**海**

**马**

**项**

**目**

**开**

**发**

**文**

**档**

编写人：李培磊，白疆银

日期：2019年6月28日

1. **前言**

**1.1 文档说明**

本文档重点阐述 sea.horse项目使用JavaEE 三层架构的思想及简要说明各层代码的意义，运用springmvc搭建后台框架，vue.js+element UI搭建前段框架，实现前后端分离，简单交互的讲解。文档只包含了重要步骤及截图，具体配置时以实际操作为主。

**1.2 MVC设计模式**

MVC 的全称是 Model-View-Controller，即“模式—视图—控制器”， 其思想是：首先用户通过视图层发出请求，接着控制器接受用户的请求，并决定调用哪个模型来进行处理，然后模型用业务逻辑来处理用户的请求并返回数据，最后控制器将处理后的数据传递给视图层，通过视 图层将结果展现给用户。其各个层次具体功能如下：

1、Model（模型）

模型代表了 web 中的核心功能，包括业务逻辑层和数据库访问层。在 Java Web 应用中，业务逻辑层一般有 JavaBean 或 EJB 构建。数据访问层（数 据持久层）则通常应用 JDBC 或 Hibernate 构建，主要负责于数据库打交 道，如从数据库中取数据、向数据库中保存数据等。

2、View（视图）

视图主要指用户看到并与之交互的界面，即 Java Web 应用程序的外观。 视图部分一般由 JSP 和 HTML 构建，视图可以接受用户的输入、但并不 包含任何实际的业务处理，只是将数据转交给控制器。在模型改变时， 通过模型和视图之间的协议，视图得知这种改变并修改自己的显示。对 于用户的输入，视图将其交给控制器进行处理。

3、Controller（控制器）

控制器负责交互和将用户输入的数据导入模型。在 Java Web 应用中，当 用户提交 HTML 表单时，控制器接受请求并调用相应的模型组件去处理 请求，之后调用相应的视图来显示模型返回的数据

**1.3 文档使用对象**

具备JavaSE基础的JavaEE开发人员

1. **项目开发工具**

**2.1 开发平台**

Microsoft Windows 10

**2.2 所需软件**

|  |  |  |  |
| --- | --- | --- | --- |
| **软件名称** | **操作系统位数** | **下载url** | **软件作用** |
| 64位 |  |  |
| Jdk、jre | jdk-8u181-windows-x64.exe | http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html | Java开发工具及运行环境 |
| IntelliJ IDEA | ideaIU-2018.2.1.exe | https://www.jetbrains.com/idea/ | Java开发IDE |
| Vue.js |  | https://cn.vuejs.org/v2/guide/installation.html | 构建用户界面的**渐进式框架** |
| Maven | apache-maven-3.5.3-bin.tar.gz | http://maven.apache.org/download.cgi | 项目的构建和管理 |
| Element-ui |  | http://element-cn.eleme.io/#/zh-CN/guide/design | 配合vue.js进行开发的ui框架 |

