

# Java 开发接口

吴小龙同學

---

# Table of Contents

序	1.1
环境搭建	1.2
JDK 安装	1.2.1
Tomcat 安装	1.2.2
IntelliJ IDEA 安装	1.2.3
MySQL 安装	1.2.4
Hello World	1.3
Servlet	1.4
创建 Servlet	1.4.1
创建数据库	1.4.2
Servlet 写接口	1.4.3
传参	1.4.4
Spring	1.5
创建 Spring MVC	1.5.1
Spring MVC 写接口	1.5.2
Spring + SpringMVC + MyBatis	1.5.3
云服务器部署	1.6

## Java 开发接口

我们开发 APP，就是通过接口与后台进行数据交互的，那这个接口是怎么做的呢？有兴趣学习吗？问题来了，怎么学？从何学起？虽然 Android 开发也是 Java 语言，但真的要搞后端，还是不知所云，该学什么呢？我利用清明假期和业余时间，写了一份 Java 开发接口的入门教程，来看看。编程有一点好处，操作性很强，认准一个正确的方向，只要肯跟着实践练习起来，必然有收获。凭世人努力程度，还轮不到拼天赋/捂脸。

## 适合对象

有一定编程基础或 **Android** 开发人员。

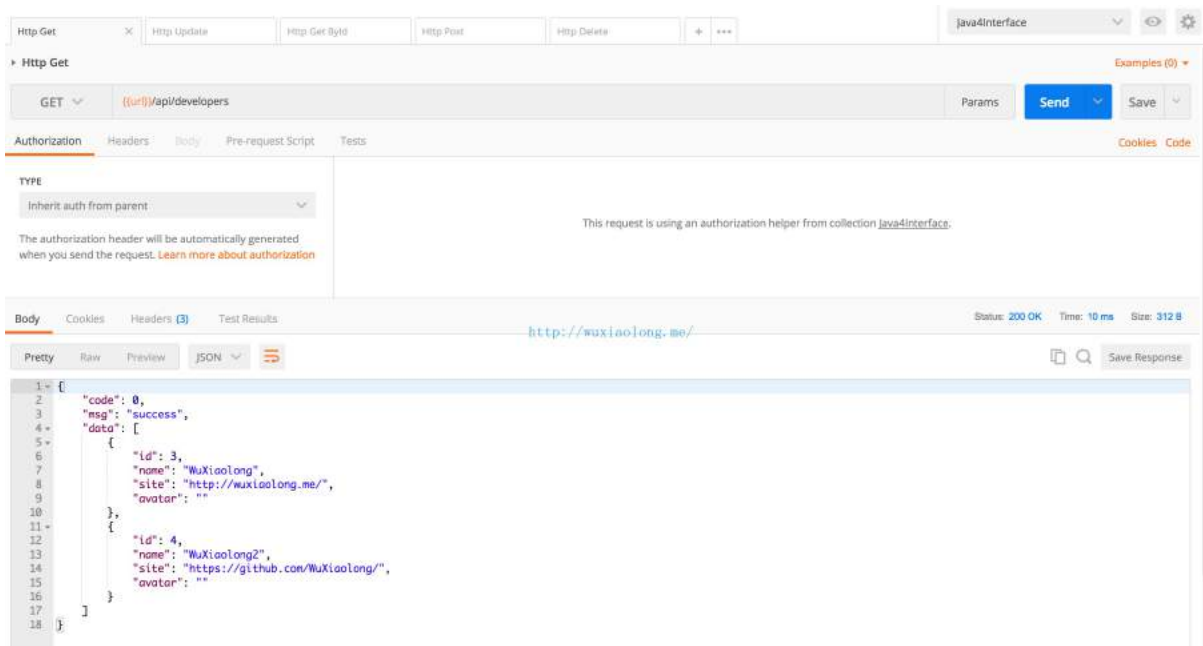
## 学习目标

入门，会用 Java 写接口。

## 学习方法

谷歌“Java 开发接口”，看看网上博客，大部分文章都很老，有 Eclipse 的，只能有选择性参考，有些是有问题的，跟着实践，错上加错；远水救不了近火，虚（qiang）心（po）请教公司后台同事，问问她现在怎么做的，用的哪些框架等，一个字，就是磨，花时间，先运行起来，看到效果再说；有目标性去学习，比如我想要接口返回什么格式，就朝着那个方向前行。原则是先做出来，看到效果，再搞明白这行代码是什么意思，爬坑爬的辛苦啊，幸亏我没有放弃，做出来了。学习过程真是困难重重，解决一个，又来一个，没人能帮我，只能谷歌，很容易陷于死胡同，放置一天再看，有可能就迎刃而解。

## 成果展示



接口请求方式包括常用的 GET、POST、PUT、DELETE，数据是来自本地数据库。



接口是与我们 APP 开发息息相关，通过这套教程，我们能学习到 Java 开发环境的搭建；Servlet 写接口；SSM（Spring、SpringMVC 和 Mybatis）等，对后台开发有个大概了解入门，学什么都不是那么简单的事，想要做好 Java 后台要学习太多的东西，之后得自己不断精进了。

## 作者简介

我是吴小龙，网名 **吴小龙同学**，坐标无锡，Android 应用攻城狮，性情中人，有点小闷骚。

个人博客：<http://wuxiaolong.me>

我的邮箱: wuxiaolong.me@gmail.com

GitHub: <https://github.com/WuXiaolong>

新浪微博: 吴小龙同學

知乎: <http://zhuanlan.zhihu.com/WuXiaolong>

CSDN: <http://blog.csdn.net/wuxiaolongtongxue>

博客园: <http://www.cnblogs.com/WuXiaolong>

SegmentFault: <https://segmentfault.com/blog/wuxiaolong>

掘金: <https://juejin.im/user/554b876fe4b0f6e981675644>

简书: <https://www.jianshu.com/u/ccf7537a4e7d>

公众号: 吴小龙同学, 欢迎扫码与我交流~



## 结语

Android 开发没人要了? 先不说 Android 是不是真的没人要了, 多学一样东西不是坏事, 《[Python 3 极简教程.pdf](#)》我自学了 Python, 现在 Python 玩玩爬虫没问题。就现在形势来看, 后台才是常青树, 我坚持 Android 为主, 其他为辅, 但也要未雨绸缪, 我的建议: 1、坚持 Android, 做到精通; 2、一线城市, 学习 Python; 3、二线城市, 学 Java 转后端。为什么这样建议? 不管做什么, 专家级都是很受欢迎的; Python 现在比较火, 趁早学习, 抓住这波红包, 不是说一线城市 Java 不好, 只是 Python

相对更好，Python 也可以做后端；为什么说二线城市学 Java，因为 Python 就业岗位太少了，学完了，却找不到工作，也是白学，Java 更稳些。如果从零开始转其他，肯定不切实际，起码接受不了薪资的落差，如果有机会，公司内部转岗是最好的，薪资会不变或变动很小，一起加油了哦。

## 更新记录

- v0.01：初稿，写于 2018 年清明假期；
- v0.02：2018 年 6 月。

## 利器准备

君子生非异也，善假于物也，好的工具让开发事半功倍。

- [JDK](#)： v 10；
- [Tomcat](#)： v 9.0.6；
- [IntelliJ IDEA](#)： v 2017.3，最好跟我用同一个版本，不同的版本，有差异；
- [MySQL](#)： v 5.7.19

# JDK 安装

## 下载

JDK : [官网](#)，下载安装。如果是 Android 开发人员，Android Studio 已经自带了 JDK，当然也可以自己去下载，这里我也下载了。没想到这里影响到了 Android 本身，我打包 aar，结果报错了：

```
./gradlew assembleRelease

FAILURE: Build failed with an exception.

* What went wrong:
A problem occurred configuring project ':ambilWarna'.
> Failed to notify project evaluation listener.
    > Could not initialize class com.android.sdklib.repository.AndroidSdkHandler

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log output.

* Get more help at 链接: Gradle | Search for Help with Gradle

BUILD FAILED in 0s
```

谷歌了一下，大概说 JDK 的 \$JAVA\_HOME 环境变量出了问题，还是换回来之前的 Android Studio 的路径，试了下，问题得以解决，也试下了 Tomcat，也能正常运行，OK！

## 环境变量配置

### Mac

1、在命令行下，进入用户目录

```
cd $HOME
```

2、.bash\_profile 文件

输入下行命令获取当前文件列表：

```
ls -al
```

查看文件列表，如果文件已经存在，则进行下一步。如果没有 .bash\_profile 文件，执行以下命令新建：



```
touch .bash_profile
```

### 3、打开.bash\_profile 文件

执行下行命令打开文件

```
open -e .bash_profile
```

说明：

```
open .bash_profile: 打开文件
touch .bash_profile: 如果文件不存在就创建文件
open -e bash_profile: 编辑文件
```

添加 JDK 安装路径到 .bash\_profile 文件里：

```
# jdk
export JAVA_HOME=/Applications/Android\ Studio.app/Contents/jre/jdk/Contents/Home
# export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk-10.jdk/Contents/Home
export PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
```

### 4、更新刚配置的环境变量

```
source .bash_profile
```

## Windows

配置环境变量：右击“我的电脑”-->“高级”-->“环境变量”。

#### 1、JAVA\_HOME 环境变量

在系统变量里点击新建，变量名填写 JAVA\_HOME，变量值填写 JDK 的安装路径(我的 JDK 安装路径 C:\Program Files\Java\jdk1.8.0\_171)

#### 2、CLASSPATH 环境变量

在系统变量里点击新建，变量名填写 CLASSPATH，变量值填写 .;%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar。CLASSPATH 变量名字，可以大写也可以小写。注意不要忘记前面的点和中间的分号，且要在英文输入的状态下的分号和逗号。

#### 3、Path 环境变量

Path 变量，是系统自带的，不用新建，找到 Path 双击，在后面加上：;%JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin”，注意前面的分号。

## 验证安装是否成功

终端输入命令行 `java -version` :

```
java -version
java version "10" 2018-03-20
Java(TM) SE Runtime Environment 18.3 (build 10+46)
Java HotSpot(TM) 64-Bit Server VM 18.3 (build 10+46, mixed mode)
```

## Tomcat 安装

启动 Tomcat, startup.bat 要调用 catalina.bat, catalina.bat 运行要用到 JAVA\_HOME 环境变量, 所以安装 JDK 是第一步。

## 下载

Tomcat: [官网](#), 下载 zip 或 tar.gz, 下载完成解压, 我放在 /Users/wuxiaolong/Library/apache-tomcat-9.0.6/ 下了。

### 9.0.6

Please see the [README](#) file for packaging information. It explains what every distribution contains.

#### Binary Distributions

- Core:
  - [zip \(pgp, sha1, sha512\)](#)
  - [tar.gz \(pgp, sha1, sha512\)](#)
  - [32-bit Windows zip \(pgp, sha1, sha512\)](#)
  - [64-bit Windows zip \(pgp, sha1, sha512\)](#)
  - [32-bit/64-bit Windows Service Installer \(pgp, sha1, sha512\)](#)
- Full documentation:
  - [tar.gz \(pgp, sha1, sha512\)](#)
- Deployer:
  - [zip \(pgp, sha1, sha512\)](#)

## 环境变量配置

### Mac

.bash\_profile 文件加入: `export PATH=$PATH:/Users/wuxiaolong/Library/apache-tomcat-9.0.6/bin`。

查看Tomcat版本信息

```
sh catalina.sh version
```

### Windows

在系统变量里点击新建, 变量名填写 CATALINA\_HOME, 变量值填写 Tomcat 的下载路径;

Path 变量, 添加变量值: `;%CATALINA_HOME%\lib;%CATALINA_HOME%\bin`。

## 启动 Tomcat

## Mac

如果没有配置 Tomcat 环境变量，使用 `./startup.sh` 和 `./shutdown.sh` 来启动和关闭 Tomcat。

如果启动遇到 `-bash: ./startup.sh: command not found` 提示，使用 `chmod +x *.sh` 设置权限。

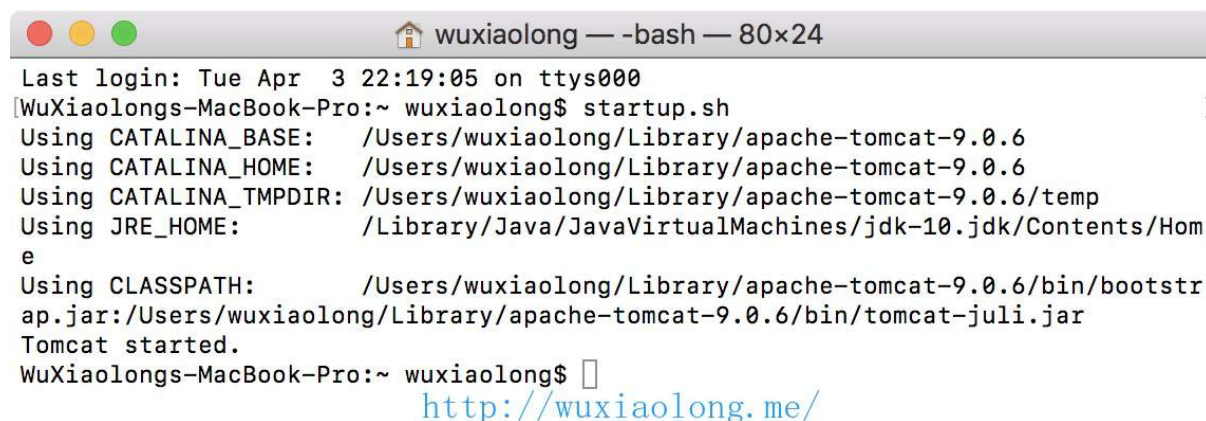
如果配置了 Tomcat 环境变量，终端任意位置输入：`startup.sh` 或 `shutdown.sh` 就能开启或关闭 Tomcat 了。

## Windows

终端任意位置输入：`startup.bat` 或 `shutdown.bat` 开启或关闭 Tomcat。

## 启动 Tomcat

如下图，启动了 Tomcat：

A screenshot of a macOS terminal window. The title bar shows the user 'wuxiaolong' and the shell '-bash' with a window size of '80x24'. The terminal text shows the user logging in on 'Tue Apr 3 22:19:05 on ttys000'. They run the command '[WuXiaolongs-MacBook-Pro:~ wuxiaolong\$ startup.sh]'. The output shows the configuration of Tomcat 9.0.6, including the base, home, temp directory, JRE home, and classpath. It ends with 'Tomcat started.' and a prompt for the user to visit 'http://wuxiaolong.me/'.

```
Last login: Tue Apr 3 22:19:05 on ttys000
[WuXiaolongs-MacBook-Pro:~ wuxiaolong$ startup.sh]
Using CATALINA_BASE:   /Users/wuxiaolong/Library/apache-tomcat-9.0.6
Using CATALINA_HOME:   /Users/wuxiaolong/Library/apache-tomcat-9.0.6
Using CATALINA_TMPDIR: /Users/wuxiaolong/Library/apache-tomcat-9.0.6/temp
Using JRE_HOME:        /Library/Java/JavaVirtualMachines/jdk-10.jdk/Contents/Home
Using CLASSPATH:        /Users/wuxiaolong/Library/apache-tomcat-9.0.6/bin/bootstrap.jar:/Users/wuxiaolong/Library/apache-tomcat-9.0.6/bin/tomcat-juli.jar
Tomcat started.
WuXiaolongs-MacBook-Pro:~ wuxiaolong$ [
http://wuxiaolong.me/
```

浏览器输入 <http://localhost:8080/>，如果出现一只猫，则说明配置成功~



## 踩坑记录

我在安装 Tomcat 过程遇到了，Tomcat 启动成功了，但是打开 <http://localhost:8080/> 始终 404，网上说是 8080 端口被占用，Mac 上可以用命令 `sudo lsof -i:8080` 查看，确实被 QQ 占用了，用 kill 命令杀该进程，命令形式是 `sudo kill pid`，其中 pid 就是 pid 号。结果还是不行，折腾了很久，去 `apache-tomcat-9.0.6/logs` 下，查看 `catalina.out`，这里有报错信息，最后我重启了电脑，解决之，顿时泪流满面。

## IntelliJ IDEA 安装

做 Android 开发想必都知道，Android Studio 就是出自 JetBrains 之手，功能强大，IntelliJ IDEA 也是如此，其实 IntelliJ IDEA 也是可以开发 Android 的，IntelliJ IDEA 最新版本是 2018.1，2018 年 3 月 27 日刚发布，我现在用的是前面的版本 2017.3。

最新版本下载：<https://www.jetbrains.com/idea/download/>，历史版本下载：<https://www.jetbrains.com/idea/download/previous.html>。

