武汉纺织大学

Web应用开发课程设计

**调用api接口实现天气预报**

**学 院： 数学与计算机学院**

**班 级： 物联网11803班**

**姓 名： 马思文**

**学 号： 1804280312**

**指导老师： 聂刚**

**成 绩：**

**完成日期： 2020年12月18日**

目 录

[1 需求分析 1](#_Toc59199722)

[1.1天气预报界面 1](#_Toc59199723)

[1.2主菜单 1](#_Toc59199724)

[2 系统设计 1](#_Toc59199725)

[2.1 UML类图（Class Diagram） 1](#_Toc59199726)

[2.1.1 ContryWeather类 1](#_Toc59199727)

[2.1.2 Employee类 2](#_Toc59199728)

[2.1.2 Explore类 3](#_Toc59199729)

[2.1.3 Sale类 4](#_Toc59199730)

[2.2 UML时序图（Sequence Diagram） 4](#_Toc59199731)

[2.2.1 Dao层数据访问 4](#_Toc59199732)

[2.2.2 数据转换servlet层 4](#_Toc59199733)

[2.2.3 服务层CountryWeatherService 4](#_Toc59199734)

[2.2.4 测试类TestContryWeatherDao 5](#_Toc59199735)

[3 系统实现 5](#_Toc59199736)

[3.1 项目结构 5](#_Toc59199737)

[3.2 配置文件 5](#_Toc59199738)

[3.2.1 pom.xml文件 6](#_Toc59199739)

[3.2.2 application-spring文件 8](#_Toc59199740)

[3.2.3 spring-mvc文件 8](#_Toc59199741)

[3.3 VO类TianqiController.java 9](#_Toc59199742)

[3.4 SERVICE接口类TianqiService.java 9](#_Toc59199743)

[3.6 工具包Util 10](#_Toc59199744)

[3.6.1 GetDataUtil.java 10](#_Toc59199745)

[3.8 用户界面配置文件web.xml 11](#_Toc59199746)

[4 系统测试 12](#_Toc59199747)

[5 系统总结 13](#_Toc59199748)

# 1 需求分析

设计一个天气查询界面，实现 “查询”功能，具体要求如下：

## 1.1天气预报界面

当程序运行时，在搜索框内输入需要搜索的城市名，回车后就会显示当前及以后一周的天气预报，

## 1.2主菜单

当用户进入网页，显示如下：



# 2 系统设计

## 2.1 UML类图（Class Diagram）

### 2.1.1 ContryWeather类

对于各个城市天气显示模块，共设计如下6个类。

* Vo实体类CountryWeather：与数据库结构进行映射的类。主要由属性，setter, getter方法组成，VO类中的属性与表中的字段相对应，每一个VO类的对象都表示表中的每一条记录。
* DAO接口CountryWeatherDao：主要定义操作的接口，定义一系列数据库的原子性操作，例如增删改查（通常称为CRUD）等。创建CountryWeatherDao用于查询数据中所有的记录，使用DbUtils组件来实现
* service业务逻辑实现类CountryWeatherService：对于数据层的原子操作进行整合。还要负责数据库的打开与关闭（不管是否出异常，数据库都要关闭）
* servlet工厂类CountryWeatherServlet：创建CountryWeatherServlet，使用注解指定访问地址是：/CountryWeather。在Servlet调用业务层，将查询到的数据转换成JSON对象，打印到浏览器端。
* test测试类类TestContryWeatherDao：使用JUnit测试一下编写的Dao方法，同时测试一下JSON对象转换是否正确。这里使用flexjson组件进行转换。