**第三章 项目环境搭建**

**3.1 安装 jdk、jre**

**3.1.1 安装 jdk、jre**

双击 jdk-9.0.4\_windows-x64\_bin，设置 jdk 和 jre 的安装路径分别为

D:\Java\jdk-9.0.4\ 和 D:\Java\jre-9.0.4\。出现如下图所示的提示界面，则说明

jdk、jre 安装成功。



**3.1.2 配置 jdk、jre**

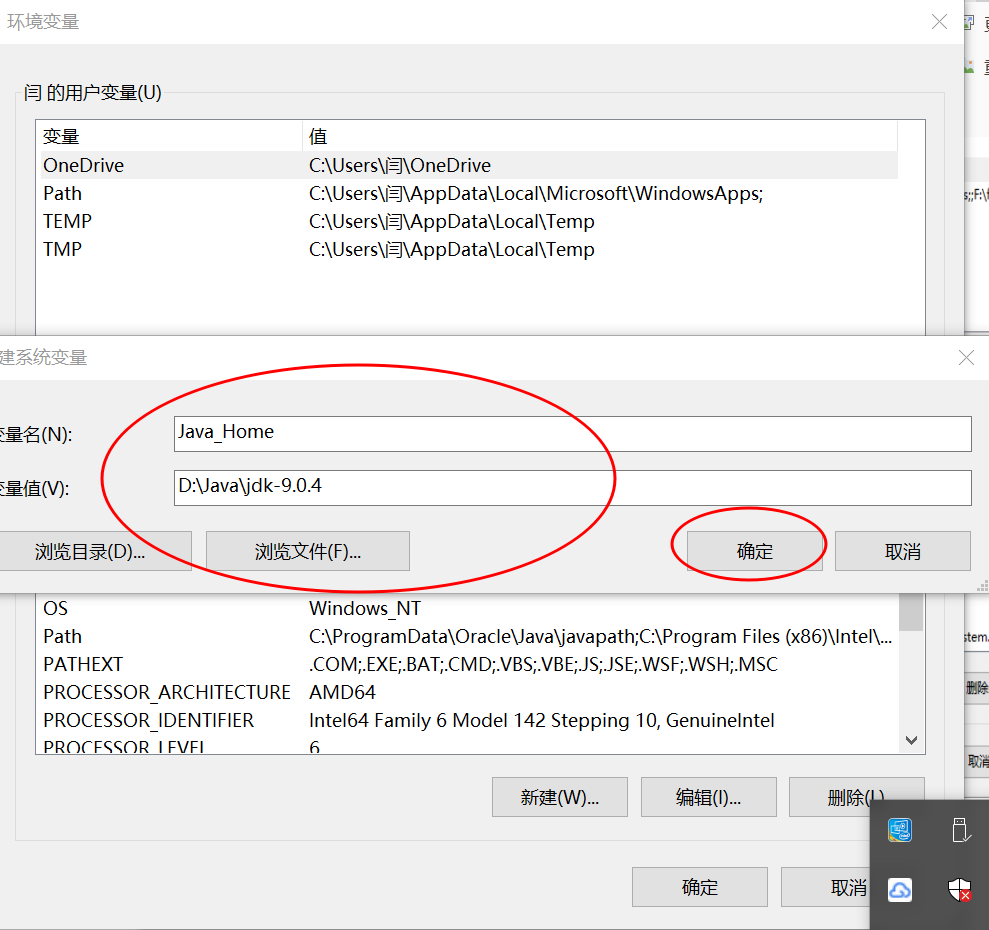
dk、jre 安装成功后，需进行环境变量的配置。配置过程如下所示：

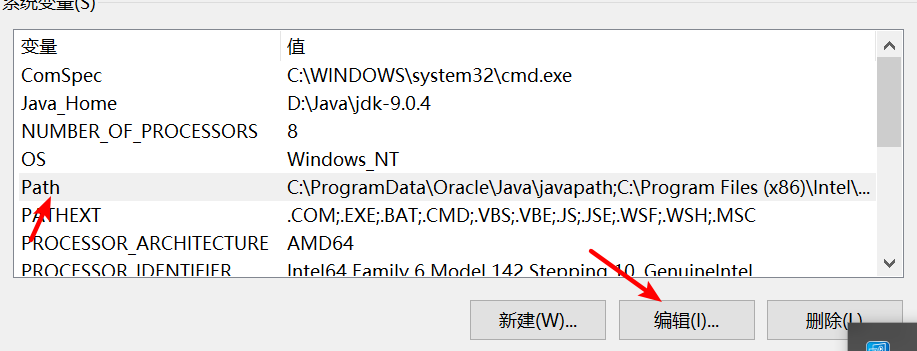
1、右键单击“我的电脑” “高级系统设置” “环境变量” “系统变

量”



