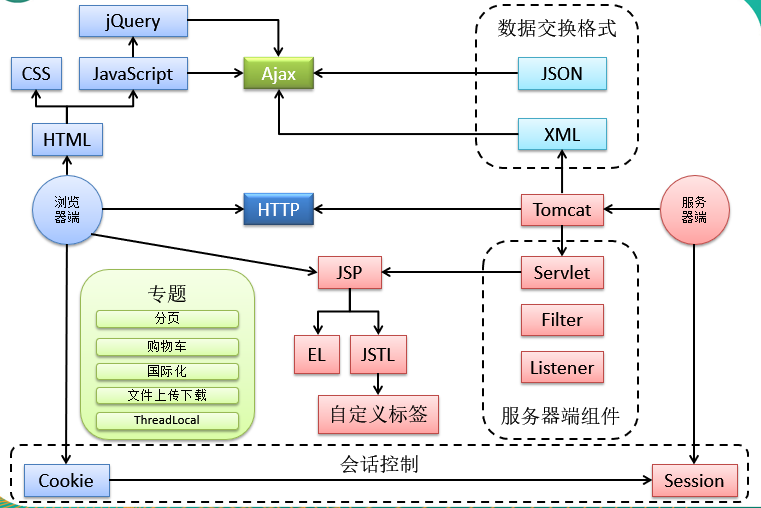
尚硅谷大数据技术之JavaWEB

# 简介

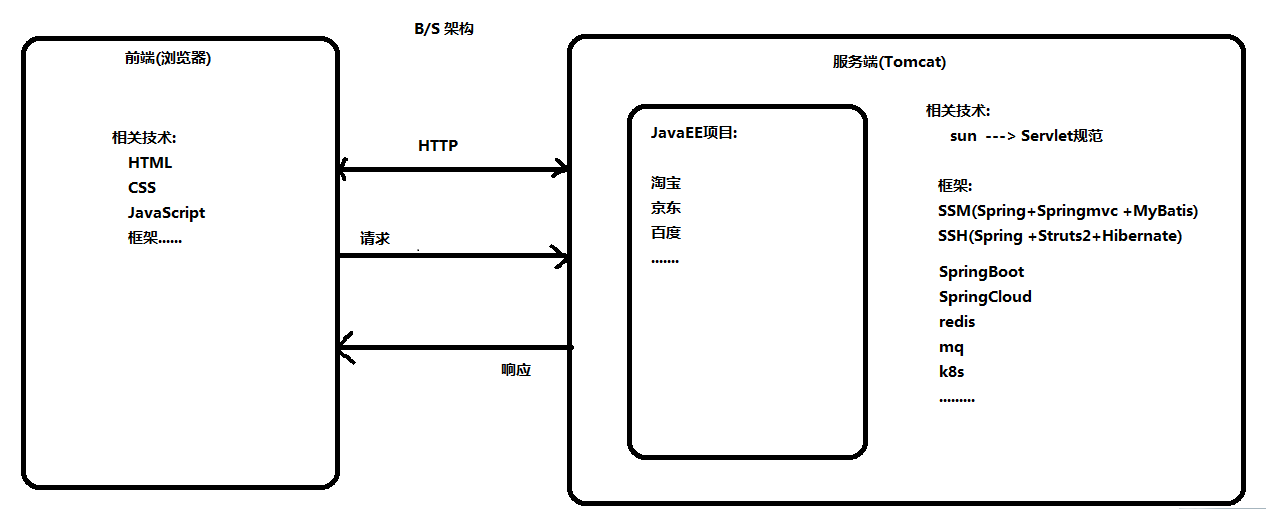
主要目的就是了解JavaWEB的技术体系, 明确前后交互的原理.

对于大数据学员, 需要知道我们分析的数据的来源 以及 分析的结果如何在前端呈现(数据可视化).

# 技术体系



简单理解



# 前端三剑客

## 3.1 HTML

HTML: Hyper Text Markup Language 超文本标记语言

语言: 沟通，交流， 人与计算机之间交流的语言就是编程语言.

标记: HTML的基本单位, 可以理解为语法的格式

文本: 只能支持普通文字(中文，英文，阿拉伯数字, 各种标点符号……)

超文本: 除了支持普通文字， 还支持超链接、表格、图片、音视频 等……

HTML 就是写网页的一门编程语言，提供了很多标记(标签)开开发页面中的各种元素。

举例： 盖房子

### 3.1.1 HTML HelloWorld

<!--

这个是注释

<html></html>: 根标记

<head></head>: 头标记

<body></body>: 体标记

-->

<html>

<head>

<!-- 主要写描述页面的信息 -->

<title>First HTML</title>

</head>

<body>

<!-- 主要写页面要展示的内容 -->

This is my first HTML Page .

<!-- hr: 分割线-->

<hr/>

<!-- a: 超链接-->

<a href="http://www.atguigu.com">国际知名网站</a>

<!-- br: 换行符-->

<br/>

<!-- img:图片-->

<a href="http://www.atguigu.com">

<img src="fangfang.jpg" style='width:400px;height=300px'></img>

<a/>

<hr/>

<!-- form: 表单 , 可以收集用户的信息，例如注册页面，登录页面等

action : 指定表单提交的地址

method : 指定表单提交的方式

-->

<form action='' method=''>

用户名称:<input type='text' name='username'>

<br/>

用户密码:<input type='password' name='password'>

<br/>

用户性别: 男 <input type='radio' name='gender' value='1'>

女 <input type='radio' name='gender' value='0'>

<br/>

用户爱好: 吃饭 <input type='checkbox' name='hobby' value='cf'>

睡觉觉 <input type='checkbox' name='hobby' value='sj'>

编代码 <input type='checkbox' name='hobby' value='cd'>

<br/>

<input type='submit' value='REGIST'>

</form>

</body>

</html>

## 3.2 CSS

CSS: Cascading Style Sheet 层叠样式

CSS: 美化页面（页面的布局，页面的风格等）

举例: 装修房子

### 3.2.1 CSS HelloWorld

<head>

<!-- 主要写描述页面的信息 -->

<title>First HTML</title>

<!-- CSS: 美化页面，一般写到head标签中，或者写到单独的css文件中，再引入到HTML中-->

<style>

#d1{

width: 300px;

height: 200px;

border: 2px solid green ;

background-color: red ;

}

</style>

</head>

<body>

<div id='d1'></div>

</body>

## 3.3 JavaScript

JavaScript: 一门编程语言,脚本语言. 运行在浏览器中，由浏览器解释执行.

JavaScript: 做页面的交互,做页面的动效.