各类的结构及类之间的关系如图所示：

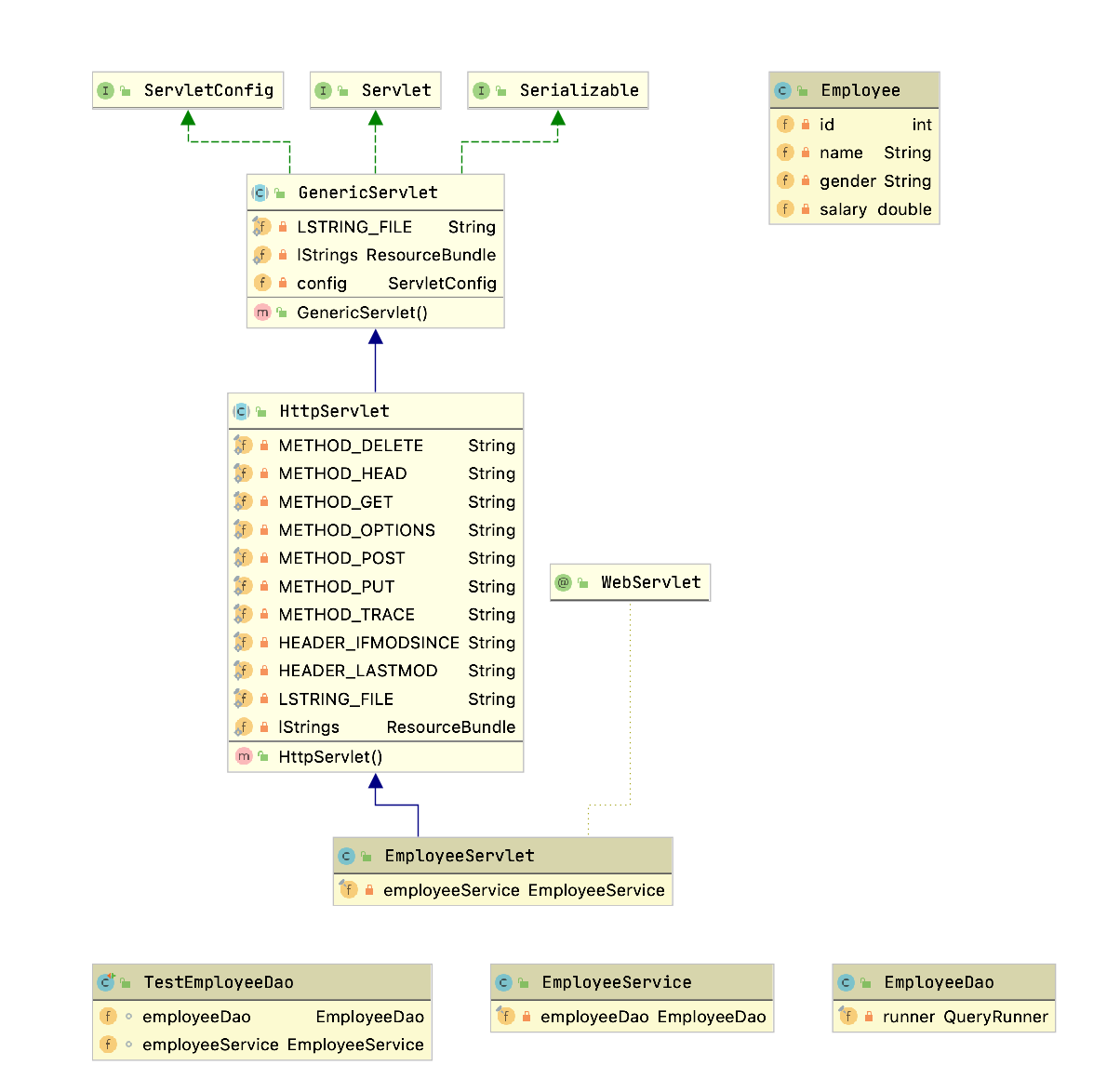
图片包含 图形用户界面

描述已自动生成

### 2.1.2 Employee类

* Vo实体类Employee：与数据库结构进行映射的类。主要由属性，setter, getter方法组成，VO类中的属性与表中的字段相对应，每一个VO类的对象都表示表中的每一条记录。
* DAO接口EmployeeDao：主要定义操作的接口，定义一系列数据库的原子性操作，例如增删改查（通常称为CRUD）等。创建EmployeeDao用于查询数据中所有的记录，使用DbUtils组件来实现
* service业务逻辑实现类EmployeeService：对于数据层的原子操作进行整合。还要负责数据库的打开与关闭（不管是否出异常，数据库都要关闭）
* servlet工厂类CountryWeatherServlet：创建EmployeeServlet，使用注解指定访问地址是：/employee。在Servlet调用业务层，将查询到的数据转换成JSON对象，打印到浏览器端。
* test测试类类TestEmployeeDao：使用JUnit测试一下编写的Dao方法，同时测试一下JSON对象转换是否正确。这里使用flexjson组件进行转换。

