**SpringBoot**框架

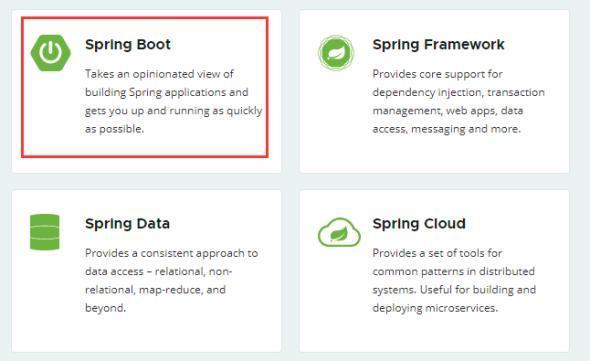
一、 SpringBoot简介

**1** 、**SpringBoot**介绍

SpringBoot ：它也是Spring体系中一员。Spring Boot是由Pivotal团队提供的全新[框](https://baike.baidu.com/item/%E6%A1%86%E6%9E%B6/1212667) 架，其设计目的是用来[简化](https://baike.baidu.com/item/%E7%AE%80%E5%8C%96/3374416)新[Spring](https://baike.baidu.com/item/Spring/85061)应用的初始搭建以及开发过程。该框架使用了特定的方

式来进行配置，从而使开发人员不再需要定义样板化的配置。通过这种方式，Spring Boot

致力于在蓬勃发展的快速应用开发领域(rapid application development)成为领导者。



Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring conﬁguration.

SpringBoot可以快速简单的去开发Spring项目。只需要通过一个简单的main方法就能够 去运行web工程(普通的java项目可以)。

SpringBoot可以简单的与第三方的库jar包或依赖)进行整合(因为SpringBoot给我们 提供大量的只配置以及启动器)。

SpringBoot框架是快速帮助我们构建企业级项目环境(项目构建、 jar包管理、框架的

整合，项目的打包，发布等等)。

**2** 、**SpringBoot**的特点

 Create stand-alone Spring applications

创建独立的Spring应用

 Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR ﬁles)

内嵌Tomcat、jetty、无法手动发布war文件

 Provide opinionated 'starter' dependencies to simplify your build conﬁguration

提供大量的starter (启动器)来快速构建项目

 Automatically conﬁgure Spring and 3rd party libraries whenever possible

SpringBoot提供自动与第三方库整合

 Provide production-ready features such as metrics, health checks and externalized