我试着升级到 2018.1，但遇到了一些问题，我是小白，不想花这个时间去折腾，降回来了，就像 Android Studio 一样，每次升级，都会折腾半天改错。

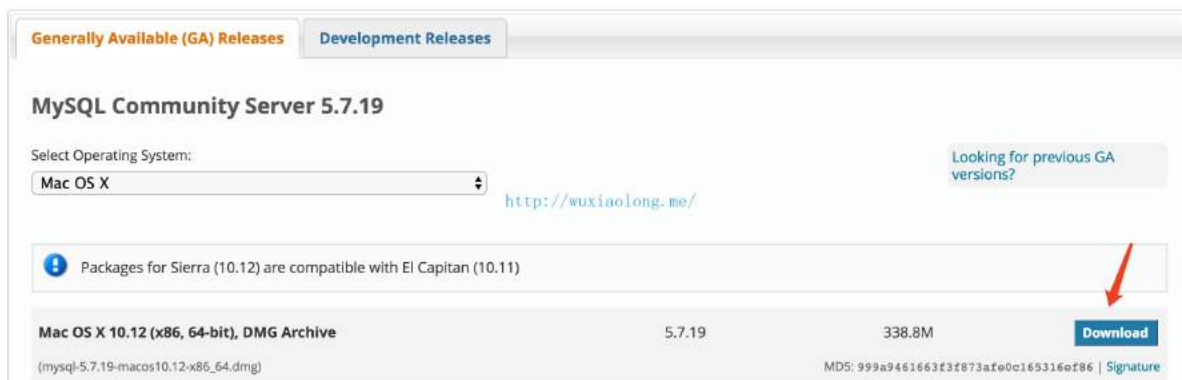


## MySQL 安装

教程基于 Mac, Windows 上如何安装 MySQL, 自己网上搜下吧。

## 下载 MySQL

访问 MySQL 官网: <https://dev.mysql.com/downloads/mysql/>



点击, 下个页面会提示需不需要注册的, 直接选择最下面的“No thanks, just start my download.”进行下载。

## Begin Your Download

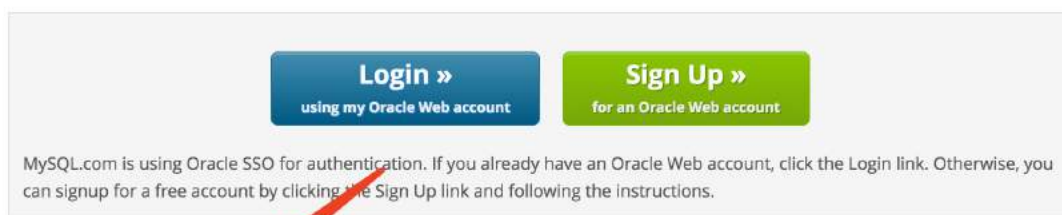
mysql-5.7.19-macos10.12-x86\_64.dmg

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system
- Comment in the MySQL Documentation

<http://wuxiaolong.me/>



[No thanks, just start my download.](#)

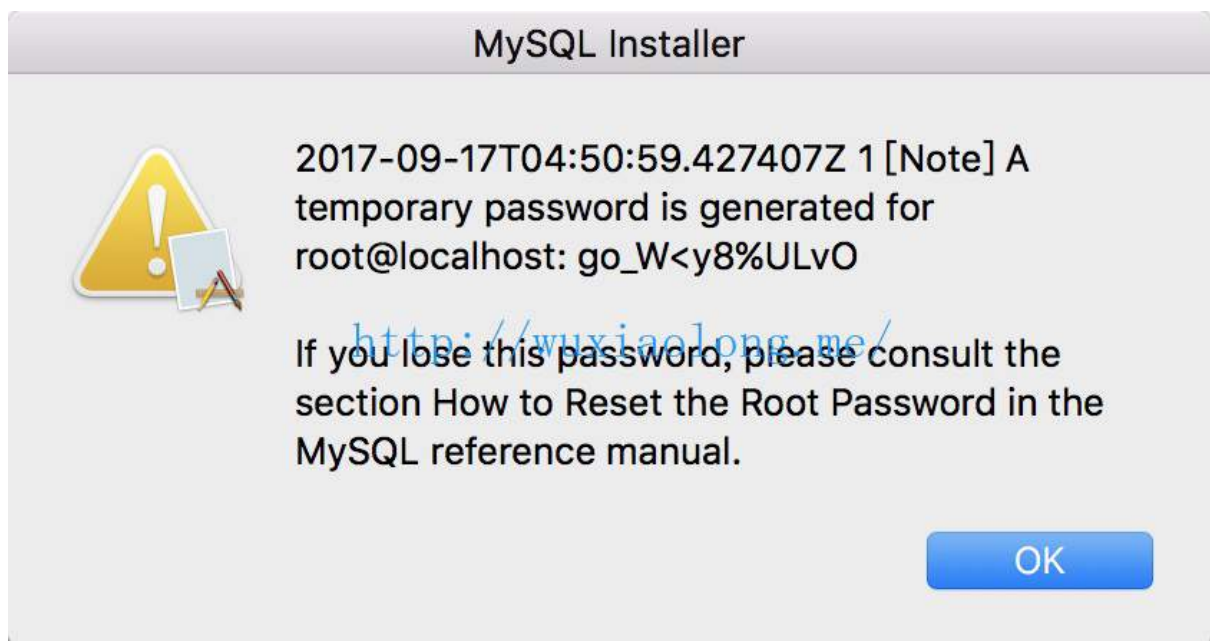
## 安装 MySQL

下载完成, 点击安装:





一直继续：



注意：安装时会弹出一个提示框，提示框中有临时密码，必须记住，后面会用到，界面上的文字可以复制。我的临时密码为：**go\_W<y8%ULvO**。

安装完成后，Mac 系统偏好设置会有：





## 配置环境变量

MySQL 安装的路径为：/usr/local/mysql，因此在 .bash\_profile 文件加入：

```
## MySQL
export PATH=${PATH}:/usr/local/mysql/bin
```

## 开启服务

点击系统偏好设置 - MySQL，开启服务：



## 连接 MySQL

命令行：

```
mysql -u root -p
```

这时候需要输入安装时的临时密码：go\_W<y8%ULvO。

## 修改 root 密码

命令行：

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'root';
```

提示 Query OK, 0 rows affected (0.00 sec)，这里我修改新密码为“root”了，然后重新开启 MySQL 服务。

## 退出 MySQL

三个命令作用相同：

- exit
- quit
- \q

## 可视化工具

1、MySQL Workbench

不知道原来官方已经提供了 Mac 版工具了。

## 2、NaviCat Premium

很熟悉的软件了，不过 Mac 版是收费的。

## 卸载 MySQL

参考 stackoverflow: [How do you uninstall MySQL from Mac OS X?](#)

命令行:

```
sudo rm -rf /var/db/receipts/com.mysql.*
```

## 参考

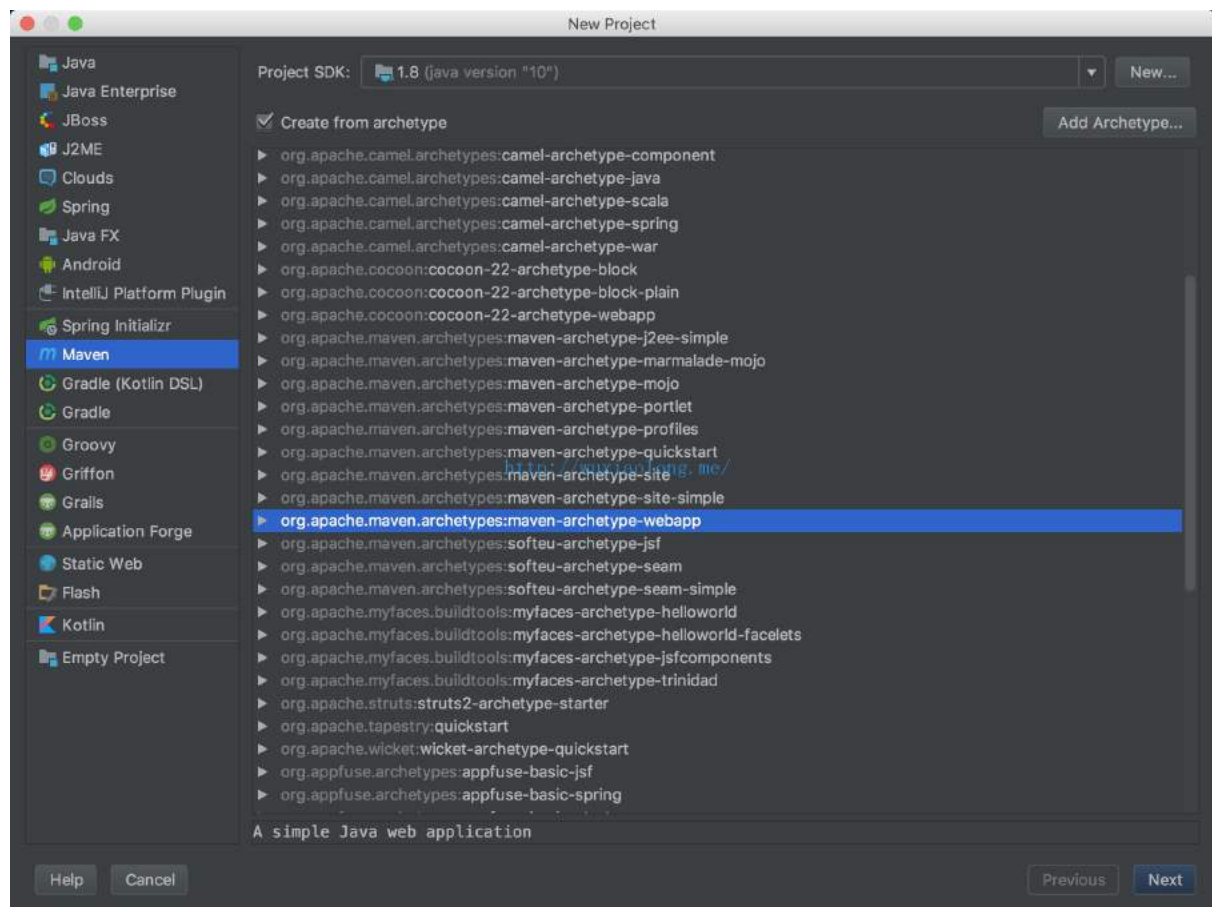
[Mac下安装mysql服务及基于workbench的使用方法](#)

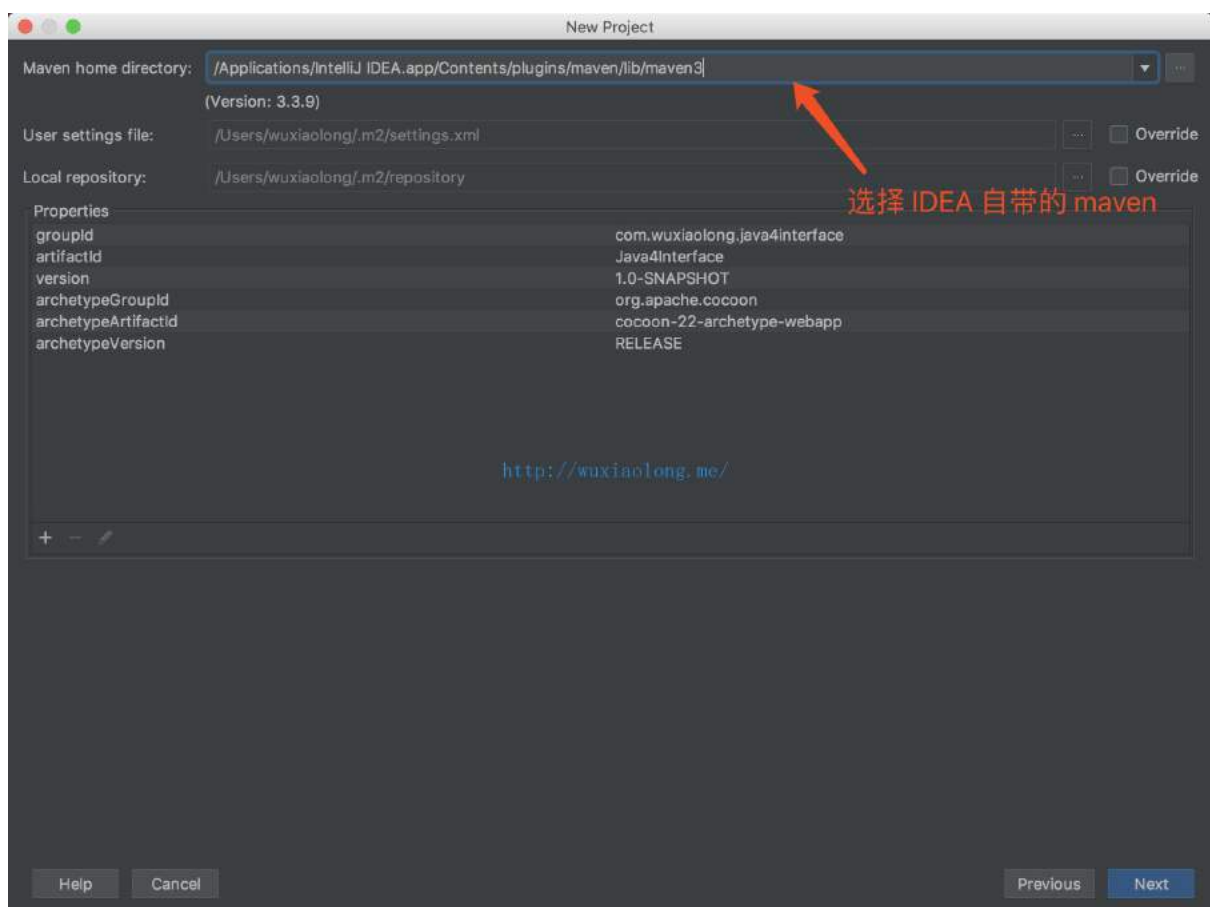
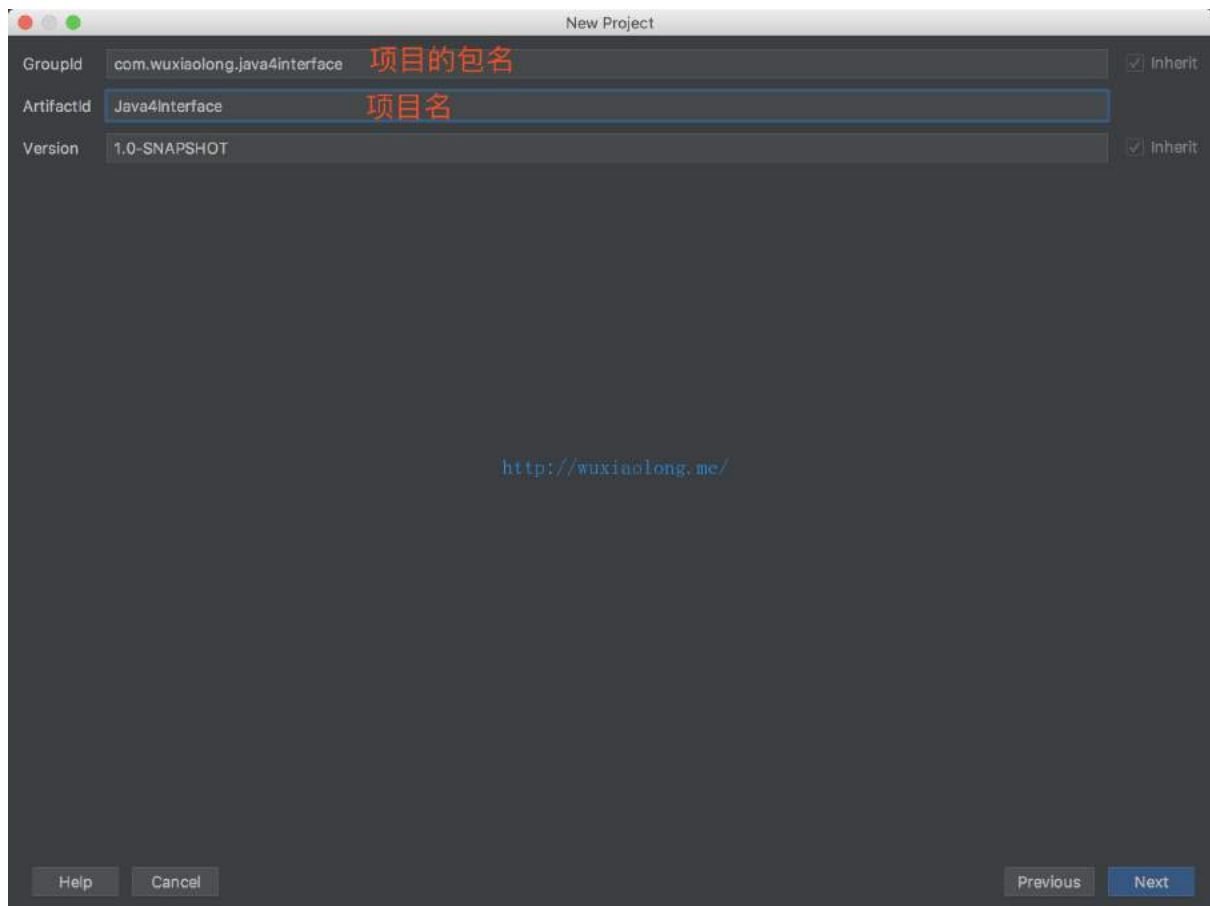
# Hello World

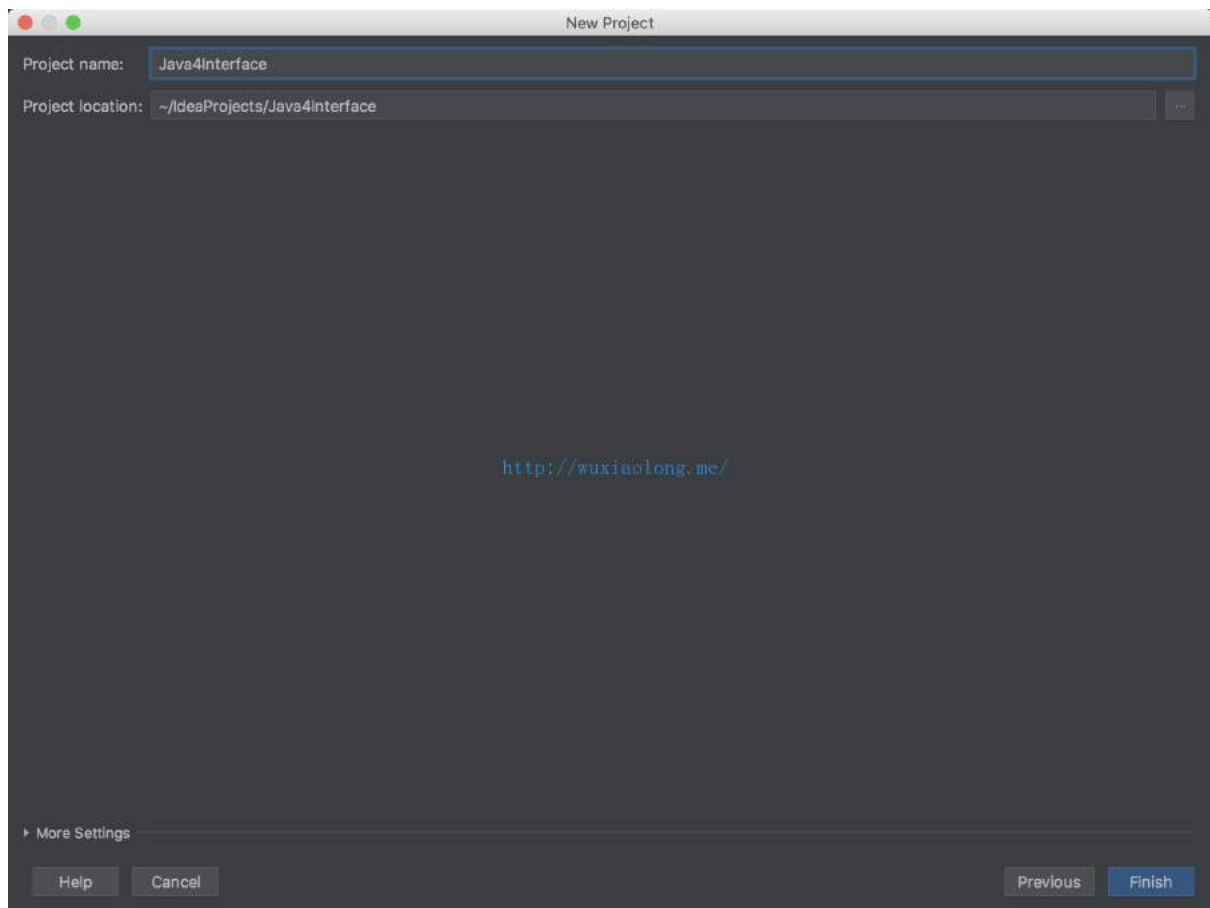
Java 开发环境已经搭建完成，我们来建个项目，开搞起来了。首先要 IntelliJ IDEA 创建 Maven 项目，Maven 构建和 Gradle 是类似的。IntelliJ IDEA 已经集成 Maven 插件了，不用我们再安装 Maven 了。

## 创建项目

打开 IDEA，点击 File - New -Project...

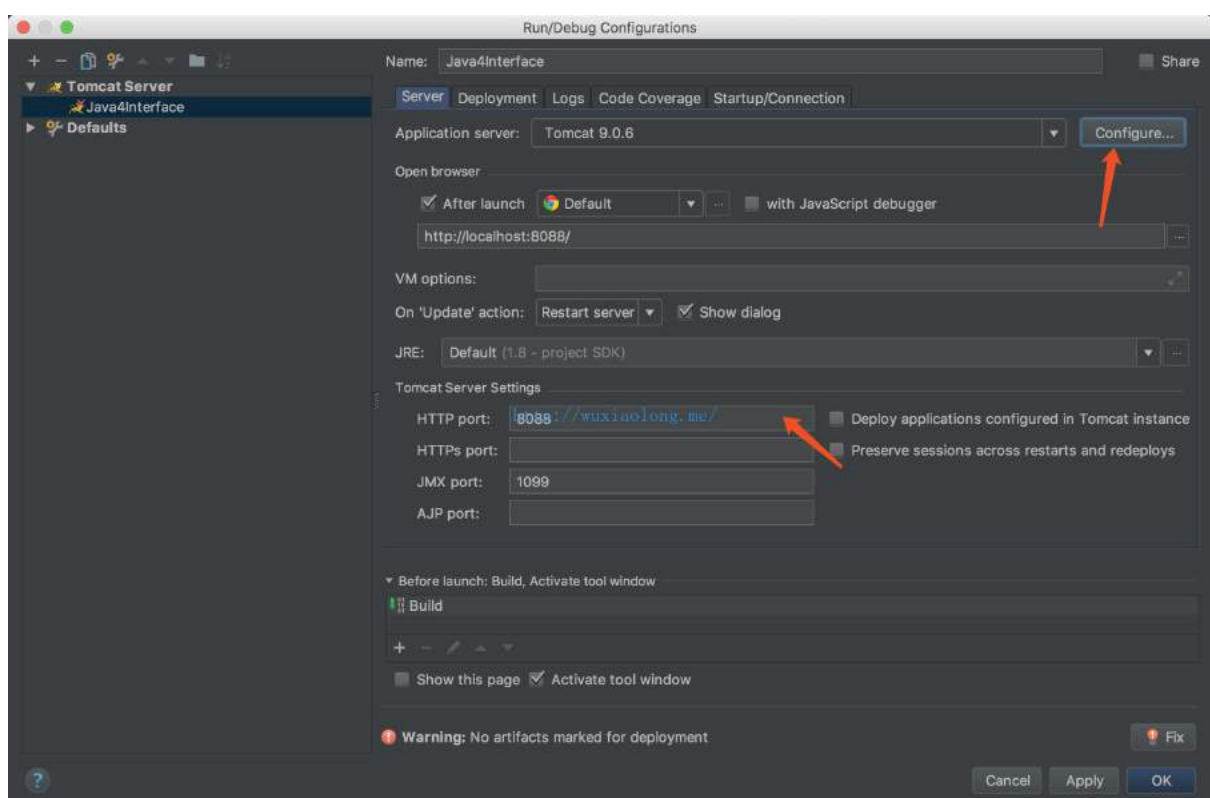
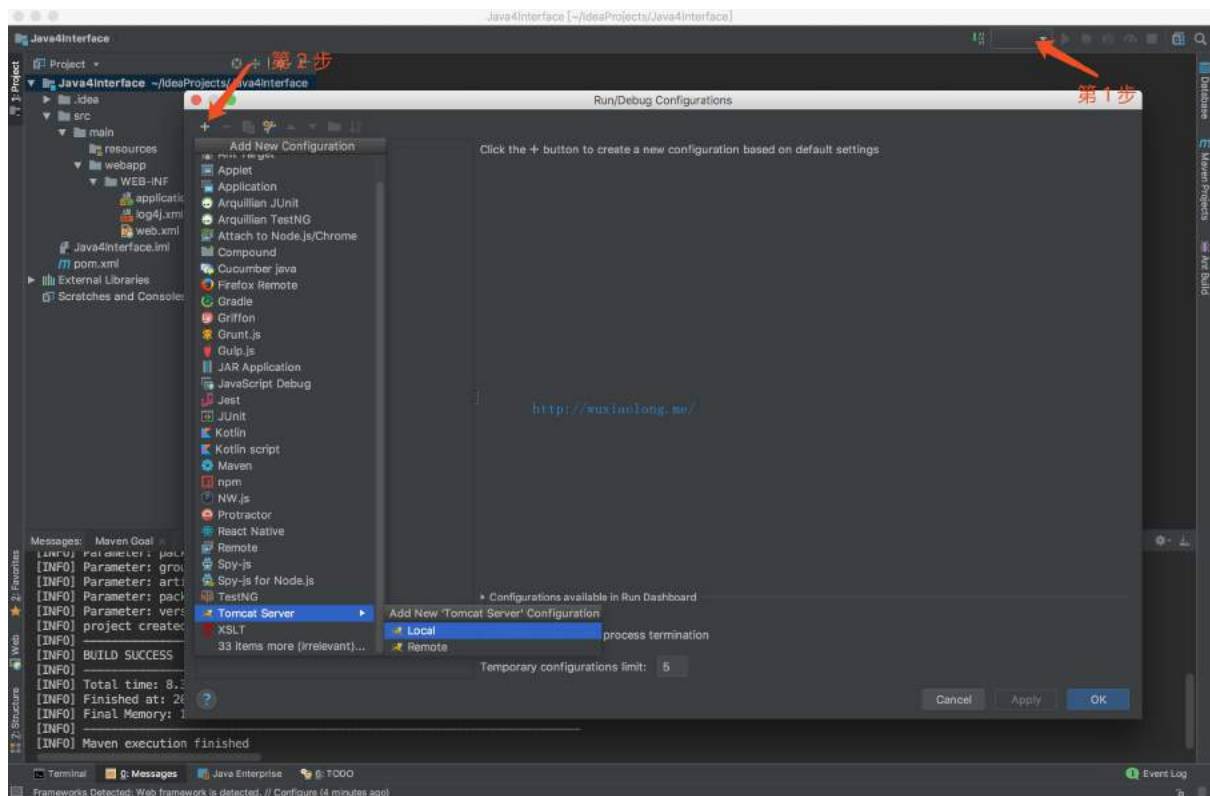




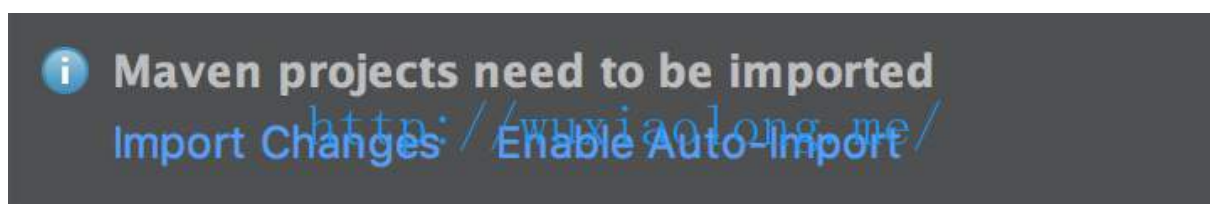
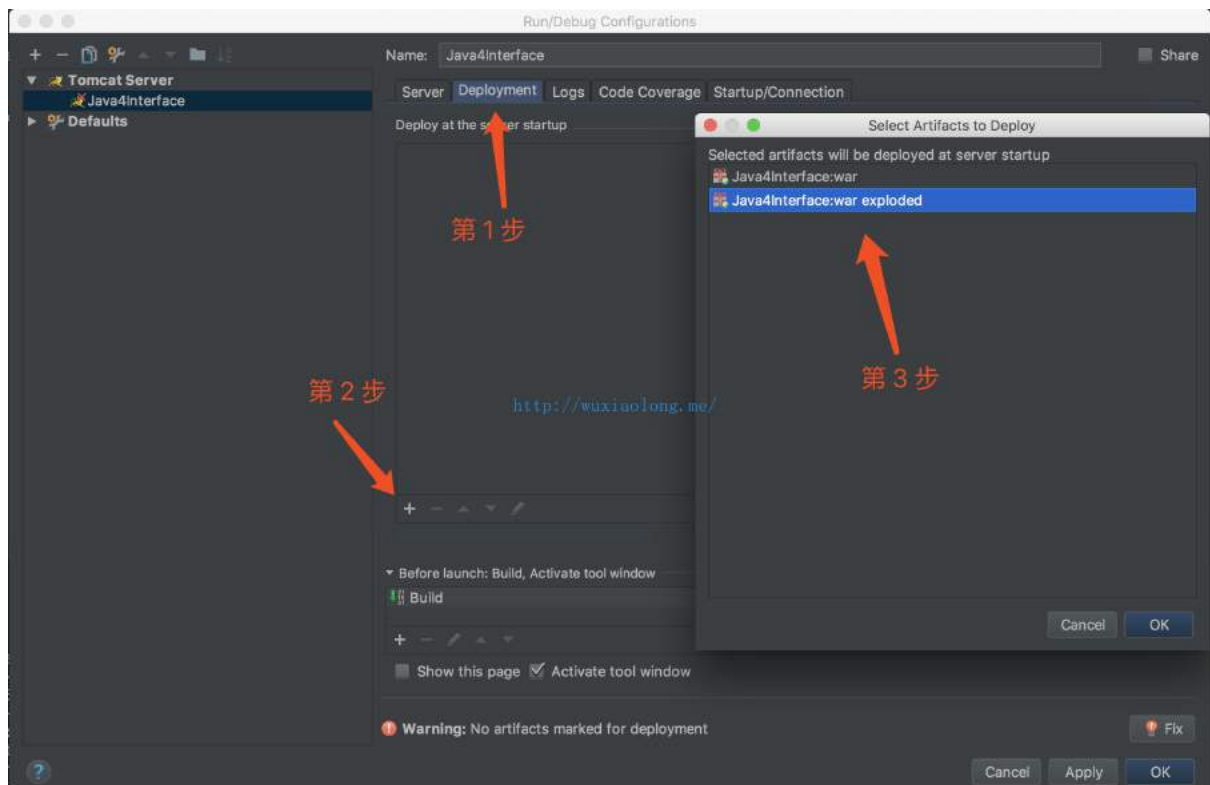


点击 Finish，这时候项目创建完毕，但是还不能运行，得配置 Tomcat。

## 配置 Tomcat



这里端口我改成了 8088，防止默认端口 8080 被占用了；点击 Configure，可以配置 Tomcat，指定到 Tomcat 路径，默认就是 Tomcat 安装路径。

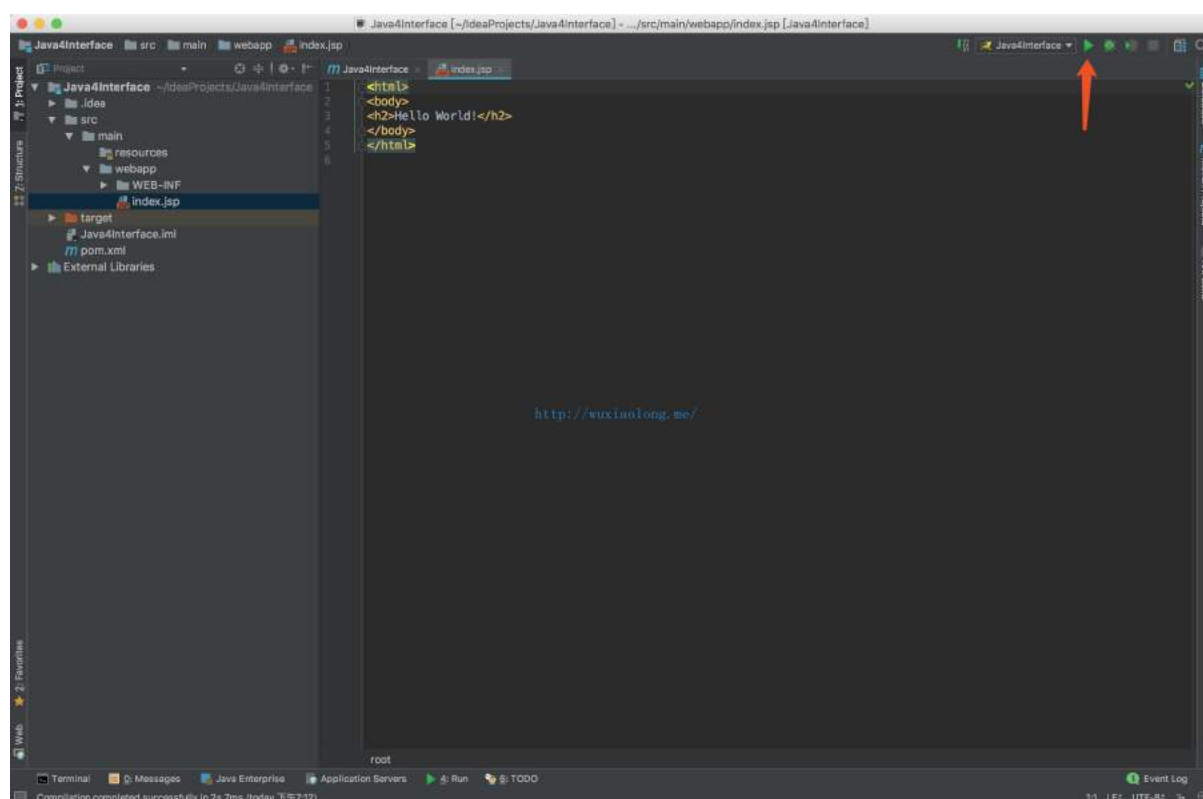


选择 Enable Auto import, 是为了在 pom.xml 文件中添加依赖之后自动引入 jar。

这样 Tomcat 配置完成, 运行项目就能看到 Hello World! 了。

## 运行项目





点击运行，会自动弹出一个页面，效果如下：



## 踩坑记录

- 1、为什么我的 IDEA 没有 Tomcat 配置呢？低版本 IDEA，插件里面 Tomcat 默认关闭了，进入 File - Settings - Plugins，搜索 Tomcat 插件，开启就行了。
- 2、添加 Configuration 时找不到 Tomcat Server？解决方法：自己配置一下路径添加即可，步骤：IntelliJ IDEA -> Preferences -> Application Servers。
- 3、配置 Deployment 时找不到 Artifact？解决方法：IntelliJ IDEA -> Preferences -> Importing -> 勾选 Import Maven projects automatically，保存即可。
- 4、创建 Servlet 时，右击 java - New - Servlet，没有 Servlet 选项？我估计他 Enable Auto import 没有勾上，加了 Servlet 依赖库没有更新，IDEA 没有识别出 Servlet，右击自然没有 Servlet 选项。如果没有勾上，需要手动刷新下 Maven。还有点击 Auto 后也会有不出现创建 Servlet 的情况，需要打开 project setting 界面 modules 中添加服务器的 jar 包。



## Servlet

Servlet (Server Applet) , 全称 Java Servlet, 未有中文译文。是用 Java 编写的服务器端程序。其主要功能在于交互式地浏览和修改数据, 生成动态 Web 内容。狭义的 Servlet 是指 Java 语言实现的一个接口, 广义的 Servlet 是指任何实现了这个 Servlet 接口的类, 一般情况下, 人们将Servlet 理解为后者。

Servlet 运行于支持 Java 的应用服务器中。从实现上讲, Servlet 可以响应任何类型的请求, 但绝大多数情况下 Servlet 只用来扩展基于 HTTP 协议的 Web 服务器。

## 工作模式

- 客户端发送请求至服务器;
- 服务器启动并调用 Servlet, Servlet 根据客户端请求生成响应内容并将其传给服务器;
- 服务器将响应返回客户端。

以上内容来自维基百科。

## 创建 Servlet

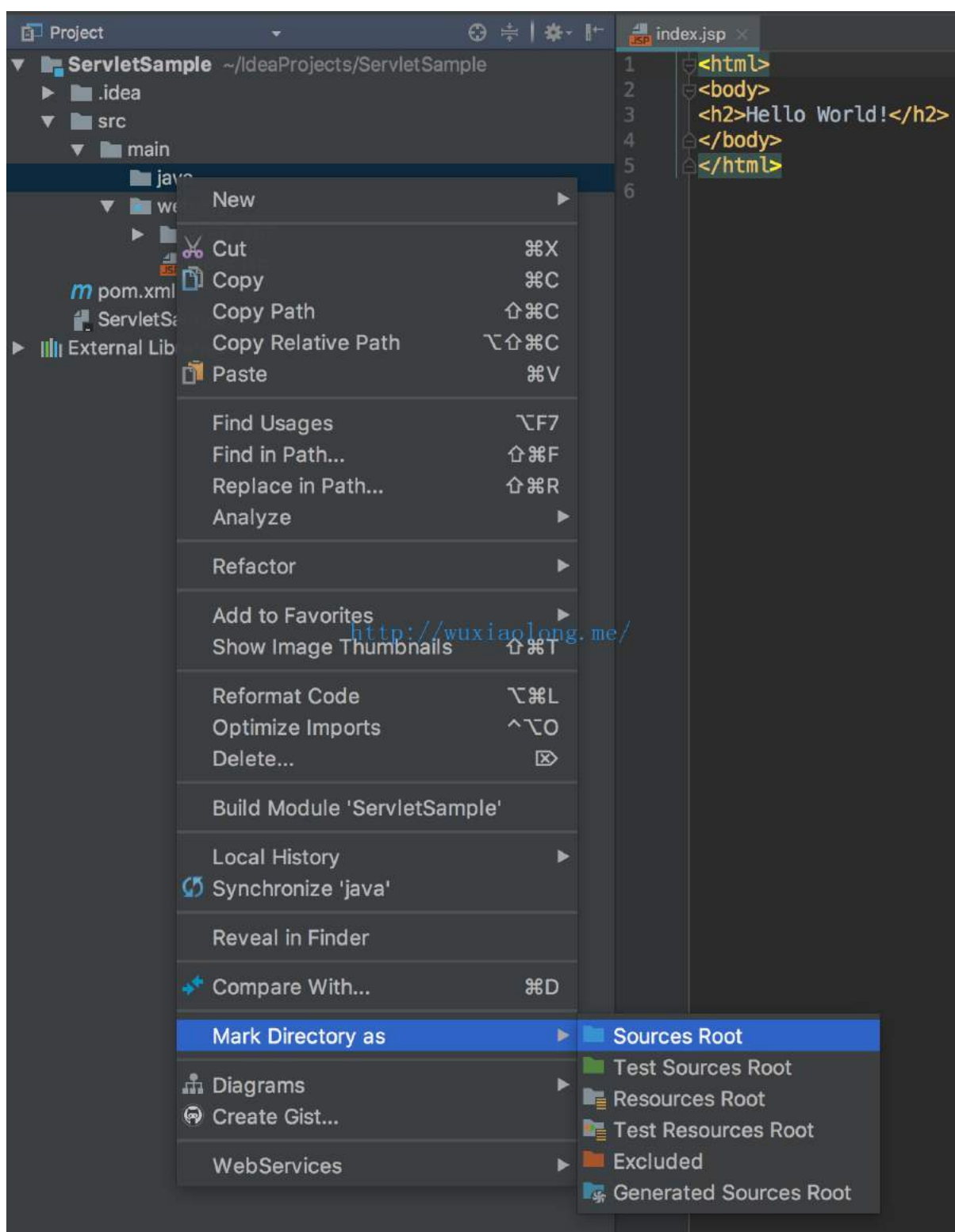
### pom.xml

添加 Servlet Maven 依赖：