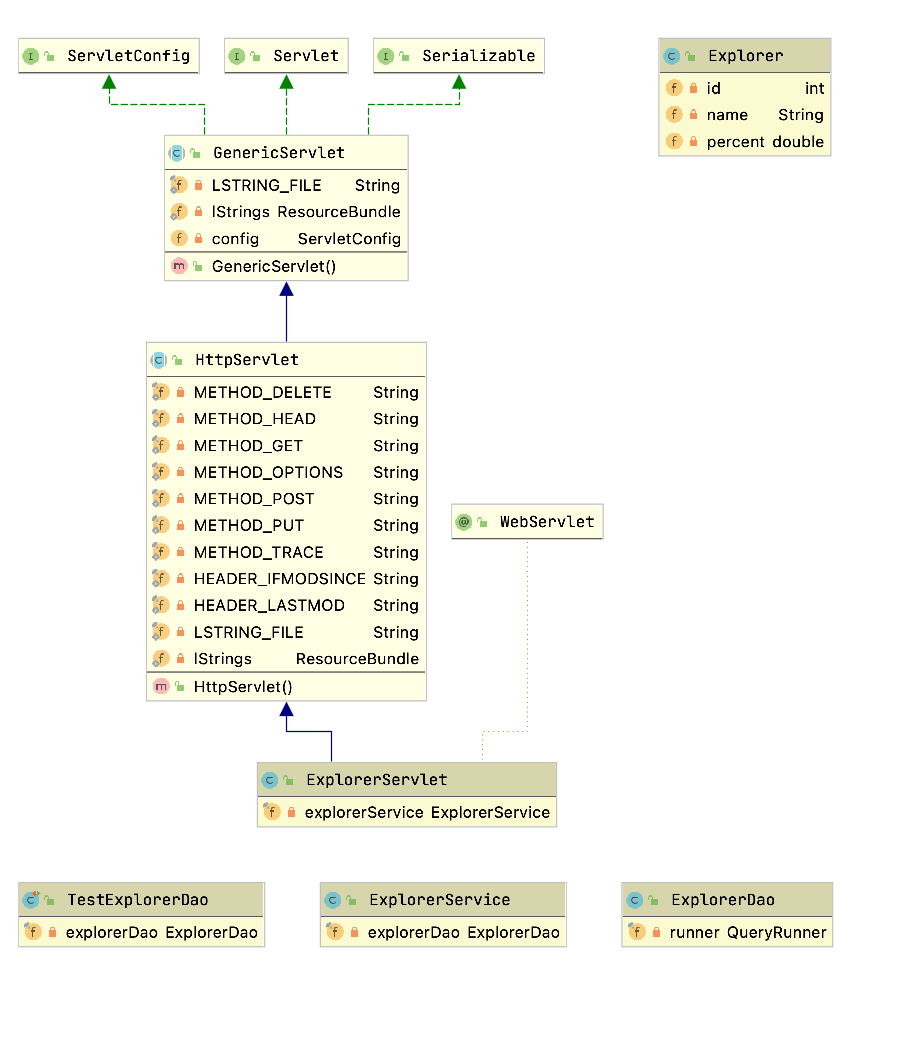
各类的结构及类之间的关系如图所示：



### 2.1.2 Explore类

* Vo实体类Explore：与数据库结构进行映射的类。主要由属性，setter, getter方法组成，VO类中的属性与表中的字段相对应，每一个VO类的对象都表示表中的每一条记录。
* DAO接口ExploreDao：主要定义操作的接口，定义一系列数据库的原子性操作，例如增删改查（通常称为CRUD）等。创建ExploreDao用于查询数据中所有的记录，使用DbUtils组件来实现
* service业务逻辑实现类ExploreService：对于数据层的原子操作进行整合。还要负责数据库的打开与关闭（不管是否出异常，数据库都要关闭）
* servlet工厂类ExploreServlet：创建ExploreServlet，使用注解指定访问地址是：/CountryWeather。在Servlet调用业务层，将查询到的数据转换成JSON对象，打印到浏览器端。
* test测试类类ExploreDao：使用JUnit测试一下编写的Dao方法，同时测试一下JSON对象转换是否正确。这里使用flexjson组件进行转换。