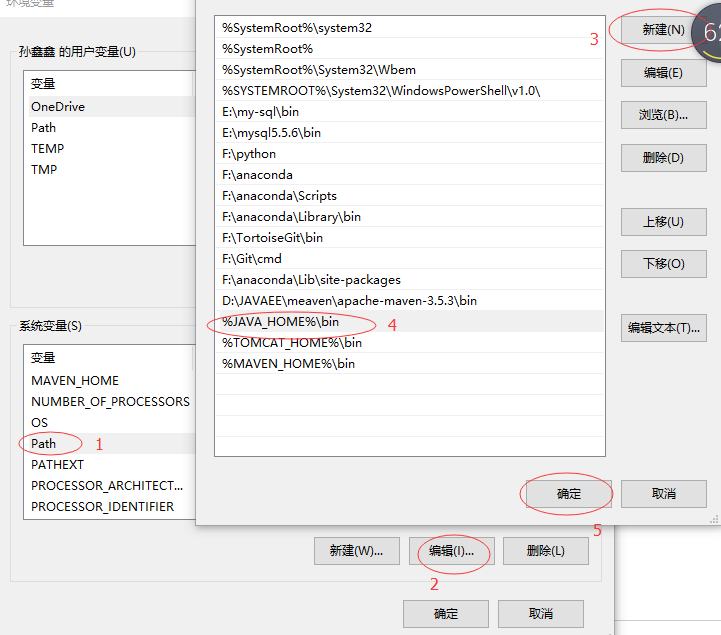
2、点击“新建”，在弹出的对话框中依次填写表 2-2 的配置信息，这里以 Java\_Home 的配置为例，如下所示：





按如下表配置

|  |  |
| --- | --- |
|  | 表格 2-2 配置信息 |
|  |  |
| **环境变量名** | **环境变量值** |
|  |  |
| JAVA\_HOME | D:\Java\jdk-9.0.4 |
|  |  |
|  |  |
| PATH | …;%Java\_Home%\bin |



**3.1.3 验证 jdk 是否配置成功**

在命令行中执行如下 java -version 命令，若出现如下窗口（显示版本信息），

则说明 jdk 配置成功。

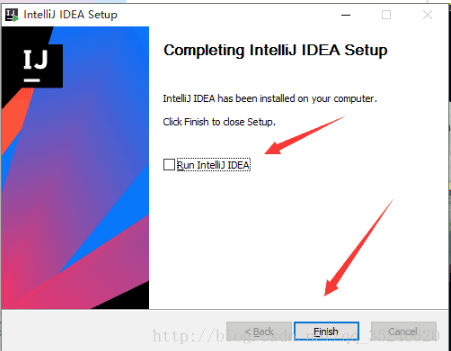


建议在 Linux 下安装一次 jdk、jre，以进一步体会 jdk、jre 的配置意义和作用。

**3.2 安装 IDEA**

可以从官方网站下载最新版本的 IDEA，具体网址https://www.jetbrains.com/，

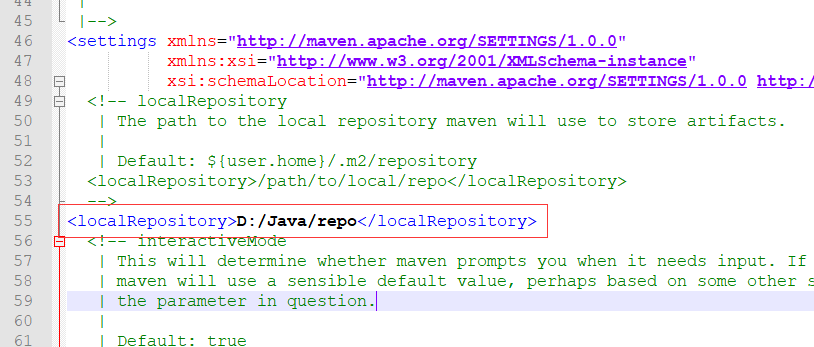
IDEA为收费软件，具体破解可以参照下面的博客https://www.cnblogs.com/zqy123455/articles/7903385.html，可以自定义安装路径，但本文建议安装把所有开发软件都放在建立的大文件夹JAVA文件下，安装后如图



**3.3 配置Maven**

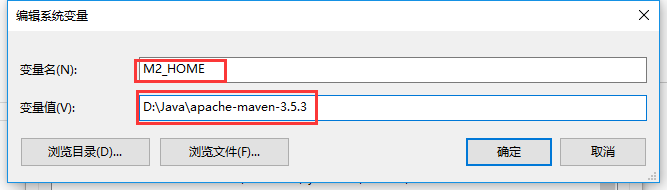
**3.3.1 下载Maven**

下载解压Maven到D:\Java\apache-maven-3.5.3，新建仓库D:\Java\repo，打开D:\java\apache-maven-3.5.3\conf\settings.xml，修改如下位置：