举例: 智能家居

### 3.1.3 JavaScript 基本语法

<!-- JavaScript: 页面交互，让页面动起来 ， 一般写在head标签中，也可以单独写到js文件中，再引入到HTML中-->

<script type="text/javascript">

//弹窗

//window.alert("就想弹个窗");

//定义变量

// 弱类型语言

//var i = 1000 ;

//window.alert(i);

//window.alert(typeof i) ; // number

//i = 'abc' ;

//window.alert(typeof i) ; // string

//i= false;

//window.alert(typeof i) ; // boolean

//i={} ;

// window.alert(typeof i) ; // object

//定义函数

function sum(i,j){

alert(i+j);

}

//调用函数

sum(1,2);

</script>

### 3.1.2 JavaScript 事件

事件主要分 系统事件和用户事件

系统事件: onload 当页面加载完毕后会自动触发onload事件.

用户事件: onclick 点击事件

onblur 失去焦点

onfoucs 获取焦点

onmouseover 鼠标悬停

onmouseout 鼠标移除

…………

function clickpassword(){

//获取密码框

// Dom: 文档对象模型.

var passwordEle = document.getElementById('password');

var passwordEletype = passwordEle.type;

if(passwordEletype=='text'){

passwordEle.type = 'password';

}else{

passwordEle.type = 'text';

}

}

用户密码:<input id='password' type='password' name='password'>

<input type='button' value='小眼睛' onclick='clickpassword();'/>

### 3.1.3 JavaScript框架 JQuery

JQuery: 是一个比较流行的js框架， 能让开发人员很方便的去实现一些相对比较复杂的页面效果.

1. 如何使用

在页面的<head></head>标签中引入Jquery

<script type='text/javascript' src='jquery-1.7.2.min.js'></script>

1. 在页面中编写JQuery代码

// JQuery的语法

$(function(){ //类似于onload,当页面加载完毕就会执行里面的代码

//alert('页面加载完毕了。。。。。');

//动态绑定事件

//$("#smalleye").click(function(){

//});

$("#smalleye").toggle(

function(){

//alert('1')

//$('#password').attr('type','text');

document.getElementById('password').type='text';

},

function(){

//alert('2');

//$('#password').attr('type','password');

document.getElementById('password').type='password';

}

);

});

# 服务端技术

## 4.1 Tomcat服务器

运行在服务器（机器）中, 负责提供JavaEE项目的运行环境，对外提供服务.

1. 安装

将 apache-tomcat-7.0.77-windows-x64.zip直接解压到某个位置,就能直接使用.