conﬁguration

提供生产环境下的安全检查、健康指标监控等。

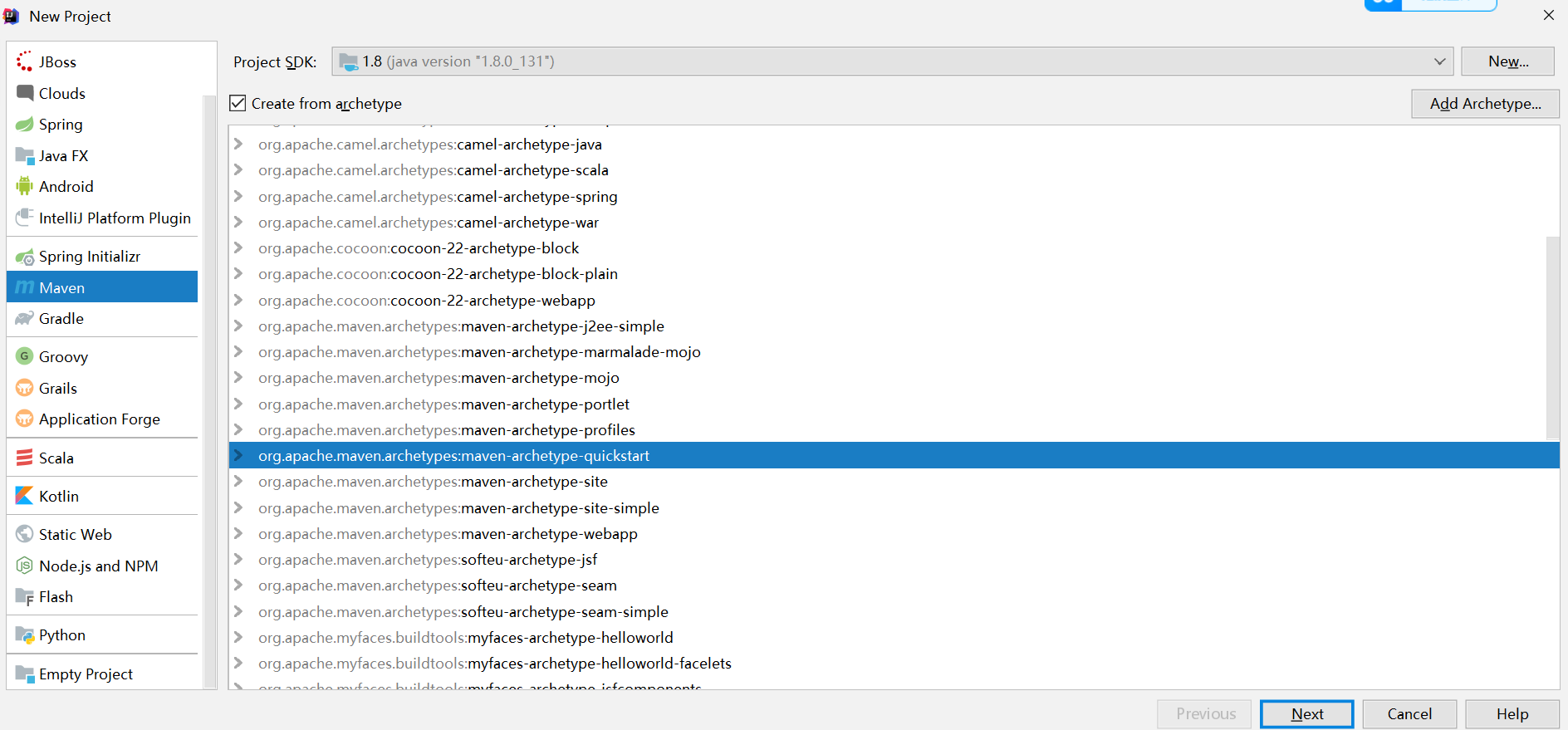
 Absolutely no code generation and no requirement for XML conﬁguration

不需要代码嵌入以及xml文件的配置

二 SpringBoot快速入门

**1** 、搭建**SpringBoot**项目

新建maven项目



**2** 、**SpringBoot**示例

2. 1 关于Maven的父子工程

maven构建的项目默认有三种方式：

 jar方式：就是普通的jar项目，通过main方法启动运行。开发的项目打包之后jar文件

 war方式：JavaWEB工程，需要运行在web服务器中。将来项目开发完成打包之后为war

文件

 pom方式：它一般在项目中用于构建整个项目公共信息(管理整个项目中通用的内容，目

前主要用来管理项目的依赖)。在我们真正开发项目的时候，自己不管创建的jar还是war ，可以在自己的 pom.xml文件中通过parent标签来引用整个pom方式的项目。



2.2 、添加SpringBoot的父工程

SpringBoot项目可以通过Spring官方配置创建，配置完成的项目可以管理开发所能用到 的几乎所有的依赖， web服务器，测试技术，缓存，数据库整合等等。

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2. 1. 13.RELEASE</version>  </parent> |

2.3 、添加web依赖

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

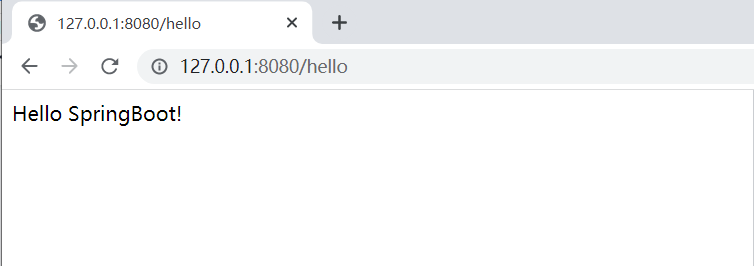
2.4 、编写核心启动类

|  |
| --- |
| /\*\*  \* SpringBootApplication:是SpringBoot的核心启动注解  \*/  @SpringBootApplication  public class SpringApplicationDemo  {  public static void main( String[] args )  {  SpringApplication.run(SpringApplicationDemo.class,args);  }  } |

