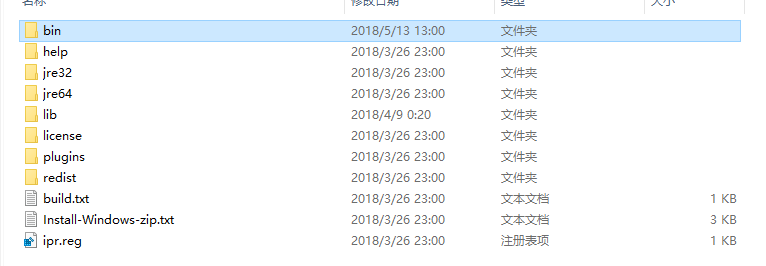


1. 再次点击ALT + Enter



1.10 深入剖析IntelliJ IDEA的目录结构

1、目录结构



bin 是 IDEA 的可执行代码目录。

help 是 IDEA 的帮助文件目录。

jre64 是 IDEA 自带的 JRE 环境，故 IDEA 可在未安装 JDK 的计算机上进行 PHP、Python等语言的编码。

lib 是 IDEA 依赖的库文件目录，里面有很多的 Jar 文件。

license 是 IDEA 的许可证文件目录。

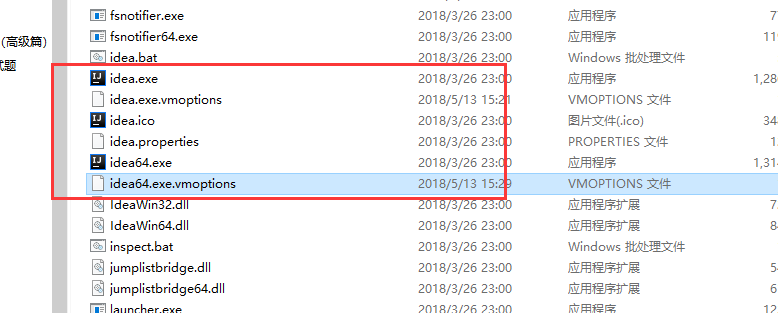
plugins 是 IDEA 的插件目录。

redist 是 IDEA 中索引机制所依赖的 redist 库目录。

两个 .txt 说明文件和

一个注册表项文件。

2、bin目录



idea.exe 文件是 IntelliJ IDEA 32位的可执行文件。

idea.exe.vmoptions 文件是 IntelliJ IDEA 32位的可执行文件的 JVM 配置文件。

idea.properties 文件是 IntelliJ IDEA 的一切全局属性的配置文件。

idea64.exe 文件是 IntelliJ IDEA 64位的可执行文件。

idea64.exe.vmoptions 文件是 IntelliJ IDEA 64位的可执行文件的 JVM 配置文件。

2.1 IntelliJ IDEA搭建SSM框架

## 2.1.1 搭建web项目

archetypeCatalog，value：internal。具体如图2-4所示。

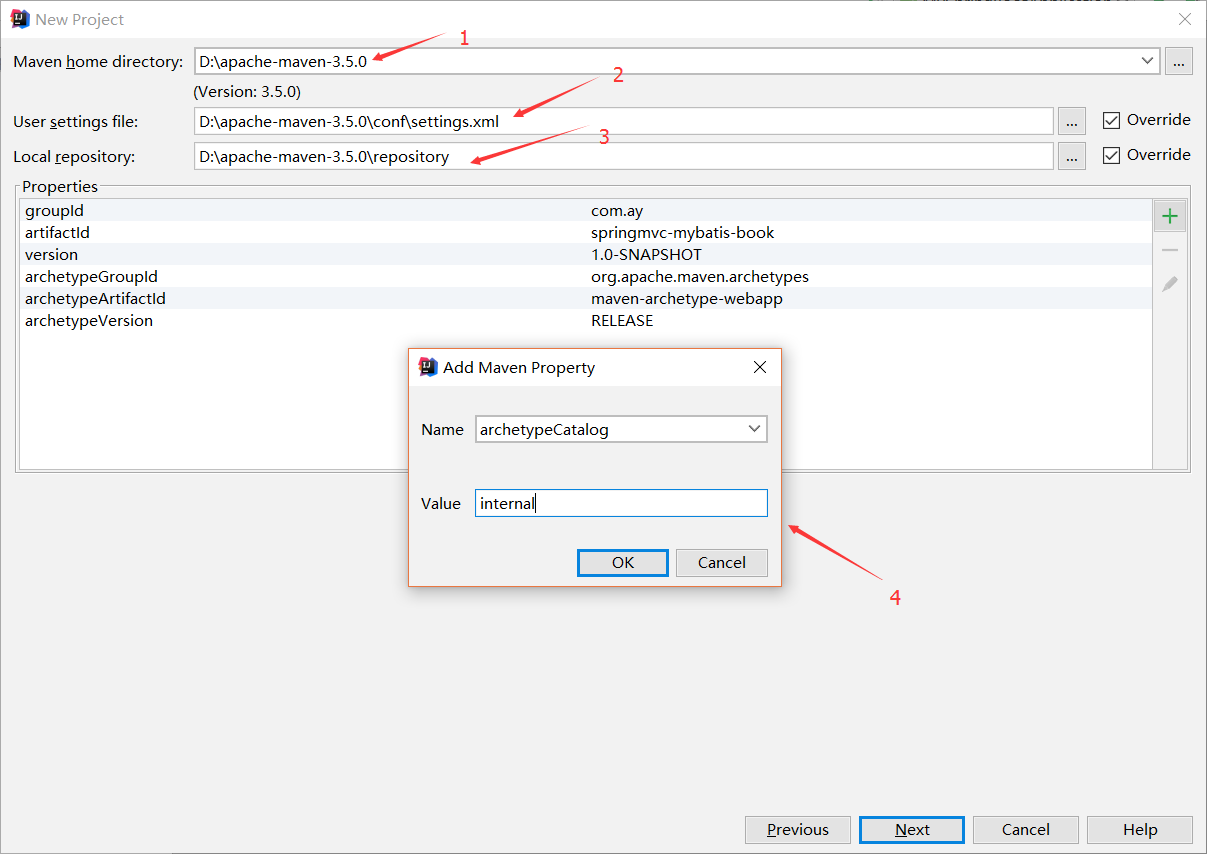


图2-4 填写Maven相关信息窗口

注意：Intellij IDEA根据maven archetype的本质，执行mvn archetype:generate命令。该命令执行时，需要指定一个archetype-catalog.xml文件。该命令的参数-DarchetypeCatalog，可选值为：remote，internal，local等，用来指定archetype-catalog.xml文件从哪里获取，默认为remote，即从 http://repo1.maven.org/maven2/archetype-catalog.xml路径下载archetype-catalog.xml文件。archetype-catalog.xml 文件约为3-4M，下载速度很慢，导致创建过程卡住。解决的办法很简单， 指定-DarchetypeCatalog为internal，即可使用maven默认的archetype-catalog.xml，而不用从remote下载。

## 2.1.2 web项目集成Spring框架

1. 添加依赖

<properties>

<spring.version>5.0.4.RELEASE</spring.version>

</properties>

<dependencies>

<!--spring start -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</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-context-support</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-aspects</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-expression</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-test</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<!--spring end -->