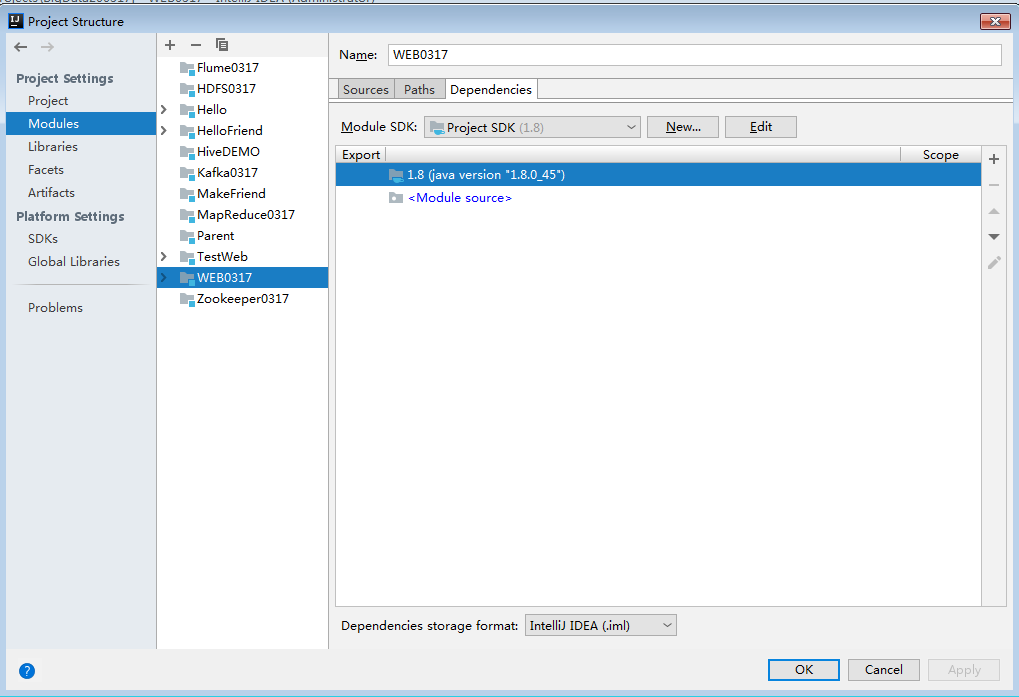
例如: E:/ apache-tomcat-7.0.77

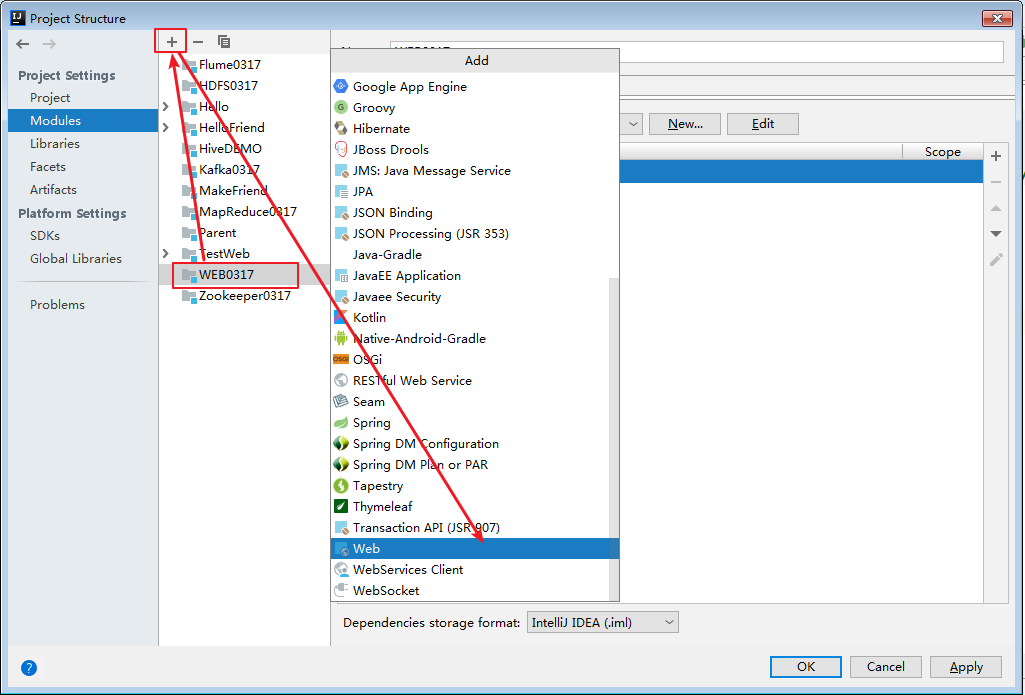
## 4.2 IDEA中创建WEB Module

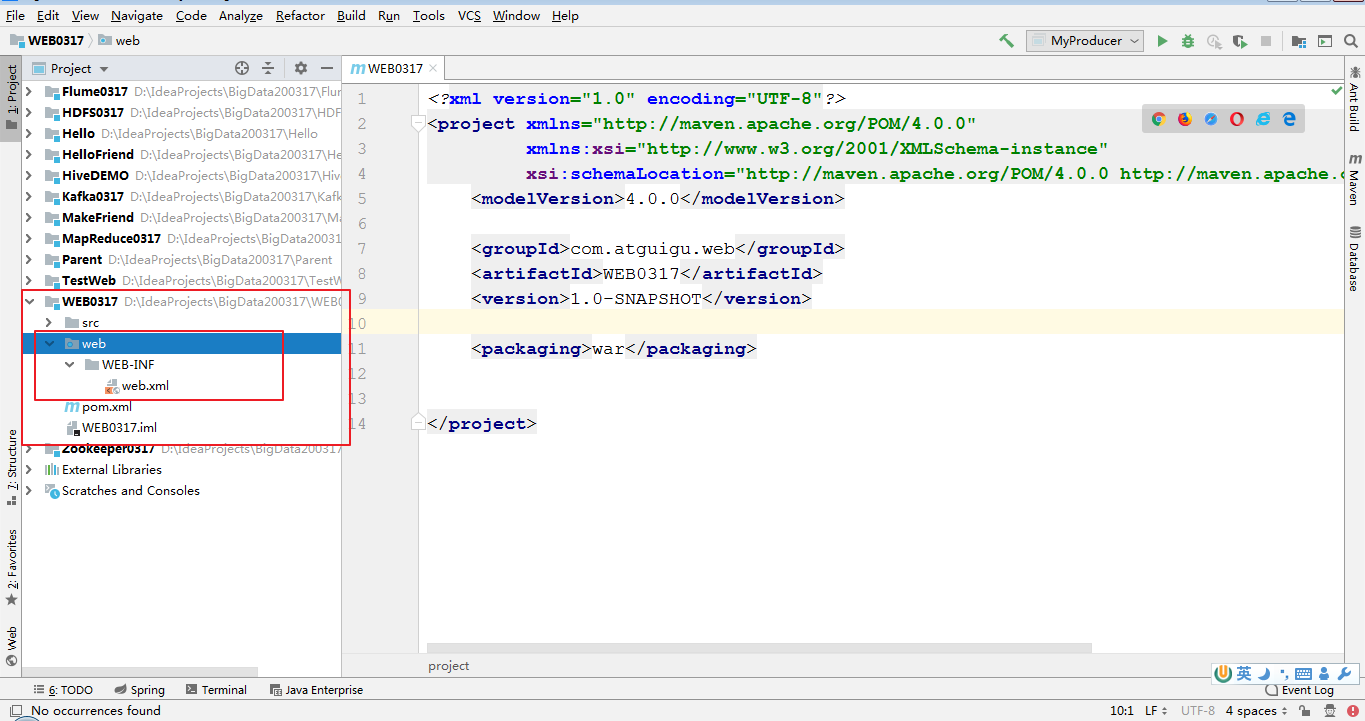
1. 创建 Maven Module
2. 在pom文件中将打包方式改为war

<packaging>war</packaging>

1. 添加web模块







## 4.3 访问项目中的页面

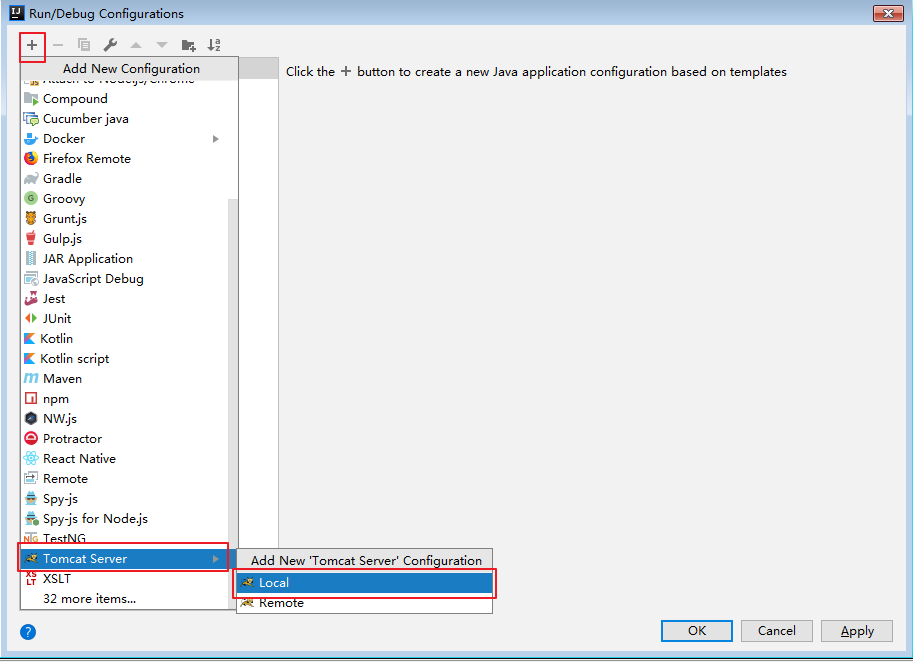
1. 在module中的 src/main/ 新建webapp目录, 在webapp目录下新建一个html目录,

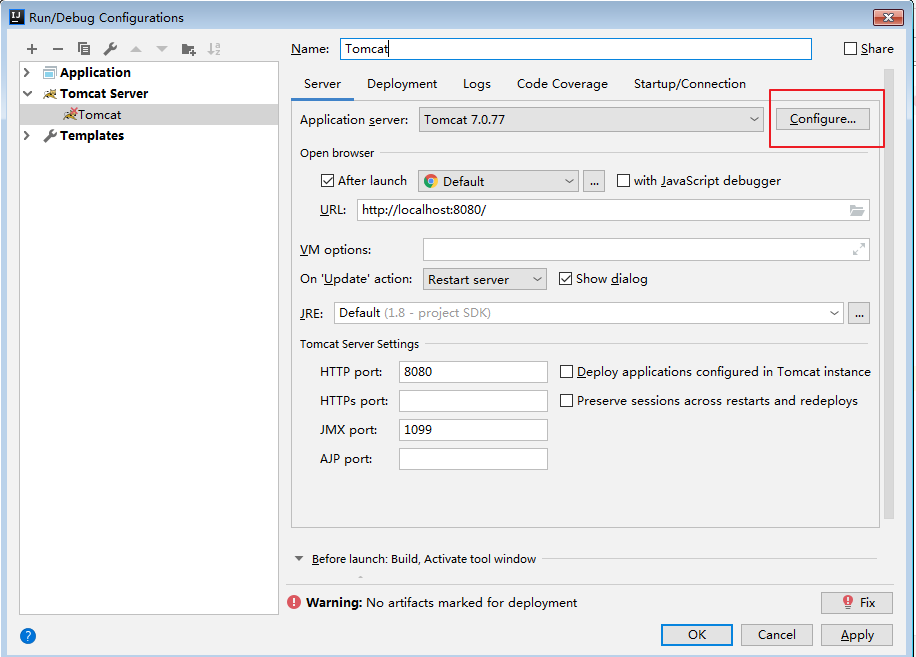
在html目录下新建一个html页面.