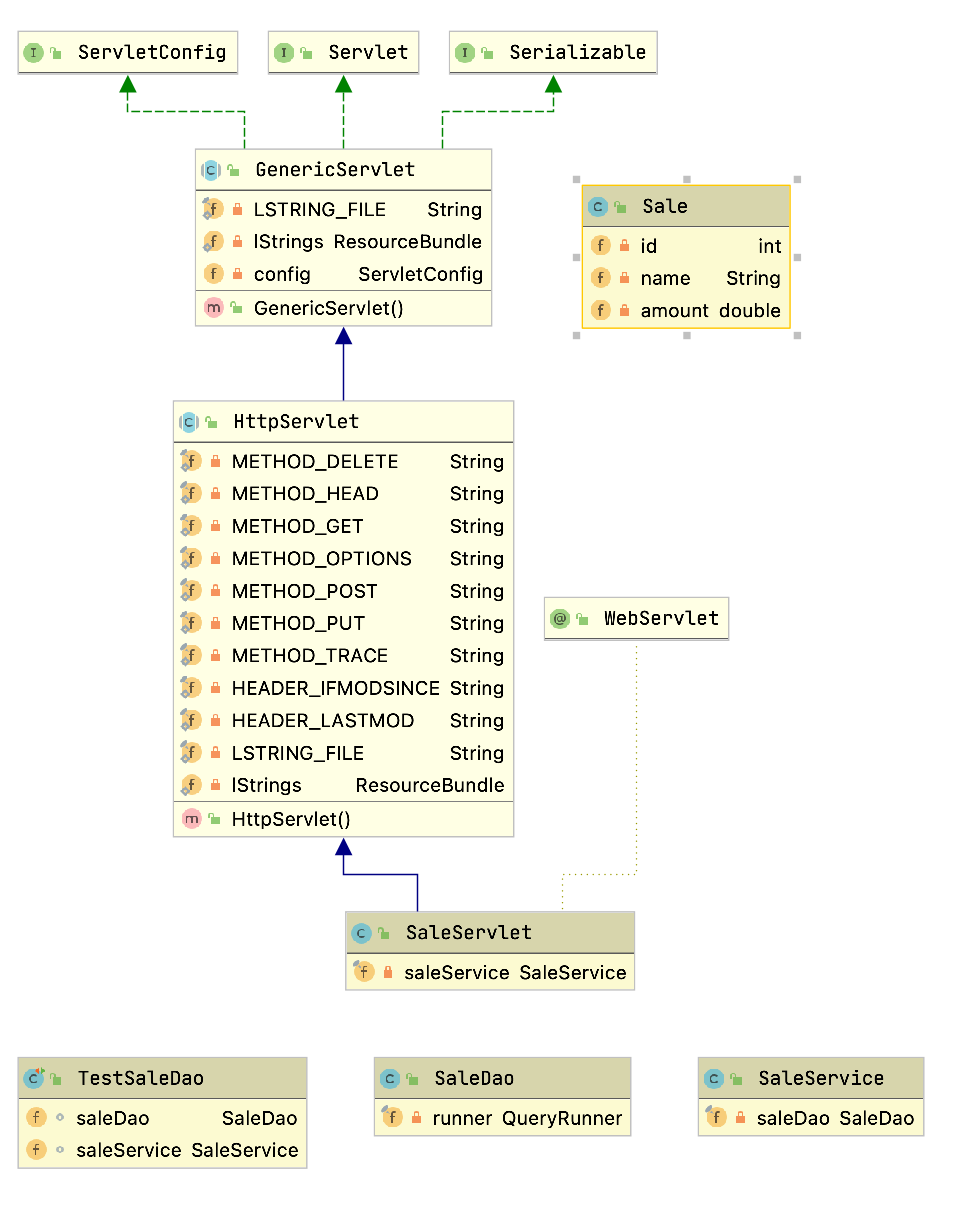
各类的结构及类之间的关系如图所示：



### 2.1.3 Sale类

* Vo实体类Sale：与数据库结构进行映射的类。主要由属性，setter, getter方法组成，VO类中的属性与表中的字段相对应，每一个VO类的对象都表示表中的每一条记录。
* DAO接口SaleDao：主要定义操作的接口，定义一系列数据库的原子性操作，例如增删改查（通常称为CRUD）等。创建SaleDao用于查询数据中所有的记录，使用DbUtils组件来实现
* service业务逻辑实现类SaleService：对于数据层的原子操作进行整合。还要负责数据库的打开与关闭（不管是否出异常，数据库都要关闭）
* servlet工厂类SaleServlet：创建CountryWeatherServlet，使用注解指定访问地址是：/CountryWeather。在Servlet调用业务层，将查询到的数据转换成JSON对象，打印到浏览器端。
* test测试类类TestSaleDao：使用JUnit测试一下编写的Dao方法，同时测试一下JSON对象转换是否正确。这里使用flexjson组件进行转换。