<!-- junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

</dependencies>

其次，在/src/main/resources目录下创建applicationContext.xml配置文件，具体代码如下：

<?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"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

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

http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<context:component-scan base-package="com.ay"/>

</beans>

接着，我们在web.xml配置文件中添加如下的代码：

<!DOCTYPE web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd" >

<web-app>

<display-name>Archetype Created Web Application</display-name>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

</web-app>

最后，在src/main/test/com.ay.test目录下创建SpringTest测试类，具体代码如下：

import org.junit.Test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.stereotype.Service;

/\*\*

\* @author Ay

\* @date 2018/04/02

\*/

@Service

public class SpringTest {

@Test

public void testSpring(){

//获取运用上下文

ApplicationContext applicationContext =

new ClassPathXmlApplicationContext("applicationContext.xml");

//获取SpringTest类

SpringTest springTest = (SpringTest) applicationContext.getBean("springTest");

//调用sayHello方法

springTest.sayHello();

}

public void sayHello(){

System.out.println("hello ay");

}

}

## 2.1.3 web项目集成Spring MVC框架

首先，我们把集成Spring MVC所需要的Maven依赖包和相关的属性值添加到pom.xml文件中，具体代码如下：

<properties>

<spring.version>5.0.4.RELEASE</spring.version>

<javax.servlet.version>4.0.0</javax.servlet.version>

<jstl.version>1.2</jstl.version>

</properties>

<!--springmvc start -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>${jstl.version}</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>${javax.servlet}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<!--springmvc end -->