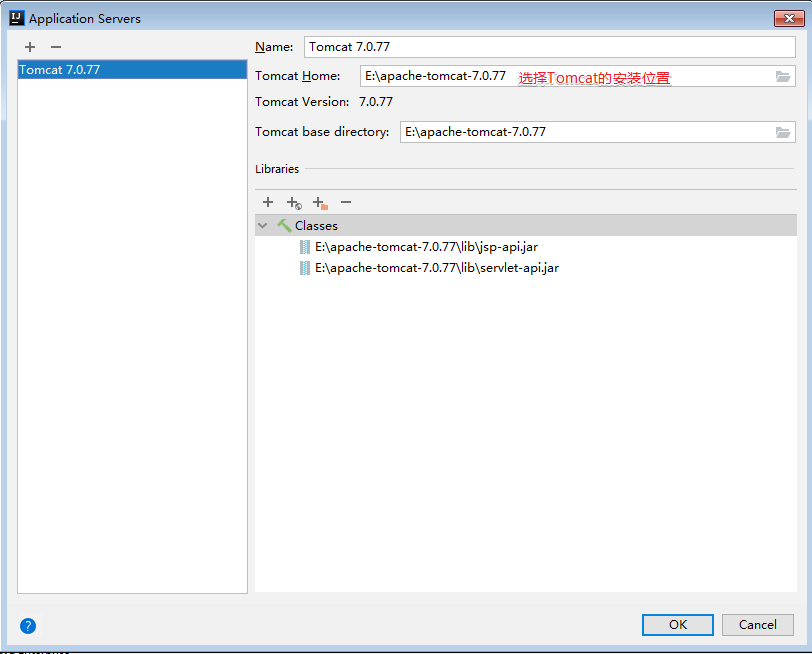
例如: regist.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>REGIST</title>  
 <style>  
 body{  
 background-color: pink;  
 }  
 </style>  
</head>  
<body id="b1">  
 <h1> WELCOME REGIST </h1>  
 <form action="" method="">  
 USERNAME: <input type="text" name="username"/>  
 <br/>  
 PASSWORD: <input type="password" name="password"/>  
 <br/>  
 <input type="submit" value="REGIST"/>  
 </form>  
</body>  
</html>