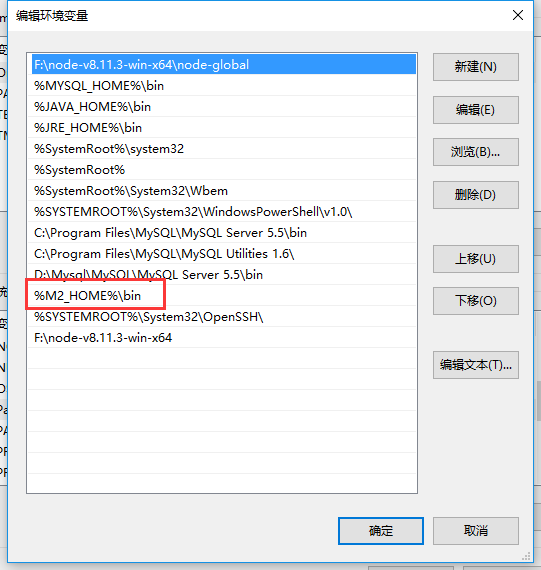


**3.3.2 配置Maven环境变量**

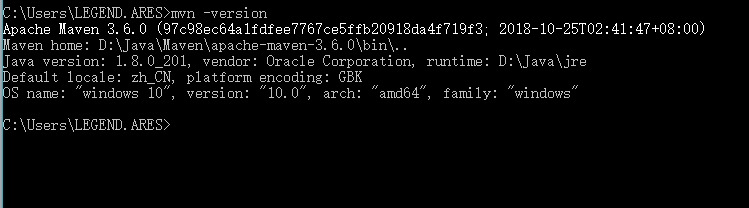
1、新建 MAVEN\_HOME 变量



2、修改Path变量，在其后加上%M2\_HOME%\bin;