其次，在web.xml配置文件中添加DispatcherServlet配置，具体代码如下：

<!--配置DispatcherServlet -->

<servlet>

<servlet-name>spring-dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 配置SpringMVC需要加载的配置文件 spring-mvc.xml -->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring-mvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring-dispatcher</servlet-name>

<!-- 默认匹配所有的请求 -->

<url-pattern>/</url-pattern>

</servlet-mapping>

接着，我们在/src/main/resources目录下创建配置文件spring-mvc.xml，具体代码如下：

<?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:mvc="http://www.springframework.org/schema/mvc"

xmlns:aop="http://www.springframework.org/schema/aop"

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/mvc

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

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

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

<!-- 扫描controller(后端控制器),并且扫描其中的注解-->

<context:component-scan base-package="com.ay.controller"/>

<!--设置配置方案 -->

<mvc:annotation-driven/>

<!--配置JSP　显示ViewResolver(视图解析器)-->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="viewClass"

value="org.springframework.web.servlet.view.JstlView"/>

<property name="prefix" value="/WEB-INF/views/"/>

<property name="suffix" value=".jsp"/>

</bean>

</beans>

最后，我们在/src/main/java目录下创建包com.ay.controller，并创建控制层类AyTestController，具体代码如下：

package com.ay.controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

/\*\*

\*@author Ay

\* @date 2018/04/02

\*/

@Controller

@RequestMapping("/test")

public class AyTestController {

@GetMapping("/sayHello")

public String sayHello(){

return "hello";

}

}

我们在/src/main/webapp/WEB-INF目录下创建views文件夹，在views文件下创建hello.jsp文件，具体代码如下：

<%@page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<!DOCTYPE HTML>

<html>

<head>

<title>Getting Started: Serving Web Content</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

</head>

<body>

hello, ay

</body>

</html>

至此，Web项目集成Spring MVC大功告成。我们把Web项目部署到Tomcat服务器上，成功启动Tomcat服务器后，在浏览器输入访问路径：<http://localhost:8080/test/sayHello。当出现如图2-8>

## 2.1.4 web项目集成MyBatis框架

首先，我们把集成MyBatis框架所需要的依赖包添加到pom.xml文件中，具体代码如下：

<properties>

<spring.version>5.0.4.RELEASE</spring.version>

<javax.servlet.version>4.0.0</javax.servlet.version>

<jstl.version>1.2</jstl.version>

<mybatis.version>3.4.6</mybatis.version>

<mysql.connector.java.version>8.0.9-rc</mysql.connector.java.version>

<druid.version>1.1.9</druid.version>

<mybatis.spring.version>1.3.2</mybatis.spring.version>

</properties>

<!--mybatis start -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.connector.java.version}</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>${druid.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>${mybatis.version}</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>${mybatis.spring.version}</version>

</dependency>

<!--mybatis end -->

其次，在/src/main/resources目录下创建jdbc.properties配置文件，具体代码如下：

//驱动

jdbc.driverClassName=com.mysql.jdbc.Driver

//mysql连接信息

jdbc.url=jdbc:mysql://127.0.0.1:3306/springmvc-mybatis-book?serverTimezone=GMT

//用户名

jdbc.username=root

//密码

jdbc.password=123456

最后，我们在applicationContext.xml配置文件添加如下的配置，具体代码如下：

<!--1、配置数据库相关参数-->

<context:property-placeholder

location="classpath:jdbc.properties" ignore-unresolvable="true"/>

<!--2.数据源 druid -->

<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"

init-method="init" destroy-method="close">

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

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

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

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

</bean>

<!--3、配置SqlSessionFactory对象-->

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

<!--注入数据库连接池-->

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

<!--扫描sql配置文件:mapper需要的xml文件-->

<property name="mapperLocations" value="classpath:mapper/\*.xml"/>

</bean>

<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">

<constructor-arg index="0" ref="sqlSessionFactory" />

</bean>

<!-- 扫描basePackage下所有以@MyBatisDao注解的接口 -->

<bean id="mapperScannerConfigurer"

class="org.mybatis.spring.mapper.MapperScannerConfigurer">

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

<property name="basePackage" value="com.ay.dao"/>

</bean>

我们在/src/main/java/com.ay.model目录下创建数据库表对应的实体类对象AyUser，具体的代码如下：

/\*\*

\* 用户实体

\* @author Ay

\* @date 2018/04/02

\*/

public class AyUser implements Serializable{

private Integer id;

private String name;

private String password;