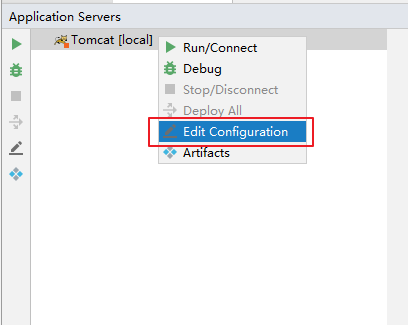
1. 在IDEA中配置tomcat

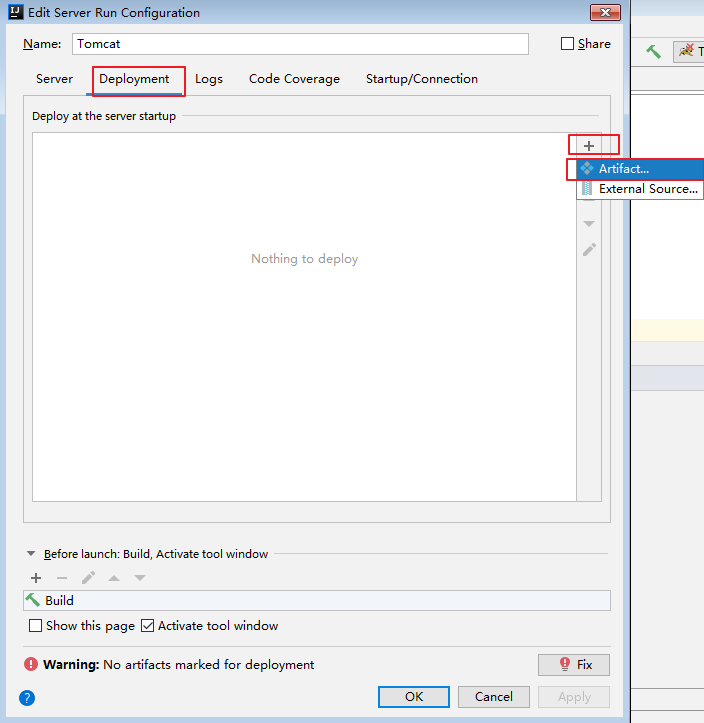


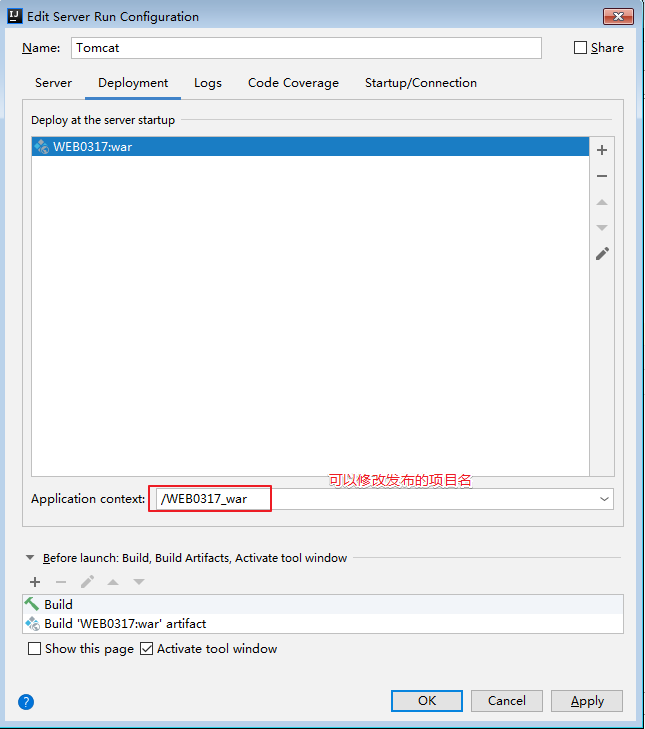


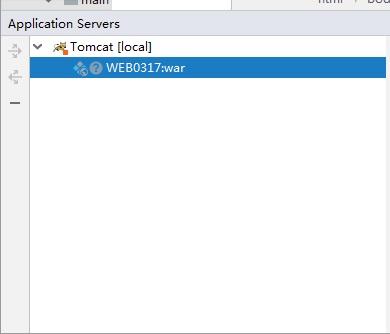


1. 发布项目到Tomcat

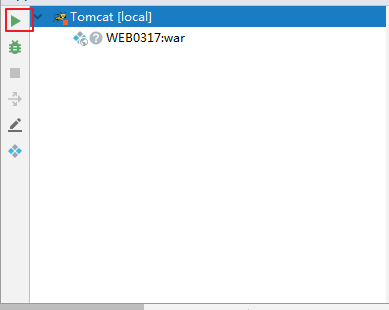




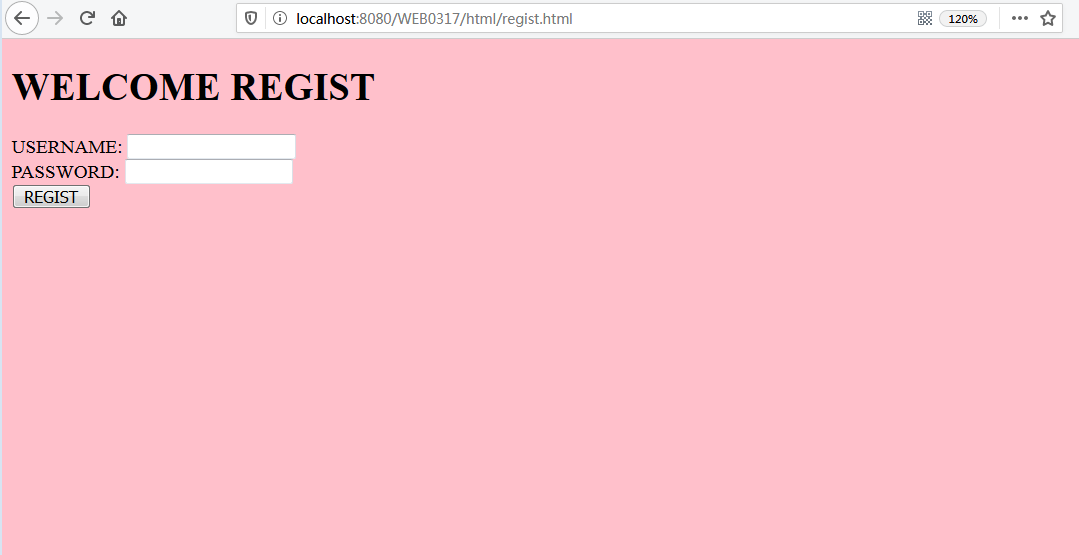




1. 启动Tomcat



1. 在浏览器中进行访问



## 4.4 实现注册功能

1. 修改regist.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>REGIST</title>  
 <style>  
 body{  
 background-color: pink;  
 }  
 </style>  
</head>  
<body id="b1">  
 <h1> WELCOME REGIST </h1>  
 <!--  
 action : 指定访问的请求地址  
 method : 指定请求的方式 GET|POST  
  
  
 表单提交的时候，会将表单的信息一并打包提交给服务端.  
 提交的数据格式为:  
 username=zhangsan  
 password=123456  
 -->  
 <form action="http://localhost:8080/WEB0317/regist" method="post">  
 USERNAME: <input type="text" name="username"/>  
 <br/>  
 PASSWORD: <input type="password" name="password"/>  
 <br/>  
 <input type="submit" value="REGIST"/>  
 </form>  
</body>  
</html>

1. 在pom.xml中添加依赖

<dependencies>  
 <dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>servlet-api</artifactId>  
 <version>2.5</version>  
 </dependency>  
</dependencies>

1. 编写 RegistServlet类

package com.atguigu.servlet;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.io.PrintWriter;  
  
/\*\*  
 \* 基于Tomcat对Servlet规范的实现， 处理客户端的请求.  
 \*  
 \* Sun制定了Servlet规范, Tomcat实现了Servlet规范.  
 \*/  
public class RegistServlet extends HttpServlet {  
  
 /\*\*  
 \* 用于处理客户端的post方式的请求  
 \* @param req  
 \* @param resp  
 \* @throws ServletException  
 \* @throws IOException  
  
 @Override  
 protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 super.doPost(req, resp);  
 }  
 \*/  
  
 /\*\*  
 \* 用户处理客户端的get方式的请求  
 \* @param req  
 \* @param resp  
 \* @throws ServletException  
 \* @throws IOException  
  
 @Override  
 protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 super.doGet(req, resp);  
 }\*/  
  
 /\*\*  
 \* 重写HttpServlet中service的逻辑， 在方法中处理所有的请求  
 \* @param req 请求对象，  
 \* @param resp 响应对象  
 \* @throws ServletException  
 \* @throws IOException  
 \*/  
 @Override  
 protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 //1. 处理请求和响应的乱码问题  
 req.setCharacterEncoding("utf-8");  
 resp.setContentType("text/html;charset=utf-8");  
 PrintWriter writer = resp.getWriter();  
 //2. 获取客户端提交的数据  
 String username = req.getParameter("username");  
 String password = req.getParameter("password");  
  
 //3. 数据的校验  
  
 //4. 数据失败， 注册请求打回. 数据校验成功，继续后续的处理.  
 if("".equals(username) || "".equals(password)){  
 writer.println("REGIST FAIL");  
 writer.close();  
 return ;  
 }  
  
 //5. 将校验成功的数据，写到数据库中.  
 // JDBC  
 System.out.println("REGIST SUCCESS: " + username + " , " + password);  
  
 //6. 给客户端进行响应  
 writer.println("REGIST SUCCESS");  
 writer.close();  
 }  
}

1. 在web.xml中进行请求和Servlet类的映射

<!-- 完成请求与Servlet的映射 -->  
<servlet>  
 <servlet-name>RegistServlet</servlet-name>  
 <servlet-class>com.atguigu.servlet.RegistServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
 <servlet-name>RegistServlet</servlet-name>  
 <!-- 与客户端的请求一致 -->  
 <url-pattern>/regist</url-pattern>  
</servlet-mapping>

1. 浏览器中测试注册功能

## 4.5 其他技术点

1. 请求对象 HttpServletRequest 和响应对象 HttpServletResponse
2. 会话管理 Cookie HttPSession
3. 页面跳转 重定向 转发
4. 过滤器 监听器
5. ………

## 4.6 JSON

JSON就是一种特殊格式的字符串. 主要用于数据的交换.