```
<!--servlet-->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.0.1</version>
</dependency>
```

## 创建 Servlet

新建 java 目录，标记为 Sources Root，同样 resources 目录，也是如此设置。



右击 java - New - Servlet，输入 DeveloperServlet，代码如下：

```
//新建 DeveloperServlet，自动生成 @WebServlet(name = "DeveloperServlet")
@WebServlet(name = "DeveloperServlet", urlPatterns = {"/DeveloperServlet"})
public class DeveloperServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {

    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        //设置响应内容类型
        response.setContentType("text/html");
        //设置逻辑实现
        PrintWriter out = response.getWriter();
        out.println("<h3>Hello world, this message is from servlet!!</h3>");
    }
}
```

## 部署 Servlet

### 1、注解方式

新建 DeveloperServlet，自动生成 `@WebServlet(name = "DeveloperServlet")`，再加个

`urlPatterns = {"/DeveloperServlet"}` 运行起来，DeveloperServlet 访问地址就是：<http://localhost:8088/DeveloperServlet>

### 2、xml 方式

web.xml 加入：

```
<web-app>
    <display-name>Archetype Created Web Application</display-name>

    <!--部署 DeveloperServlet-->
    <servlet>
        <servlet-name>DeveloperServlet</servlet-name>
        <servlet-class>DeveloperServlet</servlet-class>
    </servlet>
    <!--配置 servlet-mapping-->
    <servlet-mapping>
        <servlet-name>DeveloperServlet</servlet-name>
        <url-pattern>/DeveloperServlet</url-pattern>
    </servlet-mapping>

</web-app>
```

如果两种方式都配了，xml 优先。

## 运行 Servlet

点击运行，效果如下：



这样创建 Servlet 项目搞定了，接下来，就是进入 Servlet 写接口阶段，数据从数据库取，返回数据是 json 格式的。

## 创建数据库

在写接口前，本地先要创建数据库。

## 开启 MySQL 服务

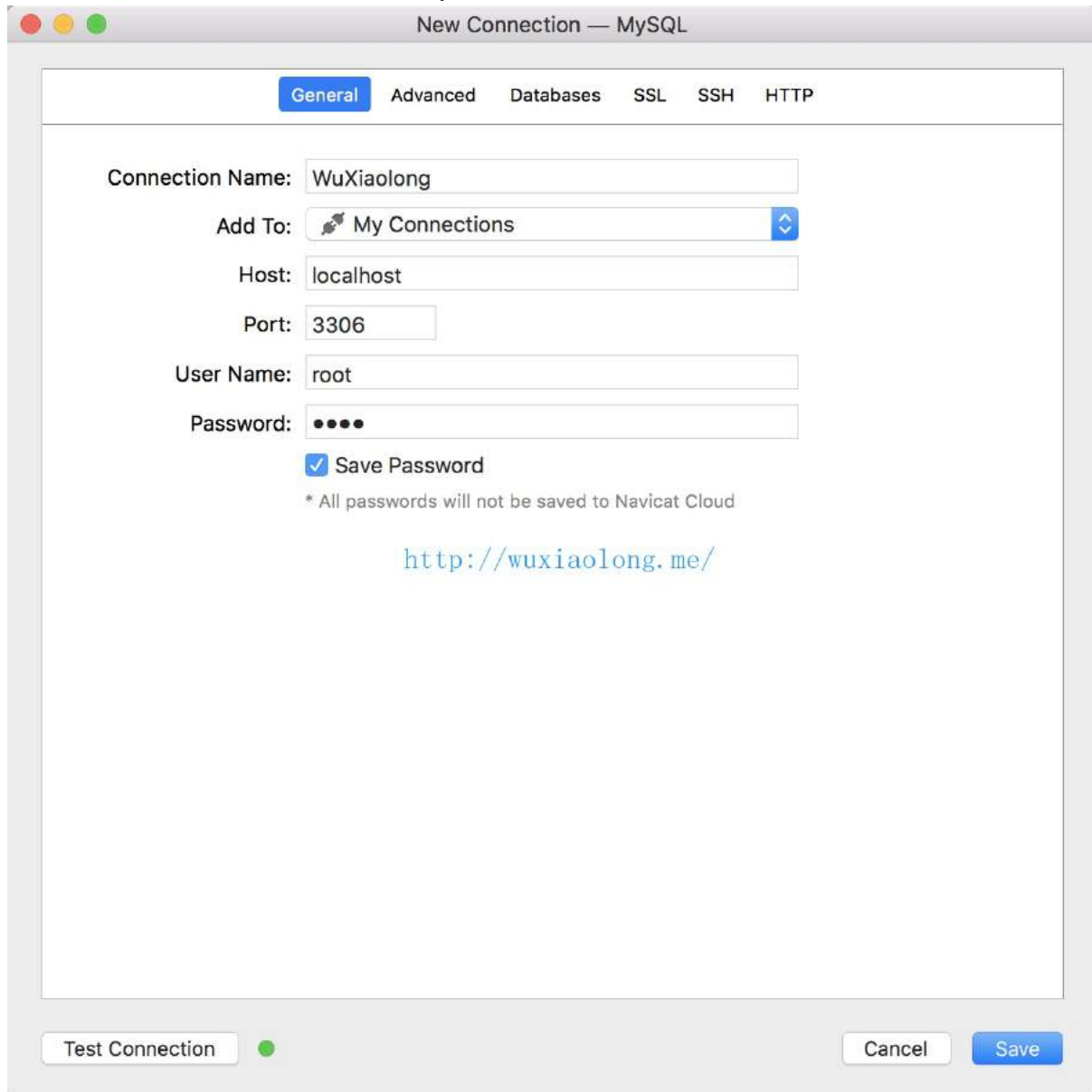
点击系统偏好设置 - MySQL，开启服务：



## 连接本地数据库



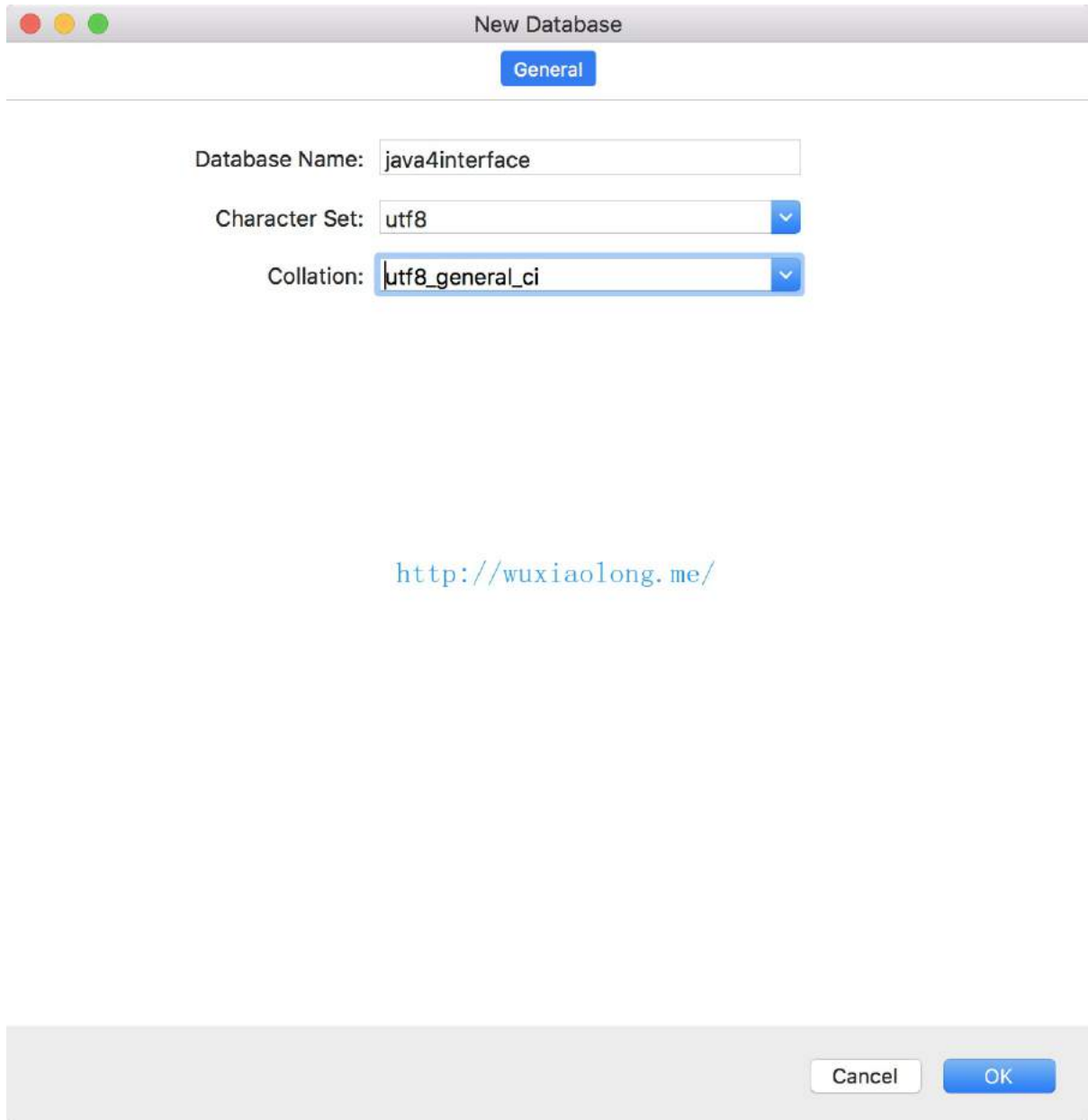
打开 Navicat Premium - Connection - MySQL:



填下 Name 和 Password, 可以点击下 Test Connection, OK, 再 Save。

## 创建数据库

双击 WuXiaolong, Open Connect, 然后右击 - New Database:



New Database

General

Database Name: java4interface

Character Set: utf8

Collation: utf8\_general\_ci

<http://wuxiaolong.me/>

Cancel OK

## 创建表

NaviCat Premium 创建，右击 Tables - 填写字段相关信息，保存填写表名 developer，我这里加了四个字段，先简单些，id 设置主键，自增：

The screenshot shows a database creation interface with several tabs: Fields, Indexes, Foreign Keys, Triggers, Options, Comment, and SQL Preview. The 'Fields' tab is active, displaying a table with the following columns:

Name	Type	Length	Decimals	Not Null	Virtual	Key	Comment
id	bigint	10	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
name	varchar	10	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
site	varchar	20	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
avatar	varchar	30	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Below the table, there is a text input field containing the URL `http://wuxiaolong.me/`. A red arrow points from this field to the 'Default Value' dropdown menu, which is currently set to 'EMPTY STRING'. Other configuration options include 'Character Set' set to 'utf8' and 'Collation' set to 'utf8\_general\_ci'. There is also an unchecked 'Binary' checkbox.

也可以代码创建，这里略。这样数据创建也完成了。

## 踩坑说明

- 1、如上图中箭头，这里可以给个默认值 EMPTY STRING，不然如果没有值，到时接口返回时，这条数据该字段不会返回，如果想返回 `name: ""`，得代码判断，麻烦。
- 2、设计数据库时，id 设置主键，并设置 Auto increment 自增，不然只能添加第一条，再次增加就返回 failed 了。

## Servlet 写接口

### 连接数据库库

我用的是 mysql-connector-java 这个库，pom.xml 加入 Maven 依赖：

```
<!--mysql-connector-java-->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.46</version>
</dependency>
```

### DBHelper

连接数据库以及关闭连接：

```
public class DBHelper {
    //java4interface 创建数据库的名字
    private static final String url = "jdbc:mysql://localhost:3306/java4interface?useUnicode=true&characterEncoding=UTF-8";
    private static final String name = "com.mysql.jdbc.Driver";
    //连接数据库账号
    private static final String user = "root";
    //连接数据库密码
    private static final String password = "root";

    private Connection connection = null;
    public PreparedStatement preparedStatement = null;

    public DBHelper(String sql) {
        try {
            Class.forName(name);
            connection = DriverManager.getConnection(url, user, password);
            preparedStatement = connection.prepareStatement(sql);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void close() {
        try {
            this.connection.close();
            this.preparedStatement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
    }  
    }  
}
```

## DeveloperModel

和表对应，代表数据库中的模型：

```
public class DeveloperModel {  
    private int id;  
    private String name;  
    private String site;  
    private String avatar = "";  
  
