springboot工程详解

1 什么是springboot?

> Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。(百度百科)

我们做一个简单的缩写：Springboot是一个框架，用来简化某种应用的开发过程。

注意两个词：框架和简化。

所谓"框架"可以理解为库的集合。也就是说：Springboot是一个库的集合。

为什么要有框架？为了"简化"。在计算机网络这门课上我们做过http的请求和相应实验。这个实验做起来并不轻松，即使大体的框架代码已经给出。而http的请求和响应只是web服务器经常使用的一个简单功能而已。也就是说，如果我们做一个项目时，从头开始编写这个项目所需要实现的所以基础功能，那需要耗费的时间和经历是不可接受的，于是程序员就提出了"框架"的概念并搭建了诸如springboot这样的框架来简化项目的开发过程。

这其实和"库"的思想是一致的。库是许多功能函数的集合，框架则是库的集合。框架就是更高维度，更重量级的"库"。

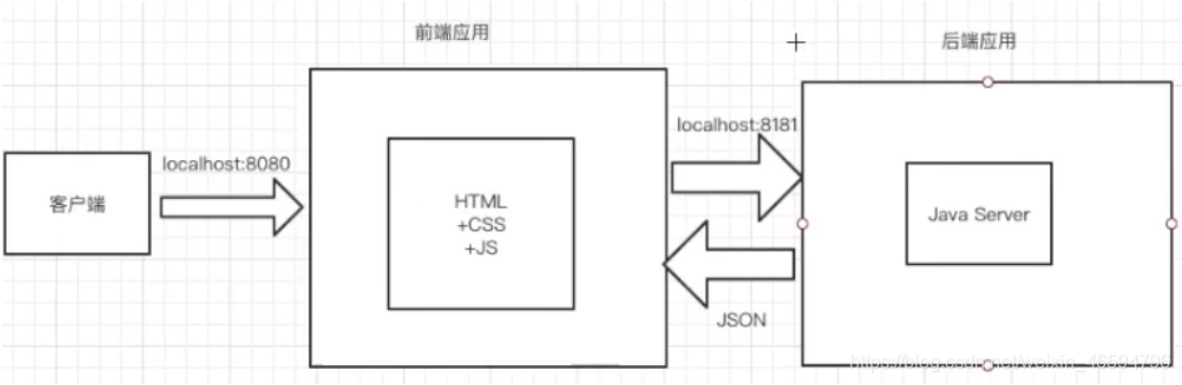
2 用springboot来干什么?

先来看任务书是怎么说的：

> 2. 本项目完成后，学员将能够使用VUE+SpringBoot+AJAX技术开发前后端分离的Web应用程序。

- vue也是一个框架，用于前端html静态页面的开发
- AJAX是一个网页请求技术，能够在不刷新页面的前提下更新部分页面内容
- Springboot用于web服务器的开发

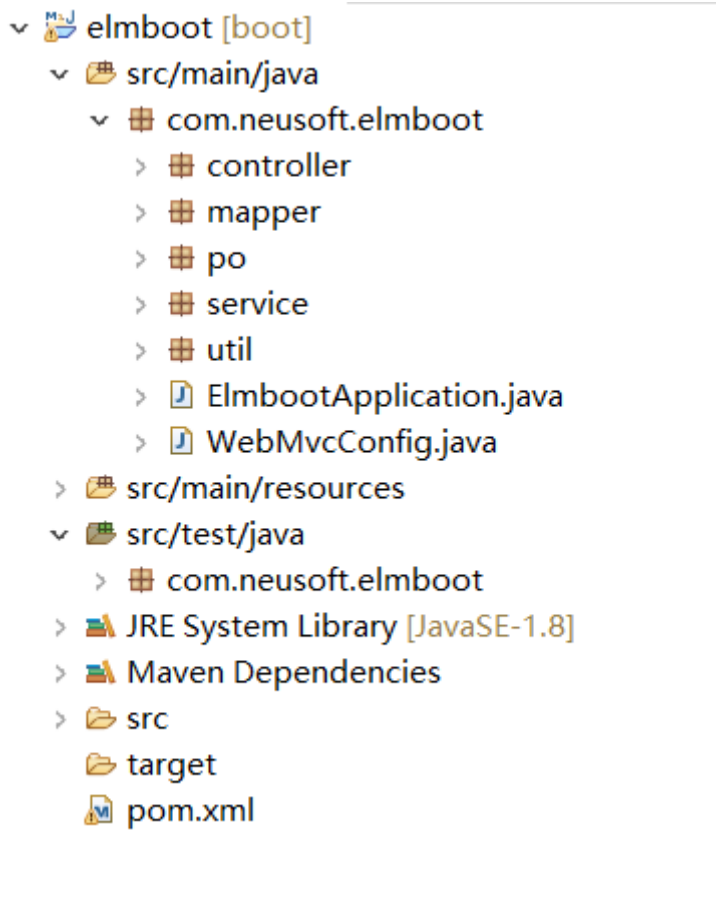
我们来看一张图：



这张图详细解析了前后端分离的逻辑。静态html页面为前端，web服务器和数据库组成后端。

为什么不把web服务器视为“中端”呢？可以，但不合适也没必要。web服务器和数据库是“高度耦合”的，更适合作为整体开发和维护。其实，前端和后端只是一个宽泛的概念。我们也可以把前端进一步细分为前端和后端，甚至前中后三端。后端也一样，我们可以把web服务器视为后端的前端，把数据库视为后端的后端。现在程序员们说的后端和前端只是一种通俗的，或者约定俗成的说法。

3 基于springboot搭建的服务器端项目目录结构



4 设置

1. 如果用的是高版本的mysql，需要在配置依赖时将使用的mysql-connector-java版本也相应上调（目前最新的版本是8.0.20，而课件上的版本是5.几的）。否则会导致程序报错。

```
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.20</version>
```