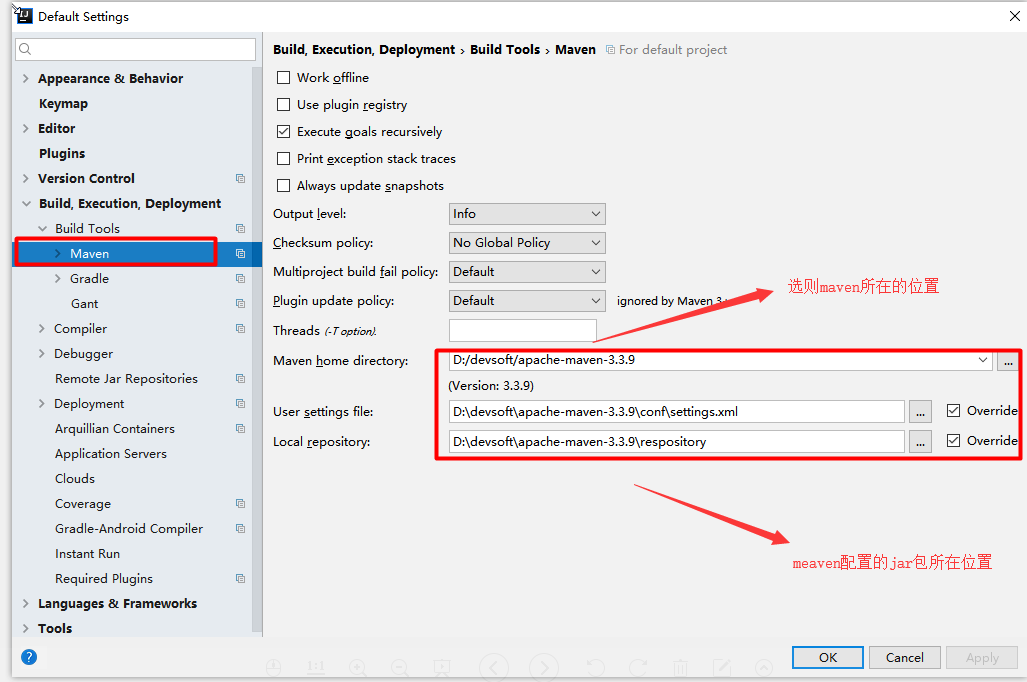


3、在cmd中输入mvn –version验证是否安装成功。

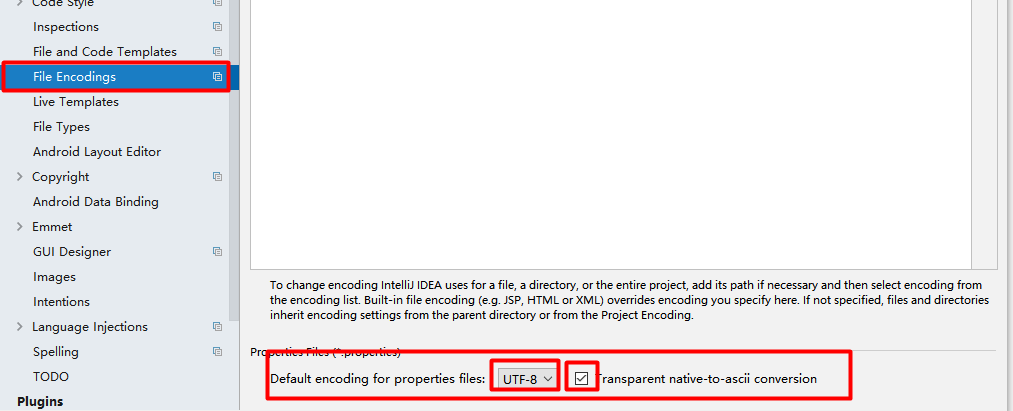


**3.3.3 在IDEA中配置Maven**

打开IDEA->File->Settings->Maven，操作如图，点击OK配置完成



设置统一的UTF-8编码，防止出现乱码现象



**3.4 安装Vuejs**

**3.4.1 安装npm、webpack、vue-cli、elemet-ui、resource、router**

**1.**npm: Nodejs下的包管理器。

**2.**webpack: 它主要的用途是通过CommonJS的语法把所有浏览器端需要 发布的静态资源做相应的准备，比如资源的合并和打包。

**3.**vue-cli: 用户生成Vue工程模板。（帮你快速开始一个vue的项目，也就是给你一套vue的结构，包含基础的依赖库，只需要 npm install就可以安装）

**4.**elemet-ui：element-ui是一个ui库，它不依赖于vue。但是却是当前和vue配合做项目开发的一个比较好的ui框架。

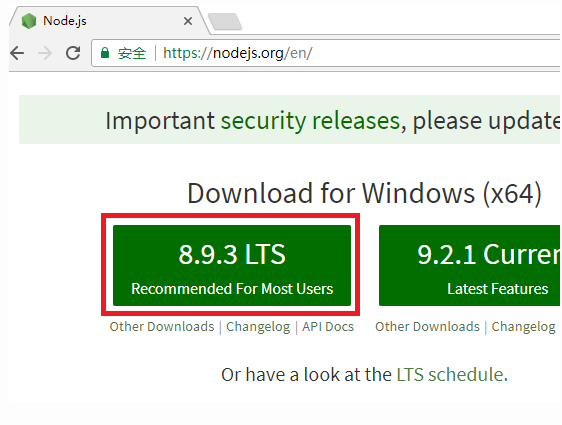
**5.**axios：axios是基于promise用于浏览器和node.js是http客户端。axios主要用于向后台发起请求的，还有在请求中做更多的可控功能，promise是一个对象用来传递异步操作的信息，它代表了某个未来才会知道结果的事件（通常是一个异步操作），并且这个事件提供统一的api，可供进一步的处理, Promise的出现主要是解决地域回调的问题，比如你需要结果需要请求很多个接口，这些接口的参数需要另外那个的接口返回的数据作为依赖，这样就需要我们一层嵌套一层，但是有了Promise 我们就无需嵌套