    //省略 get、set 方法  
  
}
```

## DeveloperBusiness

进行数据库有关的业务逻辑：

```
public class DeveloperBusiness {  
    public List<DeveloperModel> getAllDevelopers() {  
        List<DeveloperModel> developerList = new ArrayList<>();  
        String sql = "select * from developer";  
        DBHelper dbHelper = new DBHelper(sql);  
        ResultSet resultSet = null;  
        try {  
            resultSet = dbHelper.prepareStatement.executeQuery();  
            while (resultSet.next()) {  
                int id = resultSet.getInt(1);  
                String name = resultSet.getString(2);  
                String site = resultSet.getString(3);  
                String avatar = resultSet.getString(4);  
                DeveloperModel developerModel = new DeveloperModel();  
                developerModel.setId(id);  
                developerModel.setName(name);  
                developerModel.setSite(site);  
                developerModel.setAvatar(avatar);  
                developerList.add(developerModel);  
            }  
            resultSet.close();  
            dbHelper.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
        return developerList;
    }

    public DeveloperModel getDeveloper(String developerId) {
        String sql = "select * from developer where id=" + developerId;
        DBHelper dbHelper = new DBHelper(sql);
        DeveloperModel developerModel = null;
        try {
            ResultSet resultSet = dbHelper.prepareStatement.executeQuery();
            while (resultSet.next()) {
                int id = resultSet.getInt(1);
                String name = resultSet.getString(2);
                String site = resultSet.getString(3);
                String avatar = resultSet.getString(4);
                System.out.println("avatar=" + avatar);
                developerModel = new DeveloperModel();
                developerModel.setId(id);
                developerModel.setName(name);
                developerModel.setSite(site);
                developerModel.setAvatar(avatar);
            }
            resultSet.close();
            dbHelper.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return developerModel;
    }

    public boolean addDeveloper(DeveloperModel developerModel) {
        String sql = "INSERT INTO developer(name,site,avatar) VALUES(" +
            "'" + developerModel.getName() + "'," +
            "'" + developerModel.getSite() + "'," +
            "'" + developerModel.getAvatar() + "'" + ")";
        System.out.println("sql=" + sql);
        DBHelper dbHelper = new DBHelper(sql);
        return execute(dbHelper);
    }

    private boolean execute(DBHelper dbHelper) {
        try {
            dbHelper.prepareStatement.execute();
            dbHelper.close();
            return true;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }
}
```

```

    }

    public boolean updateDeveloper(String id, String name) {
        String sql = "UPDATE developer SET name='" + name + "' WHERE id=" + id;
        System.out.println("sql=" + sql);
        DBHelper dbHelper = new DBHelper(sql);
        try {
            dbHelper.prepareStatement().executeUpdate();
            dbHelper.close();
            return true;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }

    public boolean deleteDeveloper(String id) {
        String sql = "DELETE FROM developer WHERE id=" + id;
        System.out.println("sql=" + sql);
        DBHelper dbHelper = new DBHelper(sql);
        return execute(dbHelper);
    }
}

```

## CommonModel

封装了状态码和状态信息：

```

public class CommonModel {
    //code, 0: 接口请求成功, 有数据返回; 1: 接口请求成功, 无数据返回。
    private int code;
    //msg
    private String msg;
    //可以单个对象或List
    private Object data;

    //省略 get、set 方法

    public void setSuccess() {
        setCode(ConstantUtil.CODE_SUCCESS);
        setMsg(ConstantUtil.MSG_SUCCESS);
    }

    public void setFail() {
        setCode(ConstantUtil.CODE_FAIL);
        setMsg(ConstantUtil.MSG_FAIL);
    }
}

```

```
}  
}
```

## Json 解析库

我用的是 Gson 这个库，pom.xml 加入 Maven 依赖：

```
<!--Gson-->  
<dependency>  
  <groupId>com.google.code.gson</groupId>  
  <artifactId>gson</artifactId>  
  <version>2.8.2</version>
```

网上说阿里的 Json 解析库 FastJson 也很不错，因为 Android 我一直用的 Gson，先玩下 Gson。

## GsonUtil

写成工具类，方便调用：

```
public class GsonUtil {  
    private static Gson gson = new GsonBuilder().create();  
  
    public static String bean2Json(Object obj) {  
        return gson.toJson(obj);  
    }  
  
    public static <T> T json2Bean(String jsonStr, Class<T> objClass) {  
        return gson.fromJson(jsonStr, objClass);  
    }  
  
    public static String jsonFormatter(String uglyJsonStr) {  
        Gson gson = new GsonBuilder().setPrettyPrinting().create();  
        JsonParser jp = new JsonParser();  
        JsonElement je = jp.parse(uglyJsonStr);  
        return gson.toJson(je);  
    }  
}
```

## DeveloperServlet

```
@WebServlet(name = "DeveloperServlet", urlPatterns = {ConstantUtil.ALL_DEVELOPERS_U  
RL, ConstantUtil.QUERY_DEVELOPER_URL,  
    ConstantUtil.ADD_DEVELOPER_URL, ConstantUtil.UPDATE_DEVELOPER_URL, Constant  
Util.DELETE_DEVELOPER_URL})  
public class DeveloperServlet extends HttpServlet {
```



```

DeveloperBusiness developerBusiness = new DeveloperBusiness();

protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    this.doGet(request, response);
}

@Override
protected void doDelete(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    //this.doGet(request, response);
    //Servlet 貌似不支持 Delete, 传参始终为 null
}

protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    System.out.println("url=" + request.getRequestURI());
    request.setCharacterEncoding("UTF-8");
    //设置响应内容类型
    response.setContentType("text/json;charset=UTF-8");
    String url = request.getRequestURI();
    if (url.equals(ConstantUtil.ALL_DEVELOPERS_URL)) {
        getAllDevelopers(request, response);
    } else if (url.equals(ConstantUtil.QUERY_DEVELOPER_URL)) {
        getDeveloper(request, response);
    } else if (url.equals(ConstantUtil.ADD_DEVELOPER_URL)) {
        addDeveloper(request, response);
    } else if (url.equals(ConstantUtil.UPDATE_DEVELOPER_URL)) {
        updateDeveloper(request, response);
    } else if (url.equals(ConstantUtil.DELETE_DEVELOPER_URL)) {
        deleteDeveloper(request, response);
    }
}

private void getAllDevelopers(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    //设置逻辑实现
    PrintWriter printWriter = response.getWriter();
    List<DeveloperModel> developerList = developerBusiness.getAllDevelopers();
    CommonModel commonModel = new CommonModel();
    if (developerList.size() > 0) {
        commonModel.setSuccess();
        commonModel.setObject(developerList);
    } else {
        commonModel.setFail();
    }
    printWriter.println(GsonUtil.bean2Json(commonModel));
    printWriter.flush();
}

```

```
        printWriter.close();
    }

    private void getDeveloper(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //设置逻辑实现
        PrintWriter printWriter = response.getWriter();
        // 接受参数
        String id = request.getParameter("id");
        System.out.println("id=" + id);
        DeveloperModel developerModel = developerBusiness.getDeveloper(id);
        CommonModel commonModel = new CommonModel();
        if (developerModel == null) {
            commonModel.setFail();
        } else {
            commonModel.setSuccess();
            commonModel.setObject(developerModel);
        }
        printWriter.println(GsonUtil.bean2Json(commonModel));
        printWriter.flush();
        printWriter.close();
    }

    private void addDeveloper(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        //设置逻辑实现
        PrintWriter printWriter = response.getWriter();
        String name = request.getParameter("name");
        System.out.println("name=" + name);
        String site = request.getParameter("site");
        String avatar = request.getParameter("avatar");

        CommonModel commonModel = new CommonModel();
        DeveloperModel developerModel = new DeveloperModel();
        developerModel.setName(name);
        developerModel.setSite(site);
        developerModel.setAvatar(avatar);
        if (developerBusiness.addDeveloper(developerModel)) {
            commonModel.setSuccess();
        } else {
            commonModel.setFail();
        }
        printWriter.println(GsonUtil.bean2Json(commonModel));
    }

    private void updateDeveloper(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //设置逻辑实现
```

```
        PrintWriter printWriter = response.getWriter();
        String id = request.getParameter("id");
        String name = request.getParameter("name");
        System.out.println("name=" + name);

        CommonModel commonModel = new CommonModel();
        if (developerBusiness.updateDeveloper(id, name)) {
            commonModel.setSuccess();
        } else {
            commonModel.setFail();
        }
        printWriter.println(GsonUtil.bean2Json(commonModel));
    }
}
```

## ConstantUtil

```
public class ConstantUtil {
    public static final int CODE_SUCCESS = 0;
    public static final int CODE_FAIL = 1;
    public static final String MSG_SUCCESS = "success";
    public static final String MSG_FAIL = "fail";
    public static final String ALL_DEVELOPERS_URL = "/developer/api/developers";
    public static final String QUERY_DEVELOPER_URL = "/developer/api/developer";
    public static final String ADD_DEVELOPER_URL = "/developer/api/developer/add";
    public static final String UPDATE_DEVELOPER_URL = "/developer/api/developer/update";
    public static final String DELETE_DEVELOPER_URL = "/developer/api/developer/delete";
}
```

分别实现 Http Get、Post、Update 请求方式，即数据库基本的增（删）改查，Servlet 貌似不支持 Delete，用 Postman 测试，能返回自己想要的数据了，大功告成，入门完成。

## 源码

公众号「吴小龙同学」回复：ServletSample，获取完整的 Sample 项目代码。

## 传参

### 接受参数

比如我请求的地址：<http://localhost:8088/DeveloperServlet?id=1>，如果拿到这个 id 值？

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
    // 接受参数  
    String id = request.getParameter("id");  
}
```

### 初始化参数

@WebServlet 和 web.xml 都能初始化参数。

### 设置

1、注解 @WebServlet 的 initParams 参数来指定的：

```
@WebServlet(name = "DeveloperServlet", urlPatterns = {"/DeveloperServlet", "/DeveloperServlet/avatar"}, initParams = {@WebInitParam(name = "id", value = "1")})
```

2、web.xml

```
<servlet>  
    <servlet-name>DeveloperServlet</servlet-name>  
    <servlet-class>servlet.DeveloperServlet</servlet-class>  
    <!--Servlet init param -->  
    <init-param>  
        <param-name>id</param-name>  
        <param-value>2</param-value>  
    </init-param>  
</servlet>
```

### 取值

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
    //初始化参数  
    String = this.getServletConfig().getInitParameter("id");  
}
```

## 踩坑记录

Postman 测试 Post 请求，需要选择 body - x-www-form-urlencoded，选择 form-data，`request.getParameter("id")` 始终是 null，浏览器的原生 form 表单，如果不设置 enctype 属性，那么最终就会以 application/x-www-form-urlencoded 方式提交数据，提交的数据按照 `key1=val1&key2=val2` 的方式进行编码。multipart/form-data，可以传文件，我平时都是用的这个。

# Spring

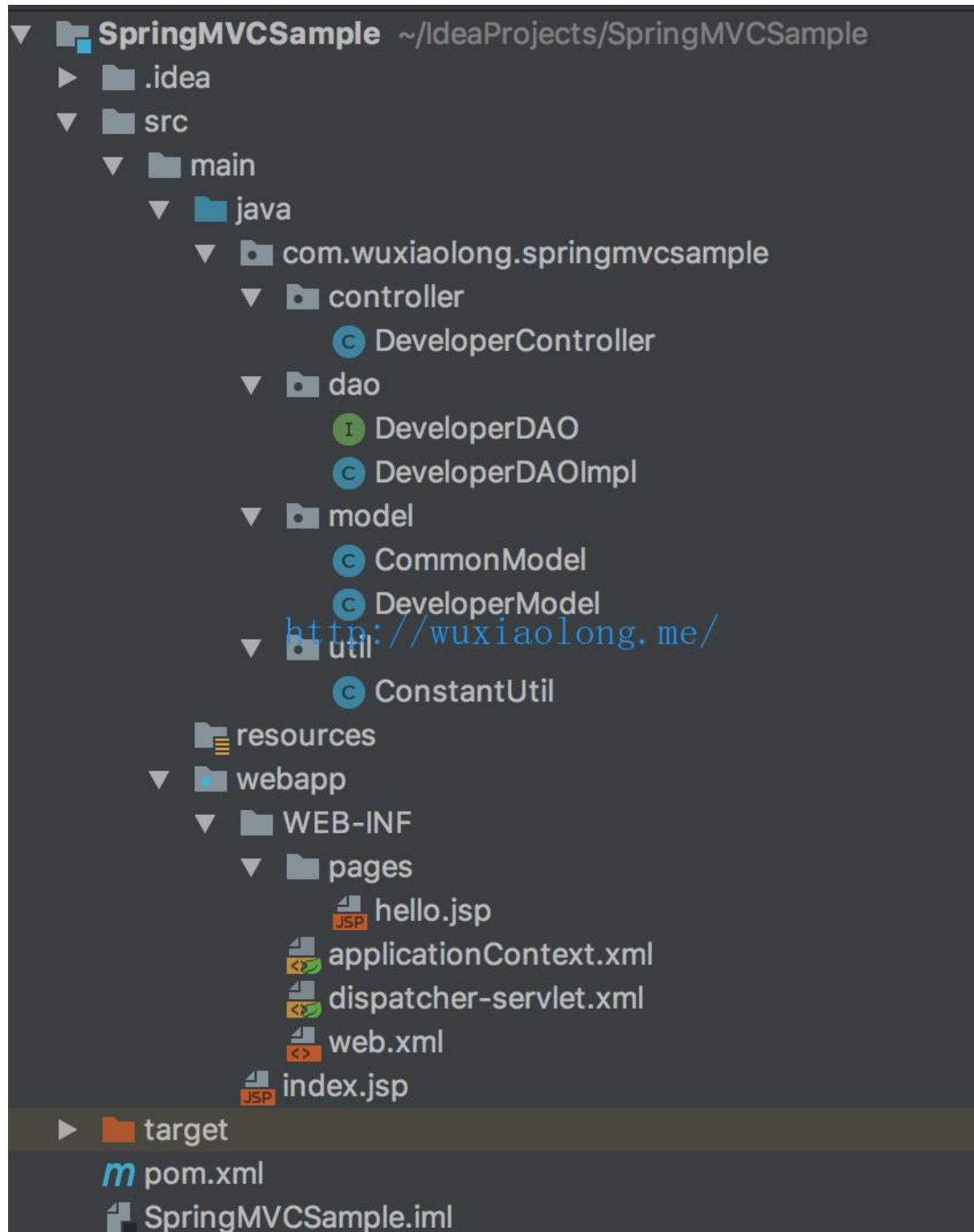
开源的 Java / Java EE 全功能栈（full-stack）的应用程序框架。

## Spring MVC

基于 Servlet 的一个 MVC 框架，解决 WEB 开发的问题，MVC 框架有 model、view、controller 三部分组成。model 是一些基本的 Java Bean，view 用于进行相应的页面显示，controller 用于处理网站的请求。因为我们只是完成接口开发，因此 view 页面显示将不涉及。

## 创建 Spring MVC

还是创建 Maven 项目，然后配置 Tomcat，前面有详细介绍，这里不再累述。先给个项目目录，包括后面的数据库交互：



- controller：用于处理网站的请求；
- dao：数据访问层（接口）；
- model：数据模型；

接下来，先实现 Spring MVC。



## pom.xml

添加 Spring Maven 依赖：

```
<properties>
  <!-- 这里省略部分代码 -->
  <spring.version>5.0.5.RELEASE</spring.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <!-- 依赖注入，事件，资源，i18n，验证，数据绑定，类型转换，SpEL，AOP。 -->
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <!-- Spring MVC -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
  </dependency>
</dependencies>
```

## Spring MVC 配置

按照上面项目目录，新建 hello.jsp、applicationContext.xml、dispatcher-servlet.xml、controller 包，其中：

hello.jsp

```
<html>
<body>
<h2>Hello, Spring</h2>
</body>
</html>
```

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

```

springframework.org/schema/beans/spring-beans.xsd">

</beans>

```

这个阶段不用配置什么。

dispatcher-servlet.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"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.s
pringframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/sch
ema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd http://www.springf
ramework.org/schema/context http://www.springframework.org/schema/context/spring-co
ntext.xsd">

    <!--此文件负责整个 MVC 中的配置-->

    <!--对 base-package 包或者子包下的所有的进行 Java 类进行扫描，并把匹配的 Java，类注册成 b
ean-->
    <context:component-scan base-package="com.wuxiaolong.springmvcsample.controller"
/>

    <!-- 定义一个视图解析器，用于支持 Servlet、JSP 路径解析-->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

        <!--/WEB-INF/pages/ 是 jsp 目录，prefix: 前缀-->
        <property name="prefix" value="/WEB-INF/pages/" />
        <!--suffix: 后缀，以 jsp 结尾-->
        <property name="suffix" value=".jsp" />
    </bean>

    <!--开启注解-->
    <mvc:annotation-driven/>

</beans>

```

web.xml

```

<web-app>
    <display-name>Archetype Created Web Application</display-name>
    <!--Spring MVC 配置-->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>

```

```
</context-param>

<!-- 注册ServletContext监听器，创建容器对象，并且将ApplicationContext对象放到Application
域中 -->
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener
-class>
</listener>

<!--在 web.xml 同级目录下将有个 dispatcher-servlet.xml 配置文件进行对应，负责整个 mvc 的
配置-->
<!--applicationContext.xml 和 dispatcher-servlet.xml 区别: http://www.cnblogs.com/
parryyang/p/5783399.html-->
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```

## DeveloperController

```
@Controller
@RequestMapping("/developer")
public class DeveloperController {

    @RequestMapping(value = "/api/hello", method = RequestMethod.GET)
    public String hello() {
        return "hello";
    }

}
```

按照之前的 Servlet，我以为浏览器打开地址：<http://localhost:8088/developer/api/hello>，会显示 hello，其实不是，这里默认会去找 hello.jsp 页面，就是前面已经创建好的 hello.jsp。



`/developer/api/hello`，对应 `RequestMapping` 值，`method` 就是 `Http` 请求方式，这样 `Spring MVC` 配置就此完成，第一步搞定，接下来就是接口部分开发，向着目标前进。

# Spring MVC 写接口

## 数据库

数据库还是用前面的，如何创建数据库，前面有详细介绍，这里也不再累述了。

## pom.xml

添加 Spring Maven 依赖：