2. 需要设置数据库时区。我们下载mysql后，默认时区是美国时区，需要调整到大陆的时区，否则会报错。

5 各层功能简析

5.1 po层

po层也叫entity层或model层，用于存放实体类，与数据库中的属性值基本保持一致，实现getter和setter方法

5.2 mapper层

mapper层又名dao层，对数据库进行数据持久化操作，他的方法语句是直接针对数据库操作的，主要实现一些增删改查操作

5.3 service层

业务层，给controller层的类提供接口进行调用。一般就是自己写的方法封装起来，就是声明一下，具体实现在**serviceImpl**中

5.4 controller层

控制层，负责具体模块的业务流程控制，需要调用service逻辑设计层的接口来控制业务流程。因为service中的方法是我们使用到的，controller通过接收前端H5或者App传过来的参数进行业务操作，再将处理结果返回到前端

5.5 其它

5.5.1 pom.xml文件

全称project object model ，项目对象模型。通过xml可扩展标记语言（EXtensible Markup Language）格式保存的pom.xml文件。该文件用于管理：源代码、配置文件、开发者的信息和角色、问题追踪系统、组织信息、项目授权、项目的url、项目的依赖关系等等。

5.5.2 WebMvcConfigurer文件

重写WebMvcConfigurer接口中的addCorsMappings方法以实现跨域请求

5.5.3 application.properties文件

application.properties是一个全局配置文件，作用是对一些默认配置的配置值进行修改

我们的application.properties文件设置了服务器端口号，服务器请求路径，数据库地址，数据库用户名和密码po层目录等信息

5.5.4 ElmbootApplication文件

启动程序

6 服务器运作流程

1. 打开ElmbootApplication文件，启动

```
package com.neusoft.elmboot;

import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ElmbootApplication {

    public static void main(String[] args) {
        SpringApplication.run(ElmbootApplication.class, args);
    }

}
```

经过短暂的等待后，我们的服务器便成功运行在8080端口上了，网页上输入[localhost:8080/elm/BusinessController/listBusinessByOrderTypeId?orderTypeId=1]

打开便可请求到数据库中储存第一列表中所有店铺的ID，地址信息以及商标等图片的编码

这个过程是怎么发生的呢？我们在第三个项目里是明确在前端控制器的代码里引用了controller类的，而在ElmbootApplication里并没有实现这个操作，那么它们的代码是如何被服务器调用起来的呢？

这就不得不提到“注解”了。在以前的编程作业中，我们除了重写代码时，其它时候很少跟注解打交道。但Springboot深度依赖注解来完成配置的自动装配工作。在编写elmboot项目的过程中，我们编写mapper层要加上@Mapper注解，写service层要加上@Service注解，写controller层要加上@RestController以及请求注解@RequestMapping。Elmbootapplication启动时，会调用SpringApplication的run方法，这个方法首先会创建一个springApplication对象，然后再利用创建好的对象调用run方法。在这个调用过程中，springboot会扫描每一个包，检查它们的注解并依靠这些注解给它们分类。

大概的过程就是这样，其实springboot的“注解”运作的过程远比这要复杂臃肿，例如有些注解会涉及到用“注解来注解注解”的套娃式的使用方法。

> SpringBoot 中有大量的注解相关代码，企图理解这些代码是乏味无趣的没有必要的，它只会把你的本来清醒的脑袋搞晕。

2. 现在，程序跑起来了，然后呢？当我们在浏览器发出请求，我们的服务器是如何识别它，并返回相应的信息呢？这个过程和第三个Servlet项目差不多。

```
public class BusinessController {
    @Autowired
    private BusinessService businessService;

    @RequestMapping("/listBusinessByOrderTypeId")
    public List<Business> listBusinessByOrderTypeId(Business business){
```

```
        return
businessService.listBusinessByOrderTypeId(business.getOrderTypeId());
    }

    @RequestMapping("/getBusinessById")
    public Business getBusinessById(Business business){
        return businessService.getBusinessById(business.getBusinessId());
    }

}
```

首先，根据端口号，请求信息发到我们的服务器，服务器解析/elm和记录在application.properties中的server.servlet.context-path的值一致，再解析/BusinessController，定位到Controller层下的BusinessController类，根据/listBusinessByOrderTypeId定位到listBusinessByOrderTypeId方法，?orderTypeId=1说明orderTypeId =1。BusinessController调用BusinessService，BusinessService文件里只定义了方法名，具体实现由BusinessServiceImpl完成。BusinessServiceImpl会进一步调用BusinessMapper里的listBusinessByOrderTypeId方法

```
public class BusinessServiceImpl implements BusinessService{

 @Autowired
 private BusinessMapper businessMapper;
 @Override
 public List<Business> listBusinessByOrderTypeId(Integer orderTypeId) {
 return businessMapper.listBusinessByOrderTypeId(orderTypeId);
 }

 @Override
 public Business getBusinessById(Integer businessId) {
 return businessMapper.getBusinessById(businessId);
 }
}
```

BusinessMapper根据定义好的固定操作语句从数据库里调取business表中ordertypeId=1的数据返回给Service层，Service层再返回给BusinessController，最后java web服务器应用将获取的数据回复给前端网页应用。

```
public interface BusinessMapper {

    @Select("select * from business where orderTypeId=#{orderTypeId}")
    public List<Business> listBusinessByOrderTypeId(Integer orderTypeId);

    @Select("select * from business where businessId=#{businessId}")
    public Business getBusinessById(Integer businessId);
}
```

其它组件功能的实现思路大致相同，在此不一一列举。