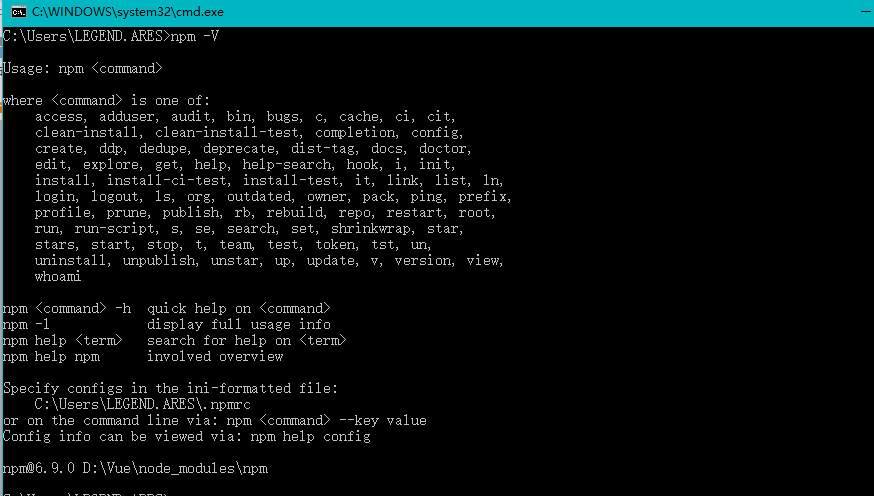
**6.**router：Vue Router 是 [Vue.js](http://cn.vuejs.org/" \t "https://router.vuejs.org/zh/_blank) 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建单页面应用变得易如反掌。包含的功能有：

* 嵌套的路由/视图表
* 模块化的、基于组件的路由配置
* 路由参数、查询、通配符
* 基于 Vue.js 过渡系统的视图过渡效果
* 细粒度的导航控制
* 带有自动激活的 CSS class 的链接
* HTML5 历史模式或 hash 模式，在 IE9 中自动降级
* 自定义的滚动条行为

**3.4.2 安装npm，即安装node.js**

Node.js的下载网址https://nodejs.org/en/，node.js会加入默认的环境配置，一路默认next，最后cmd检查





在cmd中继续输入以下命令

npm config set prefix "D:\nodejs\node\_global"

npm config set cache "D:\nodejs\node\_cache"

npm config set registry=http://registry.npm.taobao.org 配置镜像

npm config get registry 检查镜像站是否可用

1. 配置nodejs环境变量

增加环境变量NODE\_HOME为D:\nodejs\node\_global\node\_modules

编辑环境变量Path为D:\nodejs\node\_global

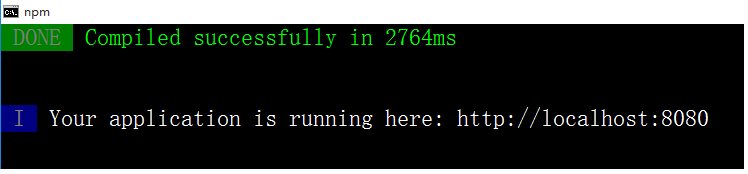
1. 安装webpack、vue、vue-cli

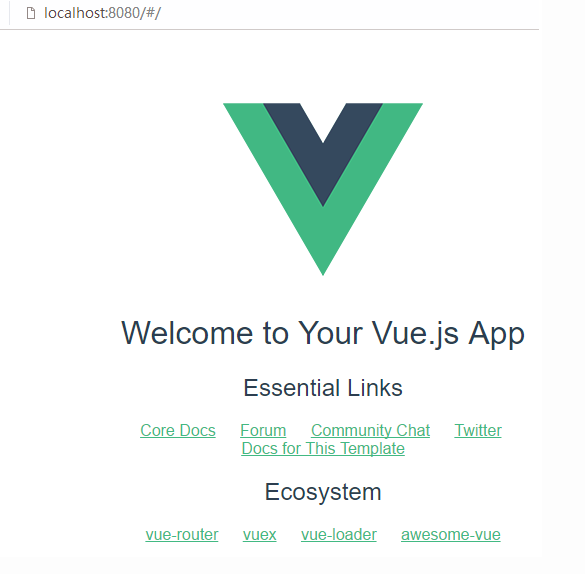
重新打开cmd安装vuejs

npm install vue vue-router vue-cli axixos vuex –g

1. **初始化Vuejs项目测试是否安装成功，步骤如下，最后在浏览器打开8080窗口，出现界面如下即成功**







**3.4.3配置element-ui**

**这里作为插件引入，步骤如下**

进入项目所在目录输入IMG_256引入如下图红色字体然后再项目中即可直接使用，使用方法参见element-ui官方网站http://element.eleme.io/#/zh-CN