```
<!--Spring 数据库-->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring.version}</version>
</dependency>
<!--JDBC-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.34</version>
</dependency>
<!--Gson-->
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.2</version>
</dependency>
```

## Spring MVC 配置

applicationContext.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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.s
pringframework.org/schema/beans/spring-beans.xsd">

    <!--配置数据源-->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerD
ataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url"
            value="jdbc:mysql://localhost:3306/java4interface?useUnicode=true
```

```

&characterEncoding=UTF-8"/>
    <property name="username" value="root"/>
    <property name="password" value="root"/>
</bean>

<!--jdbc-->
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <constructor-arg ref="dataSource"/>
</bean>

</beans>

```

## dispatcher-servlet.xml

添加 model 和 dao 包的扫描:

```

<?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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.s
pringframework.org/schema/beans/spring-beans.xsd http://www.springframew
ork.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd http://www.springf
ramework.org/schema/context http://www.springframework.org/schema/context/spring-co
ntext.xsd">

    <!--此文件负责整个 MVC 中的配置-->

    <!--对 base-package 包或者子包下的所有的进行 Java 类进行扫描, 并把匹配的 Java, 类注册成 b
ean-->
    <context:component-scan base-package="com.wuxiaolong.springmvcsample.controller"
/>
    <context:component-scan base-package="com.wuxiaolong.springmvcsample.model"/>
    <context:component-scan base-package="com.wuxiaolong.springmvcsample.dao"/>

    <!-- 定义一个视图解析器, 用于支持 Servlet、JSP 路径解析-->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

        <!--/WEB-INF/pages/ 是 jsp 目录, prefix: 前缀-->
        <property name="prefix" value="/WEB-INF/pages/" />
        <!--suffix: 后缀, 以 jsp 结尾-->
        <property name="suffix" value=".jsp"/>
    </bean>

    <!--开启注解-->
    <mvc:annotation-driven/>

</beans>

```

## web.xml

加个字符过滤器，解决数据库插入中文乱码问题：

```
<!--<!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>

    <!--Spring MVC 配置-->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>

    <!-- 注册ServletContext监听器，创建容器对象，并且将ApplicationContext对象放到Application域中 -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <!--在 web.xml 同级目录下将有个 dispatcher-servlet.xml 配置文件进行对应，负责整个 mvc 的配置-->
    <!--applicationContext.xml 和 dispatcher-servlet.xml 区别: http://www.cnblogs.com/parryyang/p/5783399.html-->
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <!-- 字符过滤器 传值乱码-->
    <filter>
        <filter-name>encodingFilter</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>encodingFilter</filter-name>
```

```
<url-pattern>/*</url-pattern>
</filter-mapping>

</web-app>
```

加这个字符过滤器会报错：The content of element type "web-app" must match , 解决方案是将头部注掉。

```
<!--<!DOCTYPE web-app PUBLIC-->
<!--"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"-->
<!--"http://java.sun.com/dtd/web-app_2_3.dtd" >-->
```

## 写接口

### Model 层

#### DeveloperModel

```
public class DeveloperModel {
    private int id;
    private String name;
    private String site;
    private String avatar;

    //省略 get 和 set 方法

}
```

#### CommonModel

```
public class CommonModel {
    //code, 0: 接口请求成功, 有数据返回; 1: 接口请求成功, 无数据返回。
    private int code;
    //msg
    private String msg;
    //可以单个对象或List
    private Object data;

    //省略 get 和 set 方法

    public void setSuccess() {
        setCode(ConstantUtil.CODE_SUCCESS);
        setMsg(ConstantUtil.MSG_SUCCESS);
    }

    public void setFail() {
```



```
        setCode(ConstantUtil.CODE_FAIL);
        setMsg(ConstantUtil.MSG_FAIL);
    }
}
```

## ConstantUtil

```
public class ConstantUtil {
    public static final int CODE_SUCCESS = 0;
    public static final int CODE_FAIL = 1;
    public static final String MSG_SUCCESS = "success";
    public static final String MSG_FAIL = "fail";
}
```

## dao 层

### DeveloperDAO

```
public interface DeveloperDAO {
    List<DeveloperModel> getAllDevelopers();

    DeveloperModel getDeveloper(String developerId);

    boolean addDeveloper(DeveloperModel developerModel);

    boolean updateDeveloper(String id, String name);

    boolean deleteDeveloper(String id);
}
```

### DeveloperDAOImpl

```
@Repository("developerDAOImpl")
public class DeveloperDAOImpl implements DeveloperDAO {
    @Resource(name = "jdbcTemplate")
    private JdbcTemplate jdbcTemplate;

    @Override
    public List<DeveloperModel> getAllDevelopers() {
        String sql = "select * from developer";

        return query(sql);
    }

    @Override
    public DeveloperModel getDeveloper(String developerId) {
        String sql = "select * from developer where id=" + developerId;
        List<DeveloperModel> developerList = query(sql);
    }
}
```

```

        if (developerList.size() > 0) {
            return developerList.get(0);
        } else {
            return null;
        }
    }

}

private List<DeveloperModel> query(String sql) {
    final List<DeveloperModel> developerList = new ArrayList<>();
    jdbcTemplate.query(sql, new RowCallbackHandler() {
        @Override
        public void processRow(ResultSet resultSet) throws SQLException {
            int id = resultSet.getInt(1);
            String name = resultSet.getString(2);
            String site = resultSet.getString(3);
            String avatar = resultSet.getString(4);
            DeveloperModel developerModel = new DeveloperModel();
            developerModel.setId(id);
            developerModel.setName(name);
            developerModel.setSite(site);
            developerModel.setAvatar(avatar);
            developerList.add(developerModel);
        }
    });
    return developerList;
}

@Override
public boolean addDeveloper(DeveloperModel developerModel) {
    String sql = "INSERT INTO developer(name,site,avatar) VALUES(" +
        "'" + developerModel.getName() + "'," +
        "'" + developerModel.getSite() + "'," +
        "'" + developerModel.getAvatar() + "'" + ")";
    System.out.println("sql=" + sql);
    try {
        jdbcTemplate.execute(sql);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return false;
}

@Override
public boolean updateDeveloper(String developerId, String name) {
    String sql = "UPDATE developer SET name='" + name + "' WHERE id=" + developerId;
    System.out.println("sql=" + sql);
    try {

```

```

        jdbcTemplate.update(sql);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return false;
}

@Override
public boolean deleteDeveloper(String developerId) {
    String sql = "DELETE FROM developer WHERE id=" + developerId;
    System.out.println("sql=" + sql);
    try {
        jdbcTemplate.execute(sql);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return false;
}
}

```

## Controller 层

```

@Controller
@RequestMapping("/developer")
public class DeveloperController {

    private DeveloperDAO developerDAO;

    @Autowired
    DeveloperController(DeveloperDAO developerDAO) {
        this.developerDAO = developerDAO;
    }

    @RequestMapping(value = "/api/hello", method = RequestMethod.GET)
    public String hello() {
        return "hello";
    }

    @RequestMapping(value = "/api/developers", method = RequestMethod.GET)
    @ResponseBody//通过@ResponseBody返回json数据, 通过@RequestBody解析json
    public CommonModel getAllDevelopers() {
        List<DeveloperModel> developerList = developerDAO.getAllDevelopers();
        CommonModel commonModel = new CommonModel();
        if (developerList.size() > 0) {
            commonModel.setSuccess();
            commonModel.setObject(developerList);
        }
    }
}

```

```
    } else {
        commonModel.setFail();
    }
    return commonModel;
}

@RequestMapping(value = "/api/developer", method = RequestMethod.GET)
@ResponseBody
public CommonModel getDeveloper(String id) {
    //自动匹配参数
    System.out.println("developerId=" + id);

    DeveloperModel developerModel = developerDAO.getDeveloper(id);
    CommonModel commonModel = new CommonModel();
    if (developerModel != null) {
        commonModel.setSuccess();
        commonModel.setObject(developerModel);
    } else {
        commonModel.setFail();
    }
    return commonModel;
}

@RequestMapping(value = "/api/developer/add", method = RequestMethod.POST)
@ResponseBody
public CommonModel addDeveloper(String name, String site, String avatar) {
    //自动匹配参数
    System.out.println("name=" + name);
    DeveloperModel developerModel = new DeveloperModel();
    developerModel.setName(name);
    developerModel.setSite(site);
    developerModel.setAvatar(avatar);
    CommonModel commonModel = new CommonModel();
    if (developerDAO.addDeveloper(developerModel)) {
        commonModel.setSuccess();
    } else {
        commonModel.setFail();
    }
    return commonModel;
}

@RequestMapping(value = "/api/developer/update", method = RequestMethod.POST)
@ResponseBody
public CommonModel updateDeveloper(String id, String name) {
    //自动匹配参数
    System.out.println("name=" + name);
    CommonModel commonModel = new CommonModel();
    if (developerDAO.updateDeveloper(id, name)) {
        commonModel.setSuccess();
    } else {
        commonModel.setFail();
    }
}
```

```
    }  
    return commonModel;  
}  
  
//Spring MVC 不支持 put, delete 方法实现传参, 这里用到了 PathVariable  
@RequestMapping(value = "/api/developer/delete/{id}", method = RequestMethod.DELETE)  
@ResponseBody  
public CommonModel deleteDeveloper(@PathVariable("id") String id) {  
    //id 对应 {id}  
    System.out.println("developerId=" + id);  
    CommonModel commonModel = new CommonModel();  
  
    if (developerDAO.deleteDeveloper(id)) {  
        commonModel.setSuccess();  
    } else {  
        commonModel.setFail();  
    }  
    return commonModel;  
}  
}
```

实现 Http Get、Post、Update 请求方式，即数据库基本的增删改查，在 Postman 访问就好了。这节课数据库访问还是用的 JDBC，下篇介绍主流的 Spring + SpringMVC + MyBatis。

## 源码

公众号「吴小龙同学」回复：SpringMVCSample，获取完整的 Sample 项目代码。

## Spring + SpringMVC + MyBatis

MyBatis 是一个 Java 持久化框架，它通过 XML 描述符或注解把对象与存储过程或 SQL 语句关联起来。与其他的对象关系映射框架不同，MyBatis 并没有将 Java 对象与数据库表关联起来，而是将Java方法与 SQL 语句关联。MyBatis 允许用户充分利用数据库的各种功能，例如存储过程、视图、各种复杂的查询以及某数据库的专有特性。如果要对遗留数据库、不规范的数据库进行操作，或者要完全控制 SQL 的执行，MyBatis 是一个不错的选择。

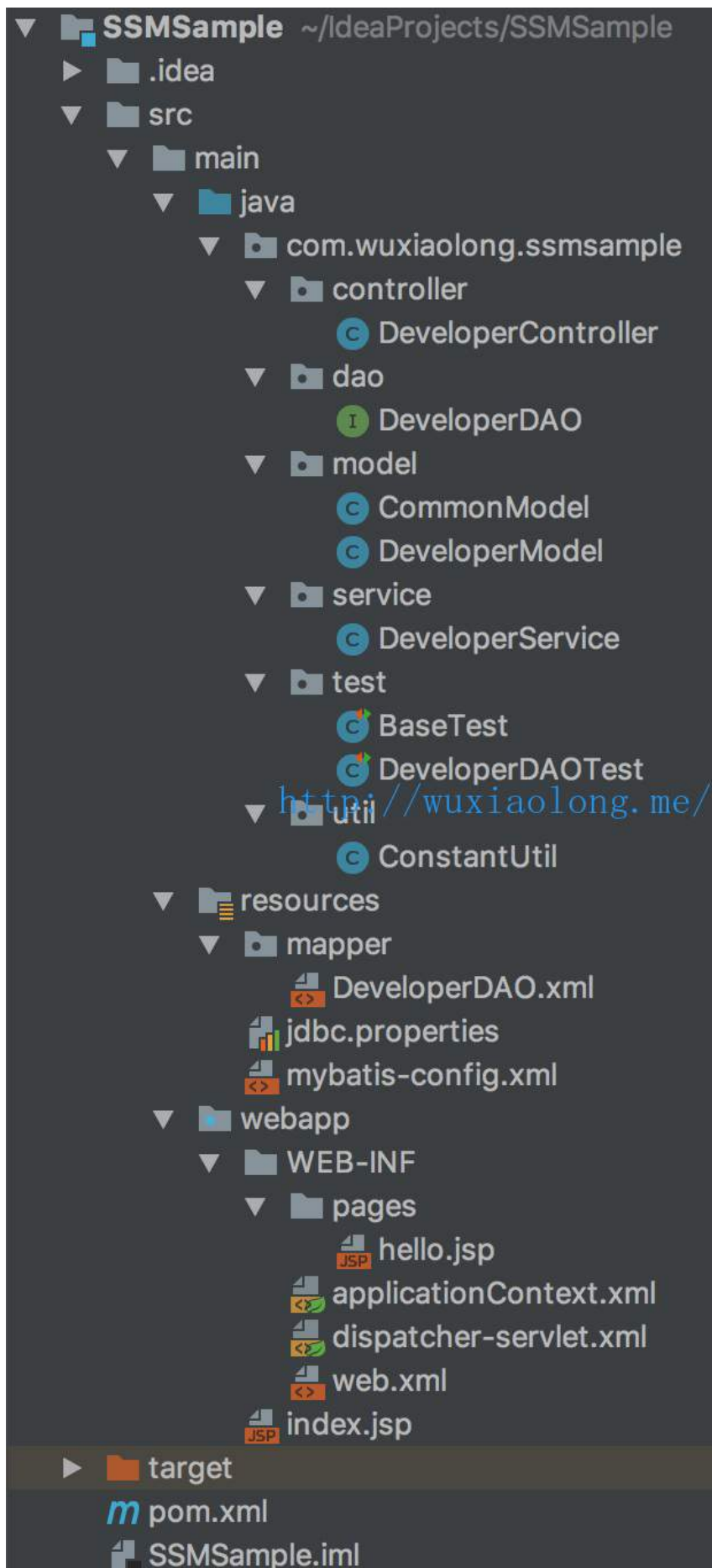
与 JDBC 相比，MyBatis 简化了相关代码：SQL 语句在一行代码中就能执行。MyBatis 提供了一个映射引擎，声明式的把 SQL 语句执行结果与对象树映射起来。通过使用一种内建的类XML表达式语言，或者使用 Apache Velocity 集成的插件，SQL 语句可以被动态的生成 - 来自维基百科。

## 准备工作

创建一个新 Maven 项目，然后配置 Tomcat，数据库还是用前面的，嗯，这些不累述了。

## 项目目录

Spring + SpringMVC + MyBatis 配置相当复杂，学习过程中折腾了很久，网上很多博客跟着做，最后都没有做成功，先看下我最终成功项目的目录：



- controller: 用于处理网站的请求;
- dao: 数据访问层 (接口) ;
- model: 数据模型;
- service: 业务逻辑层 (接口) ;
- test: 单元测试;

## pom.xml

Maven 依赖库比较多:

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
  <spring.version>5.0.5.RELEASE</spring.version>
</properties>

<dependencies>
  <!-- 单元测试 -->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
  </dependency>

  <!-- 1.日志 -->
  <!-- 实现slf4j接口并整合 -->
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.1.1</version>
  </dependency>

  <!-- 2.数据库 -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.37</version>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>c3p0</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.1.2</version>
  </dependency>

  <!-- DAO: MyBatis -->
  <dependency>
```



```
<groupId>org.mybatis</groupId>
<artifactId>mybatis</artifactId>
<version>3.3.0</version>
</dependency>
<dependency>
<groupId>org.mybatis</groupId>
<artifactId>mybatis-spring</artifactId>
<version>1.2.3</version>
</dependency>

<!-- 3.Servlet web -->
<dependency>
<groupId>taglibs</groupId>
<artifactId>standard</artifactId>
<version>1.1.2</version>
</dependency>
<dependency>
<groupId>jstl</groupId>
<artifactId>jstl</artifactId>
<version>1.2</version>
</dependency>
<!-- Gson -->
<dependency>
<groupId>com.google.code.gson</groupId>
<artifactId>gson</artifactId>
<version>2.8.2</version>
</dependency>
<dependency>
<groupId>javax.servlet</groupId>
<artifactId>javax.servlet-api</artifactId>
<version>3.1.0</version>
</dependency>

<!-- 4.Spring -->
<!-- 1)Spring核心 -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-core</artifactId>
<version>4.1.7.RELEASE</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-beans</artifactId>
<version>4.1.7.RELEASE</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>4.1.7.RELEASE</version>
</dependency>
<!-- 2)Spring DAO层 -->
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>4.1.7.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-tx</artifactId>
  <version>4.1.7.RELEASE</version>
</dependency>
<!-- 3)Spring web -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>4.1.7.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>4.1.7.RELEASE</version>
</dependency>
<!-- 4)Spring test -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>4.1.7.RELEASE</version>
</dependency>

<!-- redis客户端:Jedis -->
<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
  <version>2.7.3</version>
</dependency>
<dependency>
  <groupId>com.dyuproject.protostuff</groupId>
  <artifactId>protostuff-core</artifactId>
  <version>1.0.8</version>
</dependency>
<dependency>
  <groupId>com.dyuproject.protostuff</groupId>
  <artifactId>protostuff-runtime</artifactId>
  <version>1.0.8</version>
</dependency>

<!-- Map工具类 -->
<dependency>
  <groupId>commons-collections</groupId>
  <artifactId>commons-collections</artifactId>
  <version>3.2</version>
</dependency>
```

```
</dependencies>
```

## SSM 配置

嗯，SSM 配置相当复杂，一步步来。

### web.xml

和之前 Spring MVC 是一样的：

```
<!--<!DOCTYPE web-app PUBLIC-->
<!--"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"-->
<!--"http://java.sun.com/dtd/web-app_2_3.dtd" >-->
<!--报错: The content of element type "web-app" must match, 删除上面<!DOCTYPE...》内容-->