例如在JavaEE中，前后端的语言不通, 意思是java和 javaScript不是同一门语言,相互

不认识对方的数据格式. 我们需要一个翻译，将双方的数据翻译成对方都认识的格式.

就能够正常进行数据交换.

### 4.6.1 Json的数据类型

1. 字符串

例如: “abcd”

1. 数值类型

例如: 123456

1. 布尔值

例如: true/false

1. null值

例如: null

1. 对象

例如: {“name”: “zhangsan”,”age”: 20,”isMarry”: true}

1. 数组

例如: [123456,false,”abc”, {“name”: “zhangsan”,”age”: 20,”isMarry”: true}]

### 4.6.2 在JS中操作JSON

<html>

<head>

<script type='text/javascript'>

// Json对象(JS对象)

var jsonObj = {"name":"zhangsan","age":22,"isMarry":false};

window.alert(jsonObj);

//window.alert(jsonObj.name);

//window.alert(jsonObj.age);

//window.alert(jsonObj.isMarry);

// Json对象转换成json字符串

var jsonStr = JSON.stringify(jsonObj);

window.alert(jsonStr);

// Json字符串转换重Json对象

var jsonObj1 = JSON.parse(jsonStr);

window.alert(jsonObj1);

// Js --> json字符串 --> 发送给java

// java对象 --> json字符串 --> js

</script>

</head>

<body></body>

</html>

### 4.6.3 在Java中操作JSON

Java本身没有直接提供操作Json的库,但是第三方的操作Json的库有很多,

例如: Gson , Jackson , json-lib , fastJon ……

1. 在pom.xml中加入Gson的依赖

<dependency>  
 <groupId>com.google.code.gson</groupId>  
 <artifactId>gson</artifactId>  
 <version>2.8.5</version>  
</dependency>  
<dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
</dependency>

1. 测试Json字符串和java对象的转换

public class TestJson {  
  
 @Test  
 public void testJavaToJson(){  
 User user = new User();  
 user.setUsername("fangfang");  
 user.setPassword("123456");  
 Gson gson = new Gson() ;  
 String jsonStr = gson.toJson(user);  
 System.out.println(jsonStr);  
  
  
 ArrayList users = new ArrayList();  
 User user1 = new User();  
 user1.setUsername("kangkang");  
 user1.setPassword("654321");  
 users.add(user);  
 users.add(user1);  
  
 String listJsonStr = gson.toJson(users);  
 System.out.println(listJsonStr);  
 }  
  
 @Test  
 public void testJsonToJava(){  
 String jsonStr = "{\"username\":\"fangfang\",\"password\":\"123456\"}" ;  
 Gson gson = new Gson();  
  
 User user = gson.fromJson(jsonStr, User.class);  
 System.out.println(user);  
 }  
}

# SpringBoot + 通用Mapper

框架: 一定是简化开发. 通过良好的设计，运用各种设计模式，通过封装一些基础的工作进行开发的简化。

SpringBoot : Spring家族中非常流行的框架， 帮助我们简化项目的初始搭建及开发过程.

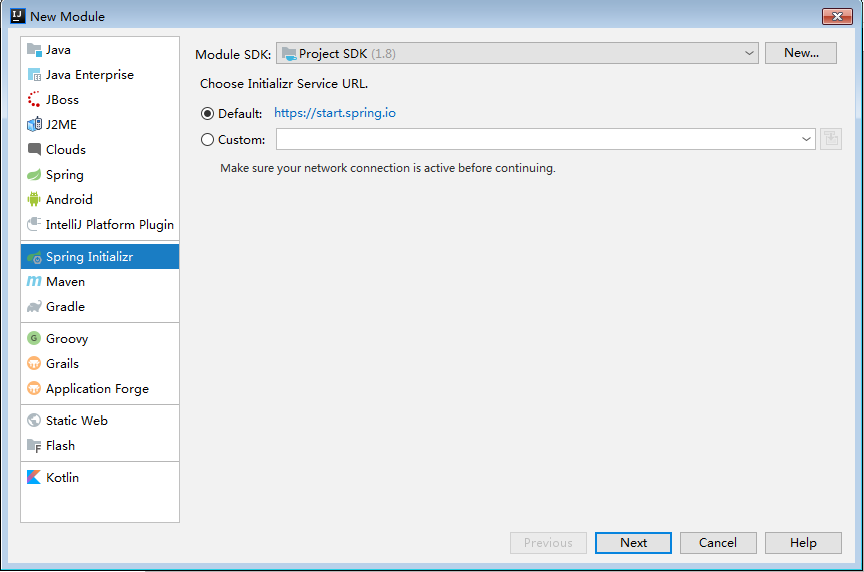
最开始写代码都是原生的API –->Spring框架 –-> SpringBoot框架

通用Mapper: 对Mybatis框架的二次封装. 提供非常便捷的基础操作.