//省略set、get方法

}

实体类对象AyUser创建完成之后，我们在/src/main/java/ com.ay.dao目录下创建对应的DAO对象AyUserDao，AyUserDao是一个接口，提供了findAll方法用来查询所有的用户。AyUserDao具体代码如下：

package com.ay.dao;

import com.ay.model.AyUser;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface AyUserDao {

List<AyUser> findAll();

}

接口类AyUserDao创建完成之后，我们在/src/main/java/com.ay.service目录下创建对应的服务层接口AyUserService，服务层接口AyUserService代码也非常简单，只提供了一个查询所有用户的方法findAll（），具体的代码如下：

package com.ay.service;

import com.ay.model.AyUser;

import java.util.List;

public interface AyUserService {

List<AyUser> findAll();

}

服务层接口AyUserService开发完成之后，我们在/src/main/java/com.ay.service.impl开发对应的服务层实现类AyUserServiceImpl，实现类主要是注入AyUserDao接口，并实现findAll（）方法，在findAll（）方法中调用AyUserDao的findAll（）方法，具体代码如下所示：

package com.ay.service.impl;

import com.ay.dao.AyUserDao;

import com.ay.model.AyUser;

import com.ay.service.AyUserService;

import org.springframework.stereotype.Repository;

import org.springframework.stereotype.Service;

import javax.annotation.Resource;

import java.util.List;

@Service

public class AyUserServiceImpl implements AyUserService{

@Resource

private AyUserDao ayUserDao;

public List<AyUser> findAll() {

return ayUserDao.listAllUser();

}

}

服务层实现类AyUserServiceImpl开发完成之后，我们在/src/main/java/com.ay.controller目录下创建控制层类AyUserController，并注入服务层接口。AyUserController类只有一个findAll（）方法。在AyUserController类上添加映射路径/user，在findAll（）方法上添加映射路径/findAll。

package com.ay.controller;

import com.ay.model.AyUser;

import com.ay.service.AyUserService;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import javax.annotation.Resource;

import java.util.List;

/\*\*

\*@author Ay

\* @date 2018/04/02

\*/

@Controller

@RequestMapping(value = "/user")

public class AyUserController {

@Resource

private AyUserService ayUserService;

@GetMapping("/findAll")

public String findAll(Model model){

List<AyUser> ayUserList = ayUserService.findAll();

for(AyUser ayUser : ayUserList){

System.out.println("id: " + ayUser.getId());

System.out.println("name: " + ayUser.getName());

}

return "hello";

}

}

最后，我们在/src/main/resources目录下创建AyUserMapper.xml文件，具体代码如下：

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.ay.dao.AyUserDao">

<sql id="userField">

a.id as "id",

a.name as "name",

a.password as "password"

</sql>

<!-- 获取所有用户 -->

<select id="findAll" resultType="com.ay.model.AyUser">

select

<include refid="userField"/>

from ay\_user as a

</select>

</mapper>

## 2.1.5 web项目集成Log4J框架

1、相关的jar包

!-- log4j2 -->

<properties>

//省略部分代码

<slf4j.version>1.7.7</slf4j.version>

<log4j.version>1.2.17</log4j.version>

</properties>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>${log4j.version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>${slf4j.version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>${slf4j.version}</version>

</dependency>

2、

集成Log4j的依赖包添加完成之后，我们在项目的/src/main/java/resources/下创建配置文件log4j.properties，具体代码如下所示：

###set log levels

log4j.rootLogger = DEBUG,Console

###输出到控制台

log4j.appender.Console=org.apache.log4j.ConsoleAppender

log4j.appender.Console.Target=System.out

log4j.appender.Console.layout=org.apache.log4j.PatternLayout

log4j.appender.Console.layout.ConversionPattern= %d{ABSOLUTE} %5p %c{1}:%L - %m%n