<web-app>

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

    <!--Spring MVC配置-->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
        <!--<param-value>classpath:spring/spring-*.xml</param-value-->
    </context-param>

    <!-- 注册ServletContext监听器, 创建容器对象, 并且将ApplicationContext对象放到Application域中 -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <!--在 web.xml 同级目录下将有个 dispatcher-servlet.xml 配置文件进行对应, 负责整个 mvc 的配置-->
    <!--applicationContext.xml 和 dispatcher-servlet.xml 区别: http://www.cnblogs.com/parryyang/p/5783399.html-->
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

```

<!-- 字符过滤器 传值乱码 -->
<filter>
    <filter-name>encodingFilter</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>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

</web-app>

```

## jdbc.properties

resources 下新建 jdbc.properties 文件，配置的是数据库信息：

```

jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/java4interface?useUnicode=true&characterEncoding=utf8
jdbc.username=root
jdbc.password=root

```

## applicationContext.xml

新建 applicationContext.xml，添加如下配置：

```

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

    <!-- 配置整合 MyBatis 过程 -->
    <!-- 1.配置数据库相关参数 properties 属性: ${url} -->
    <context:property-placeholder location="classpath:jdbc.properties"/>

```

```
<!-- 2.数据库连接池 -->
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <!-- 配置连接池属性 -->
    <property name="driverClass" value="${jdbc.driver}"/>
    <property name="jdbcUrl" value="${jdbc.url}"/>
    <property name="user" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>

    <!-- c3p0 连接池的私有属性 -->
    <property name="maxPoolSize" value="30"/>
    <property name="minPoolSize" value="10"/>
    <!-- 关闭连接后不自动 commit -->
    <property name="autoCommitOnClose" value="false"/>
    <!-- 获取连接超时时间 -->
    <property name="checkoutTimeout" value="10000"/>
    <!-- 当获取连接失败重试次数 -->
    <property name="acquireRetryAttempts" value="2"/>
</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"/>
    <!-- 配置 MyBatis 全局配置文件:mybatis-config.xml -->
    <property name="configLocation" value="classpath:mybatis-config.xml"/>
    <!-- 扫描 model 包 使用别名 -->
    <property name="typeAliasesPackage" value="com.wuxiaolong.ssmsample.model"/>

    <!-- 扫描 sql 配置文件:mapper 需要的 xml 文件 -->
    <property name="mapperLocations" value="classpath:mapper/*.xml"/>
</bean>

<!-- 4.配置扫描 dao 接口包, 动态实现 dao 接口, 注入到 Spring 容器中 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <!-- 注入 sqlSessionFactory -->
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
    <!-- 给出需要扫描 dao 接口包 -->
    <property name="basePackage" value="com.wuxiaolong.ssmsample.dao"/>
</bean>

<!--spring-service-->

<!-- 扫描 service 包下所有使用注解的类型 -->
<context:component-scan base-package="com.wuxiaolong.ssmsample.service"/>

<!-- 配置事务管理器 -->
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <!-- 注入数据库连接池 -->
```

```

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

</beans>

```

其中涉及配置文件 mybatis-config.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>
    <!-- 配置全局属性 -->
    <settings>
        <!-- 使用jdbc的getGeneratedKeys获取数据库自增主键值 -->
        <setting name="useGeneratedKeys" value="true" />

        <!-- 使用列别名替换列名 默认:true -->
        <setting name="useColumnLabel" value="true" />

        <!-- 开启驼峰命名转换:Table{create_time} -> Entity{createTime} -->
        <setting name="mapUnderscoreToCamelCase" value="true" />
    </settings>
</configuration>

```

配置目录 mapper, 新建 DeveloperDAO.xml, 这里是 sql 语句, 后面细讲。

扫描包 model、dao、service, 相应去创建即可。

## dispatcher-servlet.xml

新建 dispatcher-servlet.xml, 添加如下配置:

```

<?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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.s
pringframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/sch
ema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd http://www.springf
ramework.org/schema/context http://www.springframework.org/schema/context/spring-co
ntext.xsd">

    <!-- 此文件负责整个mvc中的配置 -->
    <!-- 扫描 controller 所在包的文件 -->
    <context:component-scan base-package="com.wuxiaolong.ssmsample.controller"/>

```

```

<!-- 定义一个视图解析器, 用于支持 Servlet、JSP 路径解析-->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

    <!--/WEB-INF/pages/ 是 jsp 目录, prefix: 前缀-->
    <property name="prefix" value="/WEB-INF/pages/" />
    <!--suffix: 后缀, 以 jsp 结尾-->
    <property name="suffix" value=".jsp" />
</bean>

<!--开启注解-->
<mvc:annotation-driven/>

<!--静态资源默认servlet配置
    (1)加入对静态资源的处理: js,gif,png
    (2)允许使用"/"做整体映射
-->
<mvc:default-servlet-handler/>
</beans>

```

这里涉及扫描接口包 controller, jsp 目录 pages, 相应去创建。

到这里 SSM 配置完成了, 不容易啊, 配置过程很容易晕掉。

## 写接口

### Model 层

ConstantUtil

新建 util 包, 再新建 ConstantUtil:

```

public class ConstantUtil {
    public static final int CODE_SUCCESS = 0;
    public static final int CODE_FAIL = 1;
    public static final String MSG_SUCCESS = "success";
    public static final String MSG_FAIL = "fail";
}

```

DeveloperModel

```

public class DeveloperModel {
    private int id;
    private String name;
    private String site;
    private String avatar;

    //省略 get 和 set 方法
}

```

```
}
```

## CommonModel

```
public class CommonModel {  
    //code, 0: 接口请求成功, 有数据返回; 1: 接口请求成功, 无数据返回。  
    private int code;  
    //msg  
    private String msg;  
    //可以单个对象或List  
    private Object data;  
  
    //省略 get 和 set 方法  
  
    public void setSuccess() {  
        setCode(ConstantUtil.CODE_SUCCESS);  
        setMsg(ConstantUtil.MSG_SUCCESS);  
    }  
  
    public void setFail() {  
        setCode(ConstantUtil.CODE_FAIL);  
        setMsg(ConstantUtil.MSG_FAIL);  
    }  
}
```

## dao 层

### DeveloperDAO

```
public interface DeveloperDAO {  
    List<DeveloperModel> getAllDevelopers();  
  
    DeveloperModel getDeveloper(String developerId);  
  
    boolean addDeveloper(DeveloperModel developerModel);  
  
    //如果想传入多个参数, 则需要在接口的参数上添加 @Param 注解  
    boolean updateDeveloper(@Param("id") String id, @Param("name") String name);  
  
    boolean deleteDeveloper(String id);  
}
```

不用写 DeveloperDAOImpl, MyBatis 会给我们动态实现 daoImpl, 我们需要编写的是相应的 mapper。

## DeveloperDAO.xml



这里为 dao 接口方法提供 sql 语句配置。

```
<?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.wuxiaolong.ssmsample.dao.DeveloperDAO">
    <!-- 目的: 为 dao 接口方法提供 sql 语句配置 -->
    <select id="getAllDevelopers" resultType="DeveloperModel">
        SELECT
        *
        FROM
        developer
    </select>

    <select id="getDeveloper" resultType="DeveloperModel" parameterType="String">
        SELECT
        *
        FROM
        developer
        WHERE
        id = #{id}
    </select>

    <insert id="addDeveloper">
        <!-- ignore 主键冲突, 报错 -->
        INSERT
        ignore
        INTO
        developer
        (name,site,avatar)
        VALUES
        (#{name}, #{site}, #{avatar})
    </insert>

    <update id="updateDeveloper">
        <!-- ignore 主键冲突, 报错 -->
        UPDATE
        developer
        SET
        name=#{name}
        WHERE
        id = #{id}
    </update>
    <delete id="deleteDeveloper" parameterType="String">
        DELETE
        FROM
        developer
        WHERE
        id= #{id}
```

```
</delete>

</mapper>
```

注意，这里 id 对应 dao 层方法名，查询就是 select，插入就是 insert，resultType 是返回值类型，parameterType 是参数类型（可选），最后 #{...} 中填写的是方法的参数等。

## service 层

业务逻辑层，中间层。

```
@Service
@Transactional
public class DeveloperService {

    @Autowired
    private DeveloperDAO developerDAO;

    public List<DeveloperModel> getAllDevelopers() {
        return developerDAO.getAllDevelopers();
    }

    public DeveloperModel getDeveloper(String developerId) {
        return developerDAO.getDeveloper(developerId);
    }

    public boolean addDeveloper(DeveloperModel developerModel) {
        return developerDAO.addDeveloper(developerModel);
    }

    public boolean updateDeveloper(String developerId, String name) {
        return developerDAO.updateDeveloper(developerId, name);
    }

    public boolean deleteDeveloper(String developerId) {
        return developerDAO.deleteDeveloper(developerId);
    }
}
```

## Controller 层

```
@Controller
@RequestMapping("/developer")
public class DeveloperController {

    private DeveloperService developerService;

    @Autowired
```

```
DeveloperController(DeveloperService developerService) {
    this.developerService = developerService;
}

@RequestMapping(value = "/api/hello", method = RequestMethod.GET)
public String hello() {
    return "hello";
}

@RequestMapping(value = "/api/developers", method = RequestMethod.GET)
@ResponseBody//通过@ResponseBody返回json数据, 通过@ResquestBody解析json
public CommonModel getAllDevelopers() {
    List<DeveloperModel> developerList = developerService.getAllDevelopers();
    CommonModel commonModel = new CommonModel();
    if (developerList.size() > 0) {
        commonModel.setSuccess();
        commonModel.setData(developerList);
    } else {
        commonModel.setFail();
    }
    return commonModel;
}

@RequestMapping(value = "/api/developer", method = RequestMethod.GET)
@ResponseBody
public CommonModel getDeveloper(String id) {
    //自动匹配参数
    System.out.println("developerId=" + id);

    DeveloperModel developerModel = developerService.getDeveloper(id);
    CommonModel commonModel = new CommonModel();
    if (developerModel != null) {
        commonModel.setSuccess();
        commonModel.setData(developerModel);
    } else {
        commonModel.setFail();
    }
    return commonModel;
}

@RequestMapping(value = "/api/developer/add", method = RequestMethod.POST)
@ResponseBody
public CommonModel addDeveloper(String name, String site, String avatar) {
    //自动匹配参数
    System.out.println("name=" + name);
    DeveloperModel developerModel = new DeveloperModel();
    developerModel.setName(name);
    developerModel.setSite(site);
    developerModel.setAvatar(avatar);
    CommonModel commonModel = new CommonModel();
    if (developerService.addDeveloper(developerModel)) {
```

```

        commonModel.setSuccess();
    } else {
        commonModel.setFail();
    }
    return commonModel;
}

@RequestMapping(value = "/api/developer/update", method = RequestMethod.POST)
@ResponseBody
public CommonModel updateDeveloper(String id, String name) {
    //自动匹配参数
    System.out.println("name=" + name);
    CommonModel commonModel = new CommonModel();
    if (developerService.updateDeveloper(id, name)) {
        commonModel.setSuccess();
    } else {
        commonModel.setFail();
    }
    return commonModel;
}

//Spring MVC 不支持 put, delete 方法实现传参, 这里用到了 PathVariable
@RequestMapping(value = "/api/developer/delete/{id}", method = RequestMethod.DELETE)
@ResponseBody
public CommonModel deleteDeveloper(@PathVariable("id") String id) {
    //自动匹配参数
    System.out.println("developerId=" + id);
    CommonModel commonModel = new CommonModel();