各类的结构及类之间的关系如图所示：



## 2.2 UML时序图（Sequence Diagram）

以CountryWeather类为例，其他类同理。

### 2.2.1 Dao层数据访问

图表, 图示, 箱线图

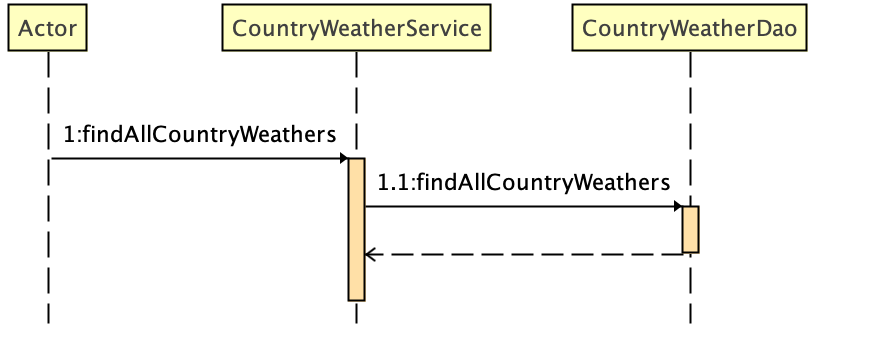
描述已自动生成

### 2.2.2 数据转换servlet层

图示

描述已自动生成

### 2.2.3 服务层CountryWeatherService



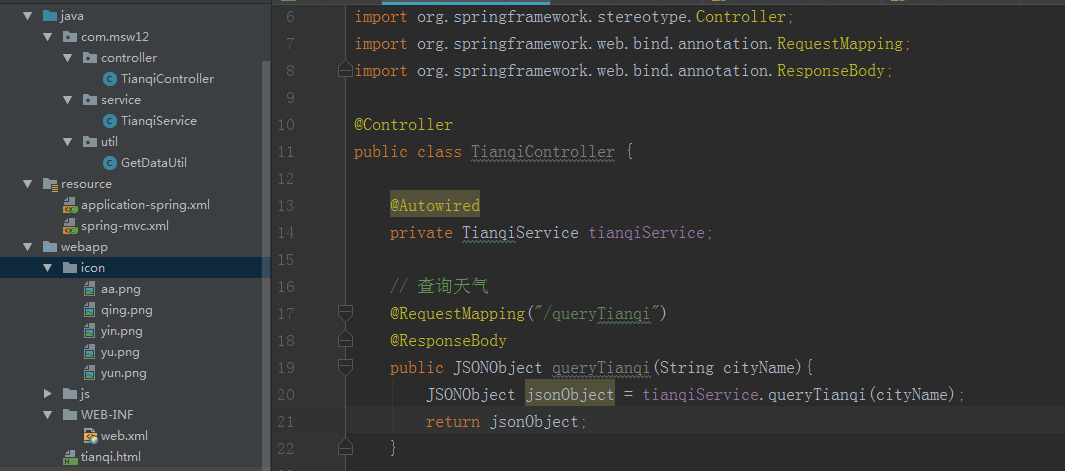
### 2.2.4 测试类TestContryWeatherDao

图示

描述已自动生成

# 3 系统实现

## 3.1 项目结构



## 3.2 配置文件

在项目下新建Source Folder，名为resoure

### 3.2.1 pom.xml文件

该配置文件主要是以文件形式保存数据的驱动类名称，连接网络数据库的地址，访问数据库的信息，程序运行时会读取该文件相关信息，当相关信息发生变化时，只需修改配置文件而不用修改源代码，增加程序的可扩展型。主要描述了项目的maven坐标，依赖关系，开发者需要遵循的规则，缺陷管理系统，组织和licenses，以及其他所有的项目相关因素，是项目级别的配置文件。

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

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.msw12</groupId>

<artifactId>demo\_msw12</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>war</packaging>

<name>demo\_msw12 Maven Webapp</name>

<!-- FIXME change it to the project's website -->

<url>http://www.example.com</url>

<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>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.62</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

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

<version>4.0.0</version>

</dependency>

<dependency>

<groupId>javax.servlet.jsp</groupId>

<artifactId>jsp-api</artifactId>

<version>2.2</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.9.5</version>

</dependency>

<!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.62</version>

</dependency>

</dependencies>

<build>

<finalName>demo\_msw12</finalName>

<pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->

<plugins>

<plugin>

<artifactId>maven-clean-plugin</artifactId>

<version>3.1.0</version>

</plugin>

<!-- see http://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_war\_packaging -->

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<version>3.0.2</version>

</plugin>

<artifactId>maven-install-plugin</artifactId>

<version>2.5.2</version>

</plugin>

<plugin>

<artifactId>maven-deploy-plugin</artifactId>

<version>2.8.2</version>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

### 3.2.2 application-spring文件

描述该文件作用，并对里面的配置进行解释

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

<beans xmlns="http://www.springframework.org/schema/beans"

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

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

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

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

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

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

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