## 2.1.6 web项目集成JUnit框架

首先，我们在项目的pom.xml配置文件中添加相关的依赖，具体代码如下：

<!-- junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

然后，我们在项目的/src/main/test/com.ay.test目录下创建测试基类BaseJunit4Test，具体代码如下所示：

/\*\*

\* 描述：测试基类

\* @author Ay

\* @create 2018/05/04

\*\*/

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations={"classpath:applicationContext.xml"})

public class BaseJunit4Test {

}

BaseJunit4Test类开发完成之后，我们在/src/main/test/com.ay.test目录下创建AyUserDaoTest测试类简单测试集成Junit框架是否成功，具体代码如下：

/\*\*

\* 描述：用户DAO测试类

\* @author Ay

\* @create 2018/05/04

\*\*/

public class AyUserDaoTest extends BaseJunit4Test{

@Resource

private AyUserDao ayUserDao;

@Test

public void testFindAll(){

List<AyUser> userList = ayUserDao.findAll();

System.out.println(userList.size());

}

}

作业：@RunWith(SpringJUnit4ClassRunner.class)详解。

3.1 IntelliJ IDEA快速搭建Spring Boot

3.2 Maven Helper插件分析依赖

3.3 GsonFormat插件转化json

1、打开快捷键：Alt + S

实例一

​ { ​ "name": "王五", ​ "gender": "man", ​ "age": 15, ​ "height": "140cm", ​ }

实例二：

​ {​ "address": {​ "city": [​ "Seattle, WA",​ "changsha"​ ],​ "street": ["P.O BOX 54534"],​ "zip": [42452]​ },​ "name": "Ann Michaels",​ "phoneNumbers": [​ "561-832-3180",​ "531-133-9098"​ ]​ },

3.4 Markdown文本编辑器插件

3.5 FindBugs插件快速排查bug

3.6 ECTranslation插件翻译英文

3.7 CheckStyle-IDEA插件检查代码风格

3.8 MyBatisCodeHelper插件生成代码

3.9 Actibpm插件绘制流程图

3.10 Intellij IDEA使用SSH连接服务器

3.11 Rest Client工具使用

3.12 背景图片和代码小视图插件

3.13 GrepConsole自定义控制台颜色

3.14 大小写转化和自动生成序列版本号插件

3.15 IdeaVim文本编辑器插件

3.16 JRebel插件实现热部署

3.17 SonarLint代码质量管理插件

Sonar可以从以下七个维度检测代码质量，而作为开发人员至少需要处理前5种代码质量问题：

1. **不遵循代码标准** sonar可以通过PMD,CheckStyle,Findbugs等等代码规则检测工具规范代码编写

2. **潜在的缺陷** sonar可以通过PMD,CheckStyle,Findbugs等等代码规则检测工具检测出潜在的缺陷

3. **糟糕的复杂度分布** 文件、类、方法等，如果复杂度过高将难以改变，这会使得开发人员难以理解它们 且如果没有自动化的单元测试，对于程序中的任何组件的改变都将可能导致需要全面的回归测试

4. **重复** 显然程序中包含大量复制粘贴的代码是质量低下的，sonar可以展示源码中重复严重的地方

5. **注释不足或者过多** 没有注释将使代码可读性变差，特别是当不可避免地出现人员变动时，程序的可读性将大幅下降 而过多的注释又会使得开发人员将精力过多地花费在阅读注释上，亦违背初衷

6. **缺乏单元测试** sonar可以很方便地统计并展示单元测试覆盖率

7. **糟糕的设计**  通过sonar可以找出循环，展示包与包、类与类之间相互依赖关系，可以检测自定义的架构规则 通过sonar可以管理第三方的jar包，可以利用LCOM4检测单个任务规则的应用情况， 检测耦合。

作业：sonarLint其它内容，希望大家动手查资料完成。

4.1 IntelliJ IDEA构建多模块项目

4.2 作业

1. Intellij IDEA远程调试
2. 插件 Mybatisx使用
3. Intellij IDEA中jetty安装使用
4. .gitignore 忽略提交的文件

**联系方式：**

QQ：[1512079179@qq.com](mailto:1512079179@qq.com)

微信：huangwenyi2010

邮箱地址：huangwenyi10@163.com

**课程服务：**

购买正版视频，提供购买订单号，可以享受以下服务：

1）学员老师一对一交流服务

2）课后作业和思考题服务

3）咨询服务（职业规划、经验交流）

4）购买【盗版视频】无法获得所有的源代码、课件及服务。