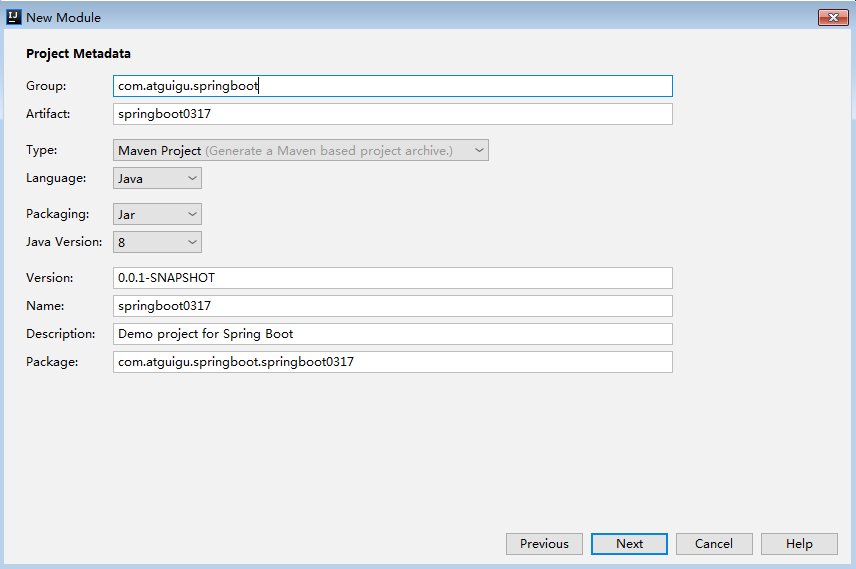
最开始写原生的JdbcAPI --> Mybatis --> 通用Mapper

## 5.1 创建Spring Boot项目

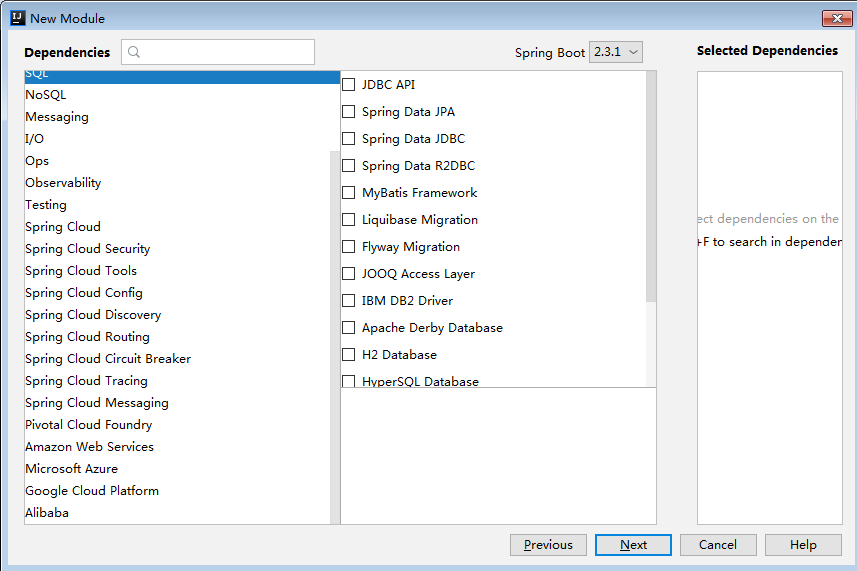
1. 创建Spring Initializr

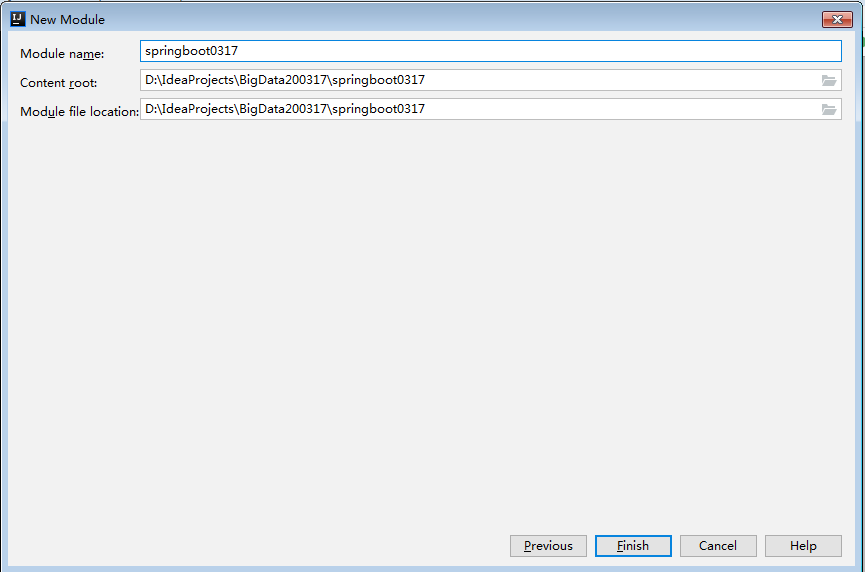


编写gav

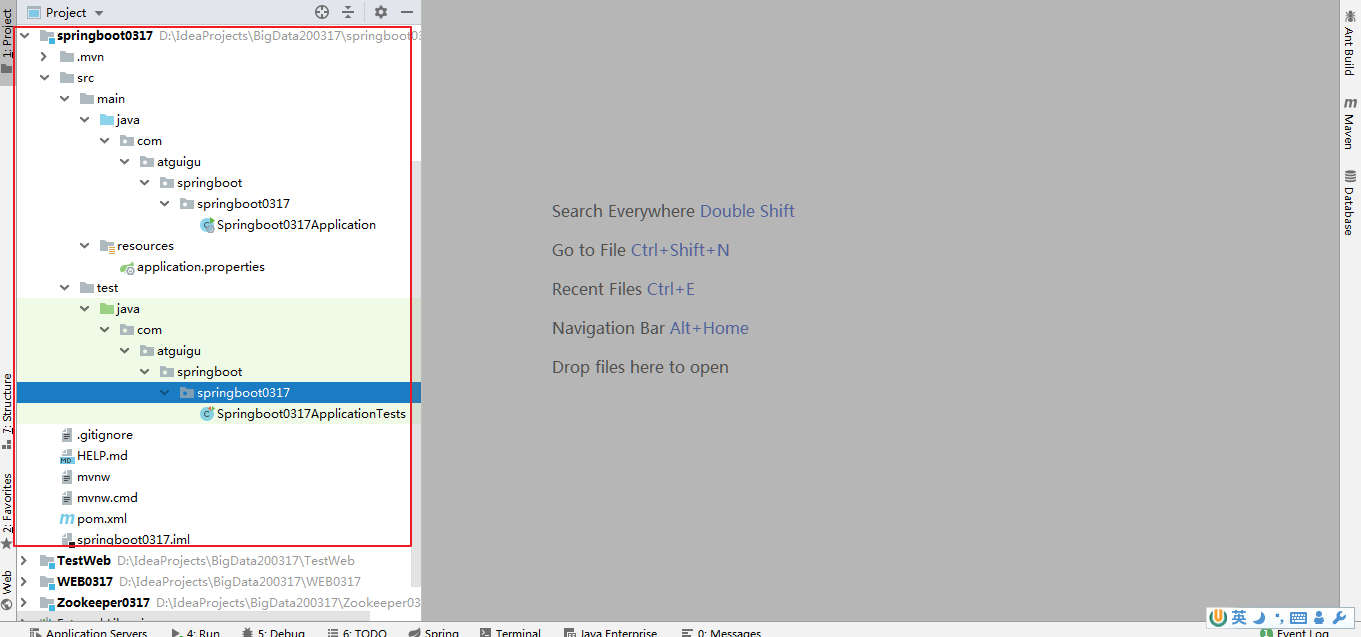


选择集成的组件和版本





创建好后的Springboot项目



## 5.2 基于SpringBoot + 通用mapper实现注册功能

1. 在src/main/resources 新建static/templates 目录，在目录下新建regist.html
2. 将Springmvc模块添加进来（Springmvc就是对Servlet的封装）

在pom.xml中添加web-starter依赖（内置了Tomcat）

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
</dependency>

1. 运行启动类

package com.atguigu.springboot.springboot0317;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
/\*\*  
 \* SpringBoot项目的启动类  
 \*/  
@SpringBootApplication  
public class Springboot0317Application {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Springboot0317Application.class, args);  
 }  
  
}

1. 浏览器访问regist.html

<http://localhost:8080/templates/regist.html>

如果要改变端口或者访问的路径等，需要在src/main/resources/application.proeprties文件中设置

server.servlet.context-path=/springboot0317  
server.port=8888

1. 新建beans包， 在包下新建User类

package com.atguigu.springboot.beans;  
  
public class User {  
 private Integer id ;  
  
 private String username ;  
  
 private String password ;  
  