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

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

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

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

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

<!--注解扫描-->

<context:component-scan base-package="com.msw12">

<context:exclude-filter type="annotation"

expression="org.springframework.stereotype.Controller"/>

</context:component-scan>

</beans>

### 3.2.3 spring-mvc文件

描述该文件作用，并对里面的配置进行解释

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

<beans xmlns="http://www.springframework.org/schema/beans"

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

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

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

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

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

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/aop

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

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

http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd

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

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

<!--开启注解扫描-->

<context:component-scan base-package="com.msw12">

<!--只扫描控制器-->

<context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>

</context:component-scan>

<mvc:default-servlet-handler/>

<mvc:annotation-driven/>

</beans>

## 3.3 VO类TianqiController.java

描述该文件作用，并对里面的关键代码进行解释

**public class TianqiController {**

**@Autowired**

**private TianqiService tianqiService;**

**// 查询天气**

**@RequestMapping("/queryTianqi")**

**@ResponseBody**

**public JSONObject queryTianqi(String cityName){**

**JSONObject jsonObject = tianqiService.queryTianqi(cityName);**

**return jsonObject;**

**}**

**}**

## 3.4 SERVICE接口类TianqiService.java

描述该文件作用，并对里面的关键代码进行解释

**@Controller**

**public class TianqiController {**

**@Autowired**

**private TianqiService tianqiService;**

**// 查询天气**

**@RequestMapping("/queryTianqi")**

**@ResponseBody**

**public JSONObject queryTianqi(String cityName){**

**JSONObject jsonObject = tianqiService.queryTianqi(cityName);**

**return jsonObject;**

**}**

**}**

## 3.6 工具包Util

### 3.6.1 GetDataUtil.java

描述该文件作用，并对里面的关键代码进行解释

public class GetDataUtil {

/\*\*

\* 向指定URL发送GET方法的请求

\*

\* @param url

\* 发送请求的URL

\* @param param

\* 请求参数，请求参数应该是 name1=value1&name2=value2 的形式。

\* @return URL 所代表远程资源的响应结果

\*/

public static String sendGet(String url, String param) {

String result = "";