2.5 、编写Controller返回Hello World

|  |
| --- |
| /\*\*  \* 前端控制器  \*/  @RestController  public class HelloController {  @RequestMapping("/hello")  public String sayHello(){  return "Hello SpringBoot!";  }  } |

2.6、运行项目



三、SpringBoot全局配置文件

**1、全局配置文件**

SpringBoot使用一个全局配置文件，配置文件名是固定的.

application.properties.

application.yml

作用：灵活方便修改Spring Boot自动配置的默认值，SpringBoot框架自动在底层配置

**2、YAML**

YAML是一个可读性高，用来表达数据序列的格式。YAML：以数据为中心进行配置

YAML基本语法

key:(空格) value 表示一对键值对；

以空格的缩进表示层级关系；只要是左对齐的一列数据，都是同一个层级的。

eg:

server:

port: 8080

四、SpringBoot常用注解

**1** 、**SpringBoot**的父工程



选择SpringBoot的artifactId ，可以进入到SpringBoot的项目pom配置文件中：



在SpringBoot项目所依赖的父工程的pom文件中使用properties标签配置了大量的可能

项目中会用的依赖的版本：

|  |
| --- |
| <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> |

同时还使用标签配置了项目中可能会用到的所有的依赖。

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.7.3</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>    <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.11</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <!--springboot整合mybatis的依赖包-->  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>2.2.2</version>  </dependency>  <!--mysql驱动包-->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.45</version>  </dependency>  <!-- https://mvnrepository.com/artifact/commons-codec/commons-codec -->  <dependency>  <groupId>commons-codec</groupId>  <artifactId>commons-codec</artifactId>  <version>1.15</version>  </dependency>  <!--log4j-->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <!-- 关闭此项的依赖传递，即别的项目依赖该项目时，不会传递依赖此jar -->  <optional>true</optional>  </dependency>  <!--项目打成war包，移除内置的tomcat-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> |

**2** 、**SpringBoot**的常用注解

 @SpringBootApplication

此注解是Spring Boot项目的基石，创建SpringBoot项目的Application时会默认加上

|  |
| --- |
| @SpringBootApplication  public class SpringSecurityApplication{  public static void main(Strings[] args){  SpringApplication.run(SpringSecurityApplication,args);  }  } |

可以把@SpringBootApplication 看作@Configuration，@EnableAutoConfiguration， @ComponentScan 注解的集合.其中 @EnableAutoConfiguration：启用SpringBoot的自动配 置机制,@ComponentScan:扫描被@Component /@Service/@Controller注解的bean，注解默认 会扫描该类所在的包下所有类@Configuration:允许在Spring上下文中注册额外的bean或导 入其他配置类.

 @Autowired

@Autowired自动注入注解，最常用的一种注解将对象自动导入到类中，注解自动装配 bean的类

 @Component

@Component:通用注解，当不知道Bean在哪一层时，可以使用@Component注解标注。 @Repository: 对应持久层—Dao层的注解，用于操作数据库相关

@Service: 对应服务层的注解，用来连接Dao层做逻辑处理

@Controller:对应Spring MVC控制层，主要接收用户请求并调用service返回给前端页面

 @RestController

@RestController注解是@Controller注解和@ResponseBody注解的合集，用来返回Json 格式给页面 (带Rest格式的就是返回的Json文本)

 @RequsetMapping

@RequsetMapping是处理HTTP请求的最常用注解

|  |
| --- |
| @RequestMapping("/users")  public ResponseEntity<List<User>> getAllUsers(){  return userRepository.findAll();  } |

 @GetMapping

@GetMapping 就等价于@RequestMapping(value="/users",method

=RequsetMethod.GET)

即使用@GetMapping就相当用接收GET方法了。

|  |
| --- |
| @GetMapping("/users")  public ResponseEntity<List<User>> getAllUsers(){  return userRepository.findAll();  } |