    if (developerService.deleteDeveloper(id)) {
        commonModel.setSuccess();
    } else {
        commonModel.setFail();
    }
    return commonModel;
}
}

```

也是实现基本的数据库增删改查，在 Postman 访问就好了，本教程最重头戏就这样讲完了。

## 源码

公众号「吴小龙同学」回复：SSMSample，获取完整的 Sample 项目代码。



## 云服务器部署

### 购买云服务器

部署服务器之前，得买个云服务器，我买的腾讯云，如图：

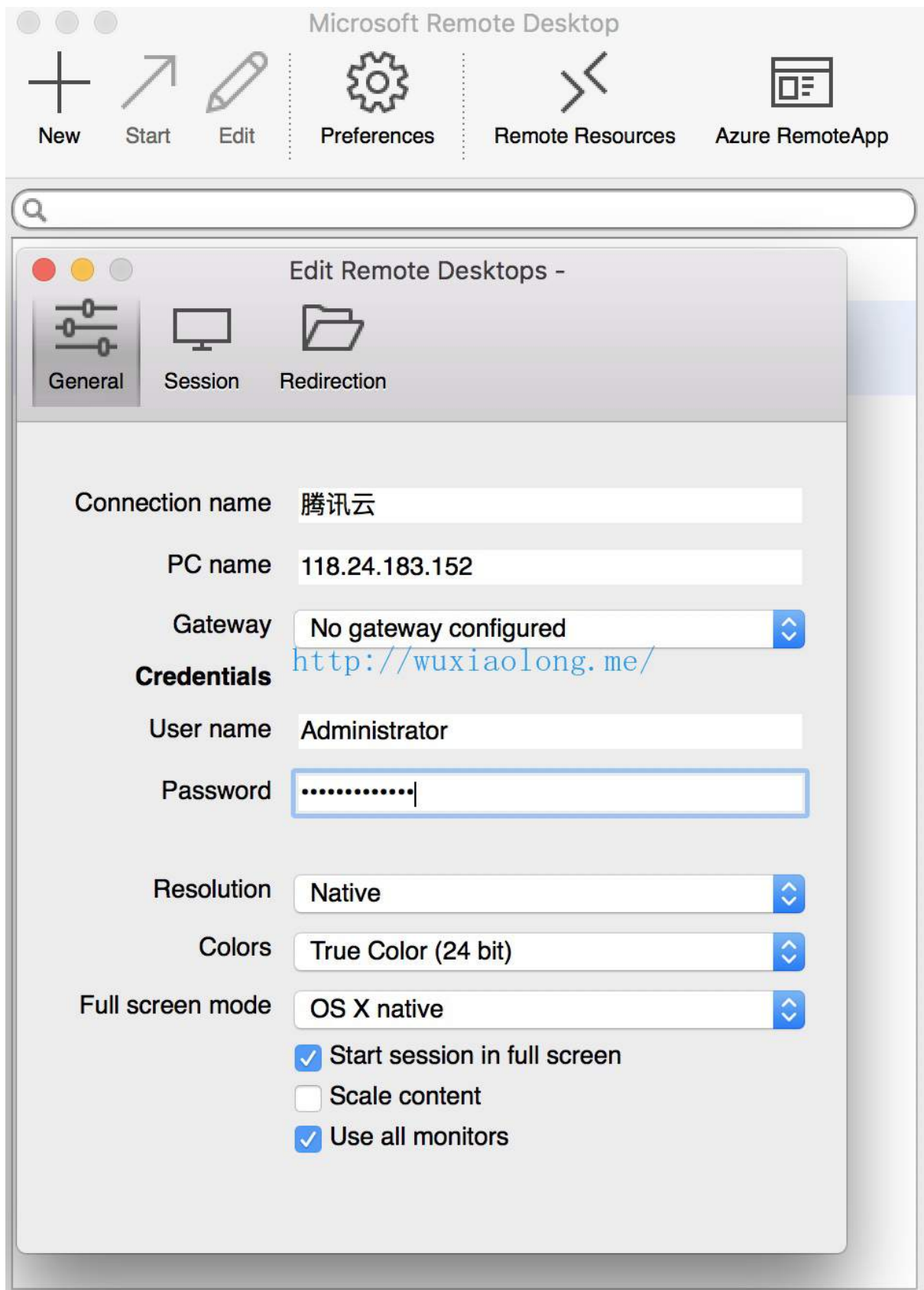
产品名称	配置详情	单价	数量	付费...	购买时长	优惠	费用
新购云服务器	地域：成都 可用区：成都二区 机型：系列2、标准型1核CPU、1G内存 镜像：Windows Server 2012 R2 标准版 64位中文版 存储：系统盘（50G云硬盘） 网络：基础网络 带宽：按带宽计费（带宽1Mbps） 名称：windows-1GB-cd-6391	60.00元/月	1	预付费	0个月	折扣：8.8折	316.80元 原价：360.00元
<a href="http://wuxiaolong.me/">http://wuxiaolong.me/</a>							
云数据库 (MySQL) 新 购	实例类型：主实例 计费模式：包年包月 配置类型：高IO版 配置：内存1000MB，硬盘25GB， MySQL5.6 地域：中国西南（成都） 可用区：成都二区 所属网络：基础网络 项目：默认项目 数据复制方式：异步复制	120.00元/月	1	预付费	<input type="button" value="-"/> 6 <input type="button" value="+"/> 月	折扣：5折	360.00元 原价：720.00元

当我把这张图片发到朋友圈，我只能说，新手是要交学费的：

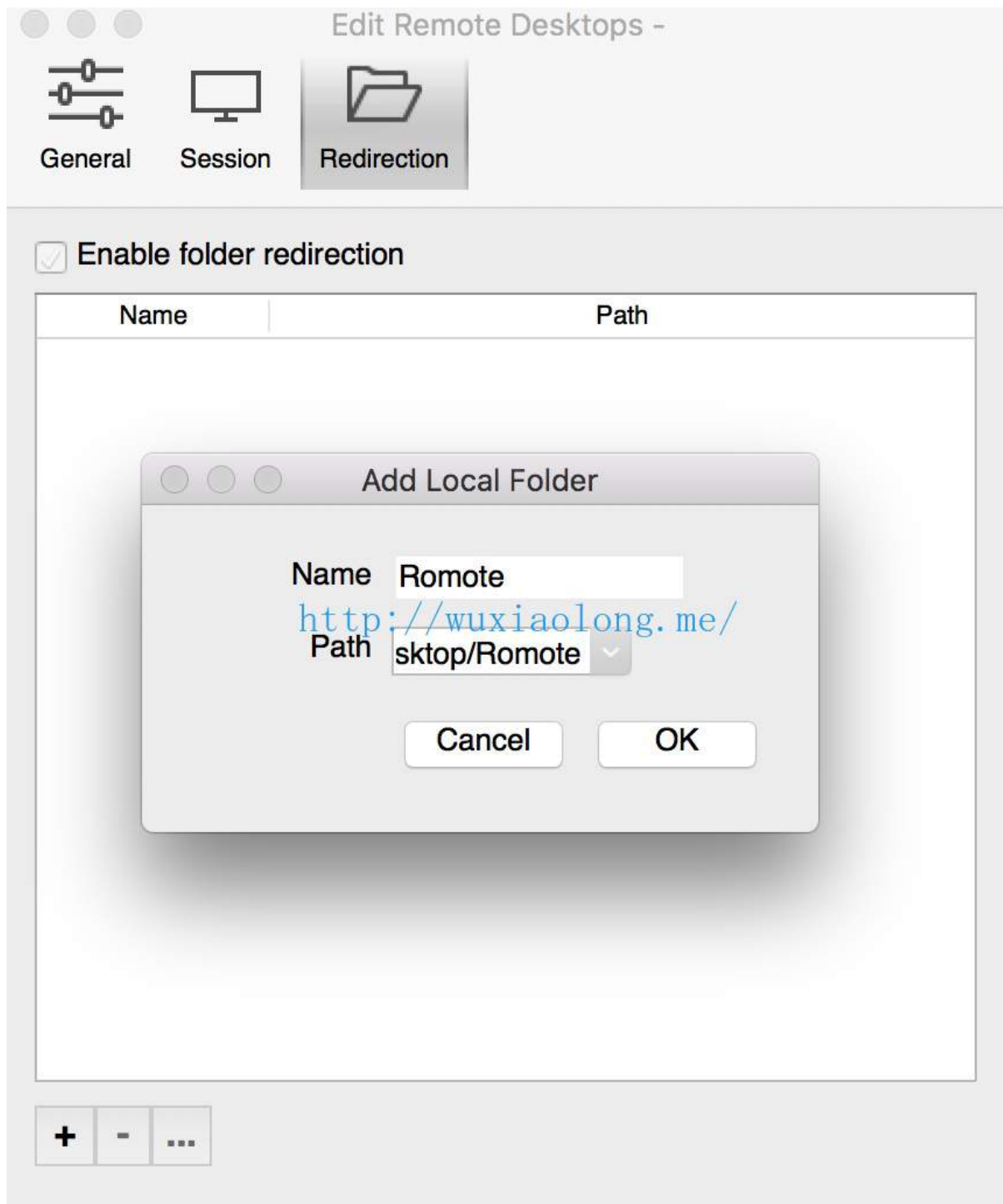
1、买贵了；2、云服务器不应该买 Windows，1G 内存，不得卡出翔，我都没有看；3、云数据库可以不买，云服务器就是一台电脑，可以自己装 MySQL。

### 连接服务器

买完云服务器，需要远程连接服务器，我用的是 Microsoft Remote Desktop，Mac 中国区下载不了，我网上找到了，放云盘了，公众号「吴小龙同学」回复：Microsoft Remote Desktop，获取这个软件，安装，然后打开 Microsoft Remote Desktop：

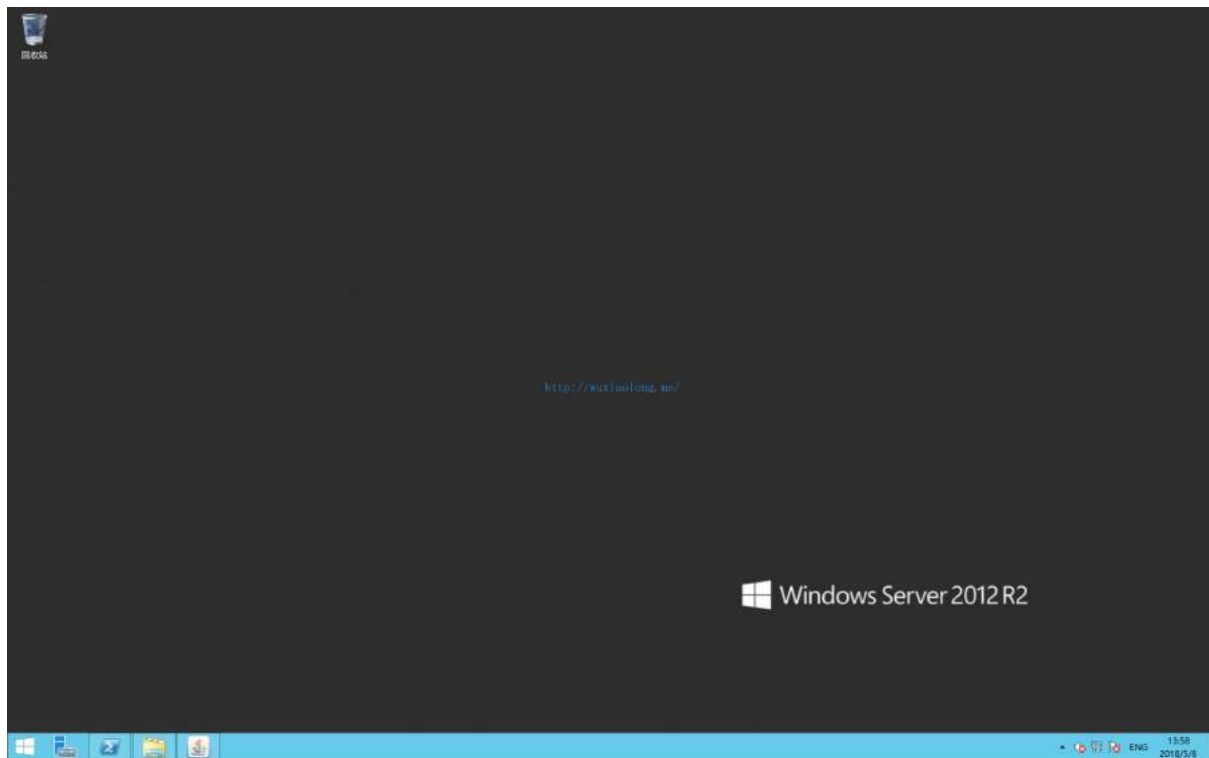


点击 New，填写相关信息，Connection name 可以随便写，PC name 就是云服务器公网地址，User name 和 Password 是购买完云服务器，会有消息通知，里面有账号信息；



点击 Redirection，勾选 Enable folder redirection，点击 +，Path 选取本地，这个很有用，可以用于本地和远程服务器文件交互，后面细讲。

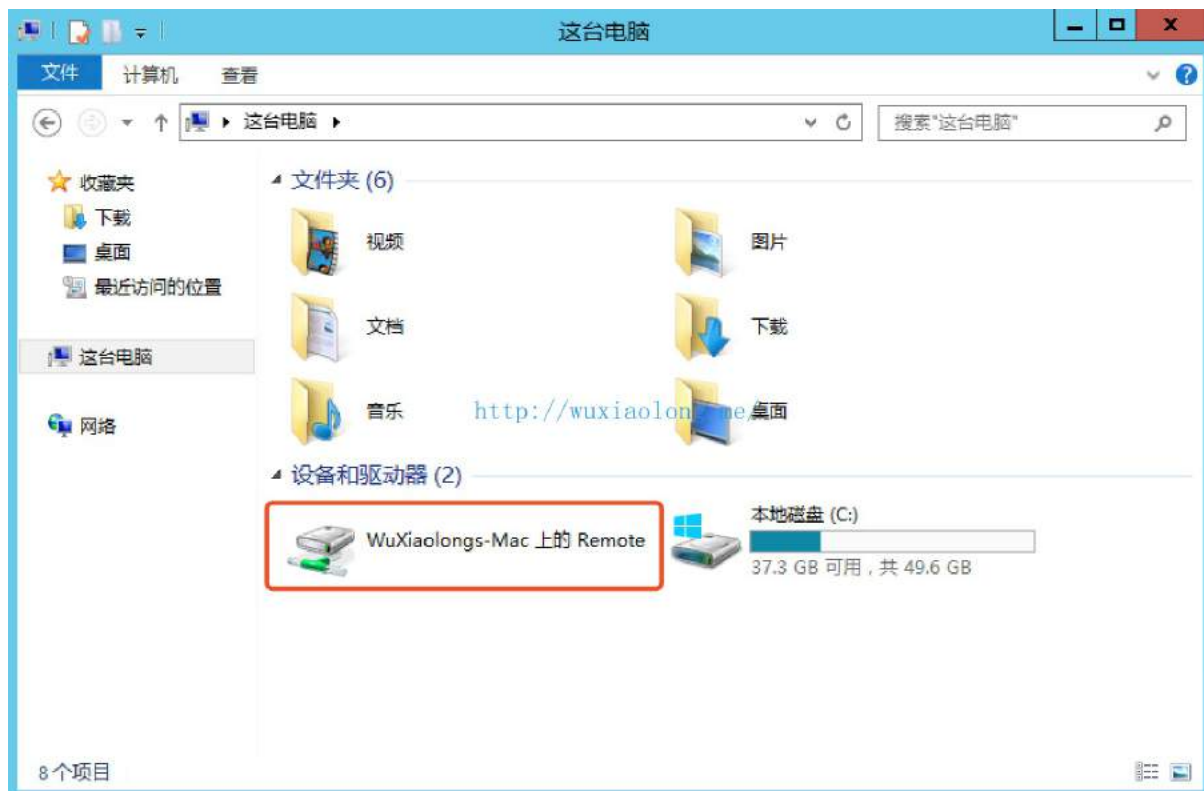




连接上的云服务器画面如上。

## 配置环境

云服务器，需要配置 JDK 和 Tomcat 环境，参考前面，这里不累述。云服务器是可以联网的，可以直接下载 JDK 和 Tomcat，然后安装，但是下载速度很慢，可以本地下载完了，传给服务器，问题来了，本地如何传文件给云服务器？网上很多教程用的 FileZilla，我没有成功，最后发现一个更简单方法，在 Microsoft Remote Desktop，我们不是配置了一个本地 Remote，把文件放到这个目录，云服务器是可以直接从这个目录，如下图，红色区域，双击打开，把文件拷出来。

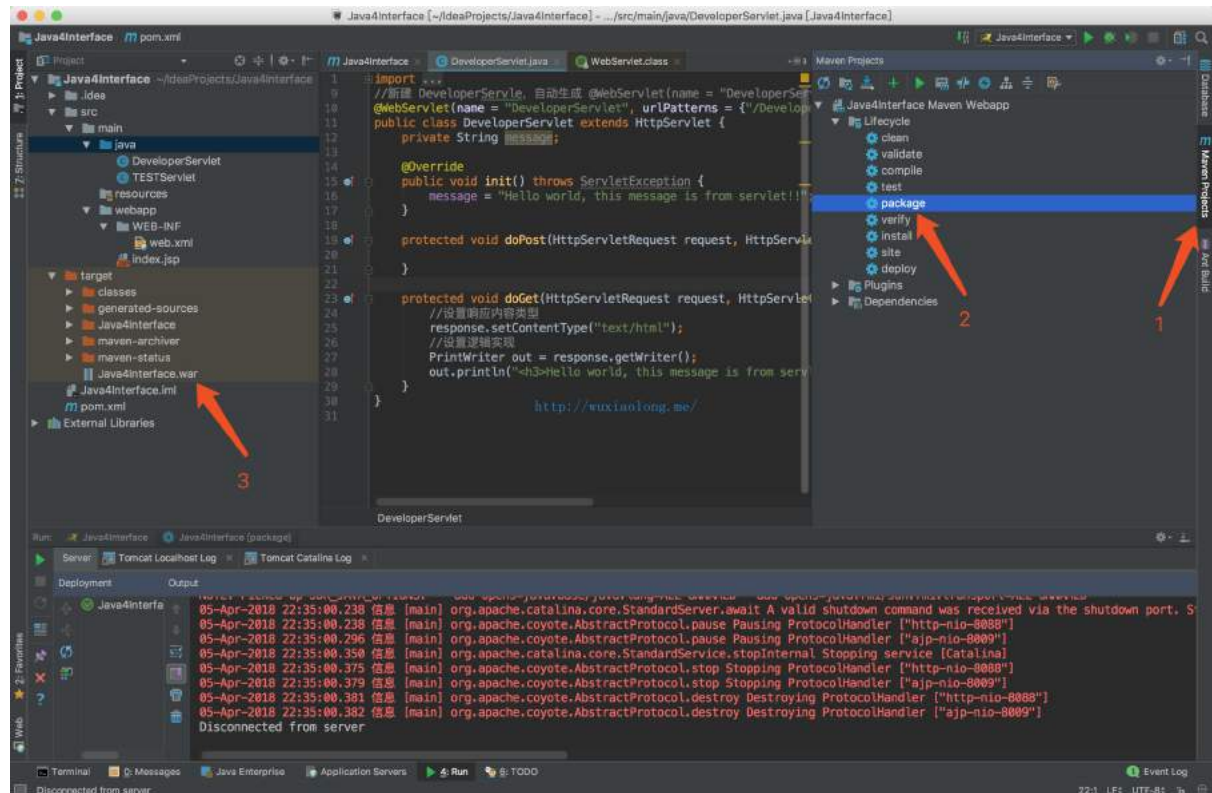


## 如何部署

项目可以打成 war 包，放到云服务器 tomcat/webapps 目录下，会自动生成对应的目录，比如 WuXiaolong.war，浏览器访问 云服务器公网 IP:端口号/项目名 即可。

## 编译打包

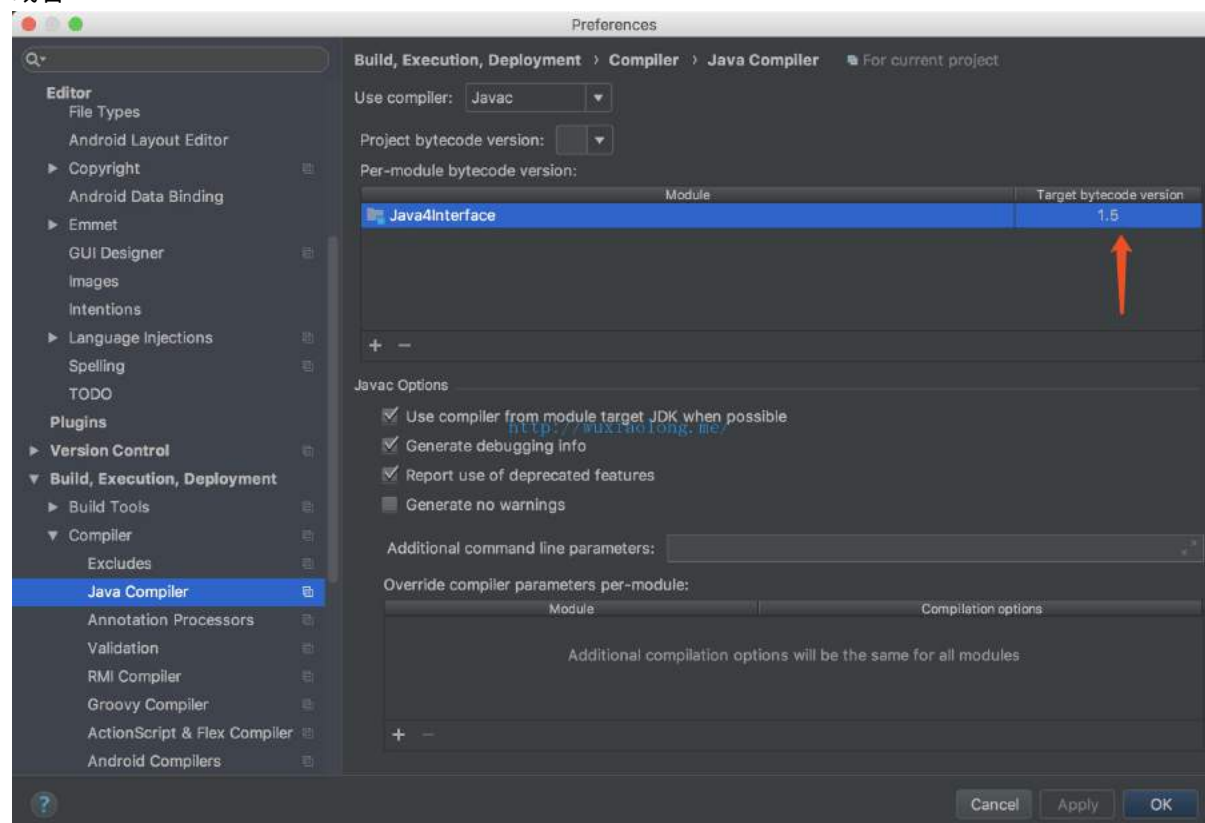
如何打成 war 包？



点击 package 可能会报：不再支持源选项 1.5。请使用 1.6 或更高版本。解决方式是 pom.xml 加入：

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

或者：



## 效果预览

<http://118.24.183.152/WuXiaolong/WuXiaolong>，云服务器截至 2018-10-25 22:16 到期，我只买了半年。这次《Java 开发接口》的学习之旅历时一个多月，2018 年清明假期起稿，到 5 月初才写好，期间困难重重，不过最后还是完成了，学无止境，这也只是后端的皮毛，我是新手，写作过程难免有错误，欢迎指正，互相交流学习，我的公众号「吴小龙同学」。