BufferedReader in = null;

try {

String urlNameString = url + "?" + param;

URL realUrl = new URL(urlNameString);

// 打开和URL之间的连接

URLConnection connection = realUrl.openConnection();

// 设置通用的请求属性

connection.setRequestProperty("accept", "\*/\*");

connection.setRequestProperty("connection", "Keep-Alive");

connection.setRequestProperty("user-agent",

"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;SV1)");

// 建立实际的连接

connection.connect();

// 获取所有响应头字段

Map<String, List<String>> map = connection.getHeaderFields();

// 遍历所有的响应头字段

for (String key : map.keySet()) {

System.out.println(key + "--->" + map.get(key));

}

// 定义 BufferedReader输入流来读取URL的响应

in = new BufferedReader(new InputStreamReader(

connection.getInputStream(), "UTF-8"));

String line;

while ((line = in.readLine()) != null) {

line = decodeUnicode(line);

result += line;

}

} catch (Exception e) {

System.out.println("发送GET请求出现异常！" + e);

e.printStackTrace();

}

// 使用finally块来关闭输入流

finally {

try {

if (in != null) {

in.close();

}

} catch (Exception e2) {

e2.printStackTrace();

}

}

return result;

}

/\*\*

\* 字符串含有Unicode编码转化为正常字符串

## 3.8 用户界面配置文件web.xml

描述该文件作用，并对里面的关键代码进行解释

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

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"

version="4.0">

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!--contextConfigLocation参数是用来指定spring的配置文件的-->

<context-param>

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

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

</context-param>

<!--定义Spring MVC前端控制器-->

<servlet>

<servlet-name>spring-mvc</servlet-name>

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

<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>

<!-- 让springmvc的前端控制器拦截所有请求-->

<servlet-mapping>

<servlet-name>spring-mvc</servlet-name>

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

</servlet-mapping>

<!--编码过滤器-->

<filter>

<filter-name>characterEncodingFilter</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>characterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

# 4 系统测试

设计测试用例，给出程序每个功能模块的运行结果截图。

1，打开用户界面显示如下



2，输入广州，查询广州今天以及未来一周天气



3，查询长沙天气及未来一周天气预报



# 5 系统总结

本次的课程设计项目，不是特别的难，采用了最基础的mvc架构来完成本次课设。遇到不会的问题也可以及时解决，本次实验没有用到数据库，都是直接调用网上的资源数据，实现天气预报的查询，很多代码都有，主要是调用api接口实现一个代码的可视化，在网站上找小图标运用在自己的项目上，由于折线图实在是不会，本次项目并未运用到折线图。输入城市名搜索后回车调用api接口后可以看到当前城市的以及未来一周的天气情况，使用json作为存放服务端数据的载体，将服务端数据生成为json,以json的形式存放，供外界调用，获取数据。前端输入城市名后回车后，后端服务器首先调用TianqiController.java文件查询，然会再调用TianqiService.java文件将数据转换成为json对象后调用GetDataUtil.java文件，发送get请求建立连接，获取需要的数据，将在网络上调取的数据返回给前端服务器，显示城市天气。做项目的时候我发现我请求的数据返回的不是正常的中文编码，特意在网上搜索找到编码格式转换的代码，解决了这个问题。这个学期以来，关于web课程的学习，有些时我是跟不上的，因为没有Java语言这方面的基础，但是还是有在一点点的跟着学，私下有找寻相关资料进行自学，结果虽然不是非常好，但是也是进了最大的努力来弥补，也很开心可以遇到这么一位负责任有能力的老师，我以后也一直不会放弃这门课程的学习，希望未来可以将自己的所学用到需要的地方，给自己就打个75分吧，缺失的5分是项目的不足，20分空间给未来继续努力的自己。我会在学习的道路上不断积累，学习更多的技术来创造更好的未来。