 @PostMapping

@PostMapping 就等价于@RequestMapping(value="/users",method =[RequsetMethod.POST](https://links.jianshu.com/go?to=http://RequsetMethod.POST))

即使用@PostMapping就相当用接收Post方法了。

|  |
| --- |
| @PostMapping("/users")  public ResponseEntity<List<User>> getAllUsers(){  return userRepository.findAll();  } |

@RequestParam

@RequestParam用于获取查询参数。

|  |
| --- |
| @GetMapping("/users/{userId}/teachers")  public List<Teacher> getUserRelatedTeachers(@RequestParam(value = "type",required = false) String type){  ...  } |

在@RequestParam中 value= “参数名” required = “true/false” (true表示参数不允许不存在，false表示参数允 许不存在) defaultValue="" 设置defaultValue时默认required为false。

 @RequestBody

该注解用于读取 Request 请求的 body 部分数据，使用系统默认配置的 HttpMessageConverter 进行解析，然后把相应的数据绑定到要返回的对象上;再把

HttpMessageConverter 返回的对象数据绑定到 controller 中方法的参数上.

注意:

GET、POST方式提交时， 根据 request header Content-Type 的值来判断 :application/x-www-form-urlencoded：可选 (即非必须，因为这种情况的数据 @RequestParam, @ModelAttribute 也可以处理，当然@RequestBody也能处理) ;multipart/form-data：不能处理 (即使用@RequestBody不能处理这种格式的数据) ;其他 格式：必须 (其他格式包括application/json, application/xml等。这些格式的数据，必

须使用@RequestBody来处理) ;

|  |
| --- |
| @RequestMapping(value = "/test", Method = RequestMethod.POST, produces = "application/json;charset="UTF-8")  @ResponseBody  public JSONObject class Test(@RequestBody Test test) {  String name = test.getName();  // 将 name 回显  Map<String, Object> map = new HashMap<>();  map.put("name", name);  JSONObject obj = JSONObject.fromObject(map);  return obj;  } |

@ResponseBody

该注解用于将 Controller 的方法返回的对象，通过适当的 HttpMessageConverter 转 换为指定格式后，写入到 Response 对象的body 数据区。@ResponseBody注解的使用在 返回的数据不是 html 标签的页面，而是其他某种格式的数据时 (如 json、xml 等) 使用

。

|  |
| --- |
| @RequestMapping(value = "/test", Method = RequestMethod.POST, produces = "application/json;charset="UTF-8")  @ResponseBody  public JSONObject class Test() {  JSONObject obj = new JSONObject;  obj.put("test", 1);  // 将 obj 回显 |

|  |
| --- |
| return obj;  } |

五、气温数据分析可视化展示

* 显示国内不同季度的气温折线图

国内不同季度的气温折线图显示效果如下图所示:

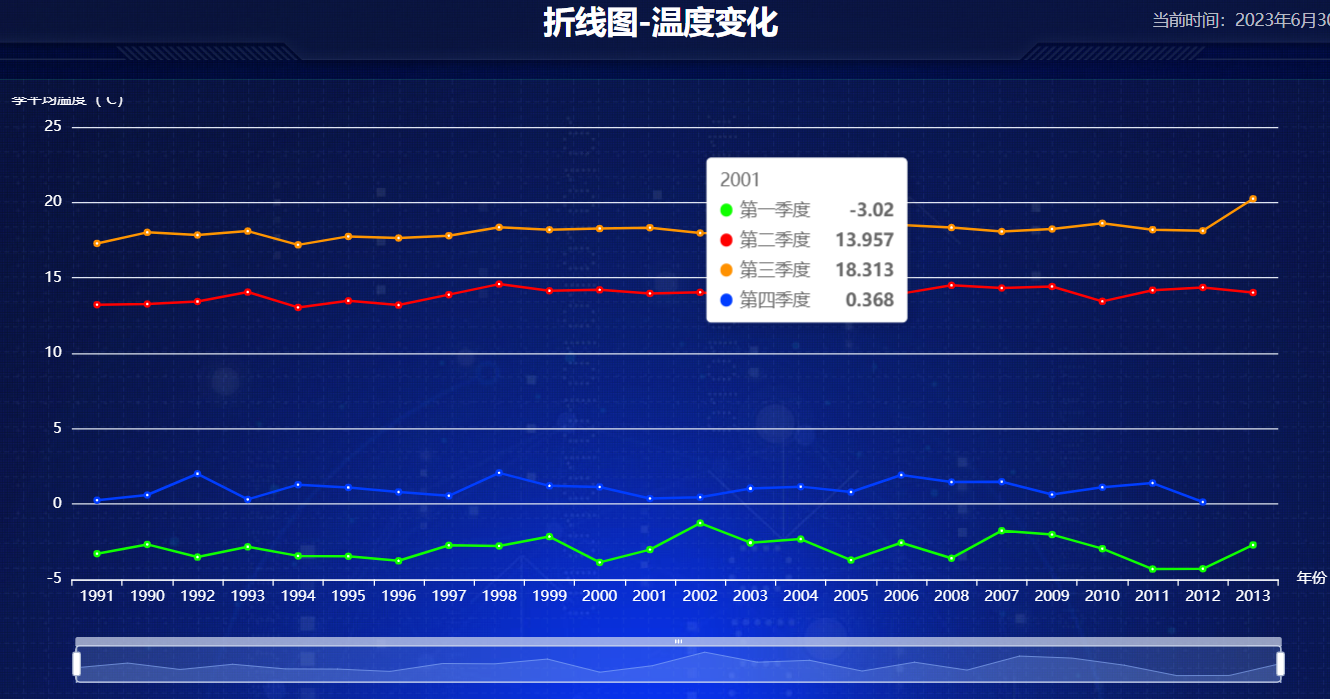


图1-1

1. 在pom.xml文件中添加MyBatis、MySQL的依赖包

|  |
| --- |
| <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>2.2.2</version> </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>5.1.47</scope>  </dependency> |

提示：如果mysql数据库的版本是8.0 ，mysql的版本号应修改为8.\*.\*

1. 在application.yml文件中配置数据源

|  |
| --- |
| server:  port: 8088  spring:  datasource:  driver-class-name: com.mysql.jdbc.Driver  url: jdbc:mysql://127.0.0.1:3306/weatherdb  username: root  password: 123456  devtools:  remote:  restart:  enabled: true  restart:  additional-paths: weatherweb/src/main/java  exclude: static/\*\* |

1. 在appliation.yml文件中配置MyBatis的相关设置

|  |
| --- |
| mybatis:  mapper-locations: classpath:mapper/\*.xml  logging:  level:  org.example.dao: debug |

4)创建HttpResponseEntity类，关键代码如下所示：

|  |
| --- |
| package org.example.beans;  import java.io.Serializable;  /\*\*  \* @ClassName: HttpResponseEntity  \* 实体层：封装前端控制器响应客户端请求的数据格式  \*/  public class HttpResponseEntity implements Serializable {  private String code; //状态码  private Object data; //内容  private String message; //状态消息  public String getCode() {  return code;  }  public void setCode(String code) {  this.code = code;  }  public Object getData() {  return data;  }  public void setData(Object data) {  this.data = data;  }  public String getMessage() {  return message;  }  public void setMessage(String message) {  this.message = message;  }  } |

5）创建TemperaturesMapper类，关键代码如下所示：

|  |
| --- |
| **package** org.example.mapper;  **import** org.example.entity.AreaAccount;  **import** java.util.List;  **public interface** AreaAccountMapper {   //折线图（面积图）显示中国每季度的平均气温  List<Map<String, Object>> selectOfAreaChart();  } |