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 @Override  
 public String toString() {  
 return "User{" +  
 "id=" + id +  
 ", username='" + username + '\'' +  
 ", password='" + password + '\'' +  
 '}';  
 }  
}

1. 新建controller包，包下新建UserController,

package com.atguigu.springboot.controller;  
  
import com.atguigu.springboot.beans.User;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
import org.springframework.web.bind.annotation.RestController;  
  
/\*\*  
 \* 控制层 : 用于处理客户端的请求，以及给客户端完成响应  
 \*/  
  
// @RestController 相当于 @Controller + @ResponseBody  
@Controller  
public class UserController {  
  
 /\*\*  
 \* 客户端: http://localhost:8888/springboot0317/regist  
 \*  
 \* @RequestMapping("/regist") : 将来处理客户端的regist请求  
 \*  
 \* 方法的user参数用来接收客户端提交的参数，只需要保证客户端的参数名与user类中的属性名一致，就能直接将  
 \* 客户端的参数封装到User类中  
 \*  
 \* @ResponseBody 将方法的返回值处理成json字符串响应给客户端  
 \*  
 \*  
 \*/  
 @ResponseBody  
 @RequestMapping("/regist")  
 public String doRegist(User user){  
 System.out.println("Controller User : " +user );  
 return "regist success" ;  
 }  
}

1. 在SpringBoot的启动类上，添加注解@ComponentScan

package com.atguigu.springboot.springboot0317;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.ComponentScan;  
  
/\*\*  
 \* SpringBoot项目的启动类  
 \* @ComponentScan(value="com.atguigu") 会扫描com.atguigu包下及子包下所有的类,将带有注解的类管理到Spring的容器中  
 \*  
 \*/  
@ComponentScan(value="com.atguigu")  
@SpringBootApplication  
public class Springboot0317Application {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Springboot0317Application.class, args);  
 }  
  
}

1. 集成通用mapper,在Pom.xml中加入如下依赖

<dependency>  
 <groupId>tk.mybatis</groupId>  
 <artifactId>mapper-spring-boot-starter</artifactId>  
 <version>2.0.2</version>  
</dependency>  
<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
</dependency>

1. 准备数据表

CREATE DATABASE bigdata0317 ;

USE bigdata0317 ;

CREATE TABLE t\_user(

id INT(11) PRIMARY KEY AUTO\_INCREMENT ,

username VARCHAR(50),

PASSWORD VARCHAR(50)

);

1. 配置数据库的连接，在src/main/resources/application.properties中

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/bigdata0317?serverTimezone=UTC  
spring.datasource.username=root  
spring.datasource.password=1234

1. 编写UserMapper

package com.atguigu.springboot.mapper;  
  
import com.atguigu.springboot.beans.User;  
import tk.mybatis.mapper.common.Mapper;  
  
/\*\*  
 \* Mapper : 通用Mapper中提供的接口。 已经提供了基本的CRUD操作.  
 \*  
 \*/  
public interface UserMapper extends Mapper<User> {  
   
}

1. 在SpringBoot启动类上，添加注解@MapperScan()

package com.atguigu.springboot.springboot0317;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.ComponentScan;  
import tk.mybatis.spring.annotation.MapperScan;  
  
/\*\*  
 \* SpringBoot项目的启动类  
 \* @ComponentScan(value="com.atguigu") 会扫描com.atguigu包下及子包下所有的类,将带有注解的类管理到Spring的容器中  
 \*  
 \*/  
@MapperScan(basePackages = "com.atguigu.springboot.mapper")  
@ComponentScan(value="com.atguigu")  
@SpringBootApplication  
public class Springboot0317Application {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Springboot0317Application.class, args);  
 }  
  
}

1. 新建service包，在包下新建UserService接口,

package com.atguigu.springboot.service;  
  
import com.atguigu.springboot.beans.User;  
  
/\*\*  
 \* 业务层  
 \*  
 \*/  
public interface UserService {  
  
 int registuser(User user);  
}

1. 在service包下，新建UserServiceImpl类

package com.atguigu.springboot.service;  
  
import com.atguigu.springboot.beans.User;  
import com.atguigu.springboot.mapper.UserMapper;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
@Service  
public class UserServiceImpl implements UserService {  
  
 @Autowired // SpringBoot会将UserMapper对象注入进来  
 private UserMapper userMapper ;  
  
 @Override  
 public int registuser(User user) {  
 return userMapper.insert(user); // 把数据写入到数据库  
 }  
}

1. 在UserController中 调用UserService

package com.atguigu.springboot.controller;  
  
import com.atguigu.springboot.beans.User;  
import com.atguigu.springboot.service.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
import org.springframework.web.bind.annotation.RestController;  
  
/\*\*  
 \* 控制层 : 用于处理客户端的请求，以及给客户端完成响应  
 \*/  
  
// @RestController 相当于 @Controller + @ResponseBody  
@Controller  
public class UserController {  
  
 @Autowired  
 private UserService userService;  
  
 /\*\*  
 \* 客户端: http://localhost:8888/springboot0317/regist  
 \*  
 \* @RequestMapping("/regist") : 将来处理客户端的regist请求  
 \*  
 \* 方法的user参数用来接收客户端提交的参数，只需要保证客户端的参数名与user类中的属性名一致，就能直接将  
 \* 客户端的参数封装到User类中  
 \*  
 \* @ResponseBody 将方法的返回值处理成json字符串响应给客户端  
 \*  
 \*  
 \*/  
 @ResponseBody  
 @RequestMapping("/regist")  
 public String doRegist(User user){  
 System.out.println("Controller User : " +user );  
 userService.registuser(user);  
 return "regist success" ;  
 }  
}

1. 需要注意一个问题:

默认情况下，通用mapper会根据实体类(User)的名字到数据库中找到对应名字的表。

如果想具体指定，可以在实体类上通过@Table注解来指定对应那张表

package com.atguigu.springboot.beans;  
  
import javax.persistence.Table;  
  
@Table(name = "t\_user") //指定表名  
public class User {  
 private Integer id ;  
  
 private String username ;  
  
 private String password ;  
  
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 @Override  
 public String toString() {  
 return "User{" +  
 "id=" + id +  
 ", username='" + username + '\'' +  
 ", password='" + password + '\'' +  
 '}';  
 }  
}

# 第六章

其实还有很多东西没有提到.奈何时间飞逝，缘分已尽。只能江湖再见……

今日的目的了解JavaWEB的相关东西，明确数据的来源和数据的展示.

如果想深入学习JavaWEB，尚硅谷官网找JavaEE视频观看. 或者报名再来一次.