5）创建TemperaturesController类，关键代码如下所示：

|  |
| --- |
| package org.example.controller;  import org.example.beans.HttpResponseEntity;  import org.example.common.Constants;  import org.example.dao.TemperaturesMapper;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.util.CollectionUtils;  import org.springframework.web.bind.annotation.\*;  import java.util.ArrayList;  import java.util.List;  import java.util.Map;  /\*\*  \* @ClassName: ChartController  \*  \*/  @RestController  @RequestMapping("/temperatures")  public class TemperaturesController {  private final Logger logger = LoggerFactory.getLogger(TemperaturesController.class);  @Autowired  private TemperaturesMapper temperaturesMapper;  /\*\*  \* 功能：折线图（面积图）显示中国每季度的平均气温  \* @return  \*/  @RequestMapping(value="/areaChart",method= RequestMethod.POST, headers = "Accept=application/json")  public HttpResponseEntity getAreaChartData() {  HttpResponseEntity httpResponseEntity = new HttpResponseEntity();  try {  List<Map<String, Object>> springData = temperaturesMapper.selectOfAreaChart();  if(CollectionUtils.isEmpty(springData) ) {  httpResponseEntity.setCode(Constants.EXIST\_CODE);  httpResponseEntity.setData(null);  httpResponseEntity.setMessage(Constants.EXIST\_MESSAGE);  }  else {  httpResponseEntity.setData(springData);  httpResponseEntity.setCode(Constants.SUCCESS\_CODE);  httpResponseEntity.setMessage(Constants.GET\_DATA\_MESSAGE);  }  } catch (Exception e) {  logger.info("获取面积图数据>>>>>>>>>>>" + e.getLocalizedMessage());  httpResponseEntity.setCode(Constants.EXIST\_CODE);  httpResponseEntity.setMessage(Constants.EXIST\_MESSAGE);  }  return httpResponseEntity;  } |

6）前端页面的可视化展示

|  |
| --- |
| const chartDom\_line = document.getElementById('lineChart'); const myChart\_line = echarts.init(chartDom\_line); let option\_line;  commonAjaxPost(true, "../temperatures/areaChart", null, getDataSuccess);  function getDataSuccess(result){  if (result.code === '1') {  const values = result.data;  let data\_spring = [];  let data\_summer = [];  let data\_autumn = [];  let data\_winter = [];  let date = [];  for(let i = 0; i < values.length; i++){  if(values[i]['season'] === 1) {  data\_spring.push(values[i]['averageTemperature']);  if(values[i]['year'] < 2014) {  date.push(values[i]['year'] + "");  }  else {  date.push(values[i]['year'] + "(预测)");  }  }  else if(values[i]['season'] === 2) {  data\_summer.push(values[i]['averageTemperature']);  }  else if(values[i]['season'] === 3) {  data\_autumn.push(values[i]['averageTemperature']);  }  else if(values[i]['season'] === 4) {  data\_winter.push(values[i]['averageTemperature']);  }   }  paintLineChart(data\_spring, data\_summer, data\_autumn, data\_winter, date);  } }  function paintLineChart(data\_spring, data\_summer, data\_autumn, data\_winter, date) {  //折线图  option\_line = {  tooltip: {  trigger: 'axis'  },  grid: {  top: '5%',  containLabel: true  },  xAxis: {  name: '年份',  nameTextStyle: {  color: '#fff'  },  type: 'category',  data: date,  axisLabel: {  color: '#fff'  },  axisLine: {  onZero: false,  lineStyle: {  color: '#fff'  }  }  },  yAxis: {  name: '季平均温度（℃）',  nameTextStyle: {  color: '#fff'  },  type: 'value',  axisLabel: {  color: '#fff'  },  axisLine: {  lineStyle: {  color: '#fff'  }  }  },  dataZoom: [  {  start: 0,  textStyle: {  color: '#fff'  }  },  {  type: 'inside'  }  ],  series: [  {  name: '第一季度',  data: data\_spring,  type: 'line',  lineStyle: {  color: '#0fff00'  },  itemStyle: {  color: '#0fff00'  }  },  {  name: '第二季度',  data: data\_summer,  type: 'line',  lineStyle: {  color: '#ff0000'  },  itemStyle: {  color: '#ff0000'  }  },  {  name: '第三季度',  data: data\_autumn,  type: 'line',  lineStyle: {  color: '#ff9100'  },  itemStyle: {  color: '#ff9100'  }  },  {  name: '第四季度',  data: data\_winter,  type: 'line',  lineStyle: {  color: '#003cff'  },  itemStyle: {  color: '#003cff'  }  }  ]  };   myChart\_line.setOption(option\_line); } |