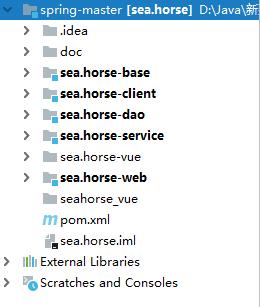
**3.4.3配置router**

进入项目所在目录输入npm install vue-router --save，使用方法参见官方文档https://router.vuejs.org/zh/installation.html

**第四章 整体项目概述**

**4.1后端项目概述**

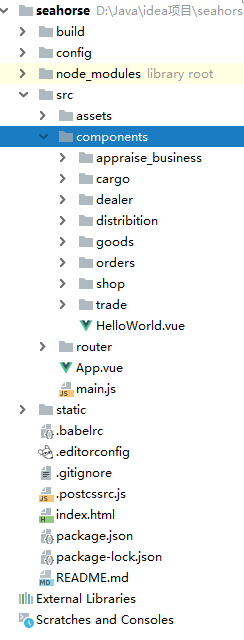
本项目分为三个组，第一组为前端组，第二组为图像分析组，第三组为文字分析组，所以建立项目如图所示，提取公共部分为base组



每组建立项目结构如图，doc放置组员日志，base配置公共jpa，dao层放置实体类和实体类接口，即数据访问层，完成持久化。service放置逻辑方法，业务层，web层放置展示层/

**4.2前端项目概述**

本项目根据后台需求设计相应的前端功能，完成交互。



**第五章 项目命名规范**

**5.1 包的书写规范Package**

**5.1.1 一般要求**

1、选择有意义的名字，能快速的传达该类的用途。

2、所有包的命名必须采用小写英文字母。但是由于 Java 面向对象编程的特性，每一名 Java 程序员都可以编写属于自己的 Java 包，为了保障每个 Java 包 命名的唯一性，在最新的 Java 编程规范中，要求程序员在自己定义的包的 名称之前加上唯一的前缀。 由于互联网上的域名称是不会重复的，所以程序 员一般采用自己在互联网上的域名称作为自己程序包的唯一前缀。

**5.1.2 实际应用**

应用系统中经常应用分层，Dao 层（数据访问层）、service（业务处理）、Web 层（页面控制 action 类）。包名的前几个为固定的，如果是网站的话，采用网站 的域名的反写；如果域名还没有确定的话，采用固定的名称。如：net.school 在包名接下来一个单词为模块的名称。如：用户模块，包名为：net.school.user 关于模块的访问操作，采用分层形式，一般分为：

①、Dao 层操作：一般定义在 net.school.xxx.dao 中，其中 xxx 为模块名称；

②、Service 层操作：一般定义在 net.school.xxx.service 中，其中 xxx 为模 块名称；

③、Web 层：般定义在 net.school.xxx.action 中，其中 xxx 为模块名称；

**5.2 类名的书写规范Class**

根据约定，Java 类名通常以大写字母开头，如果类名称由多个单词组成， 则每个单词的首字母均应为大写例如 TestPage；如果类名称中包含单词缩写， 则这个所写词的每个字母均应大写，如：XMLExample，还有一点命名技巧就是 由于类是设计用来代表对象的，所以在命名类时应尽量选择名词。 例如： Graphics

**5.2.1 一般要求**

1、选择有意义的名字，能快速地传达该类的用途。

2、参照 java 驼峰命名法，类名的首字母必须采用大写的形式，如果类名为多 词 组 合 而 成 的 话 ， 那 么 每 个 词 的 首 字 母 必 须 采 用 大 写 。 如 ： StudentAnswer.java。

3、当要区别接口类和实现类的时候，可以在类的后面加上“Impl”。如：接口 类：UserInterface.java 接口实现类：UserInterfaceImp。

4、推荐实体类没有后缀名。

**5.2.2 实际要求**

应用系统中经常应用分层，Dao 层（数据库访问）、Service 层（业务处理）、 Web 层（页面控制 action 类），每一层的类的名称尽量带上该层后缀。

1、Dao 层

I、接口类：采用 xxx+Interface+Dao 的形式来定义,即实体对象 +Interface+Dao。如：用户对象接口类： UserInterfaceDao，其中 xxx 为模块名称。

II、实现类：采用 xxx+Interface+Impl+Dao 的形式来定义, 即实体对象 +Interface+Impl+Dao 如：用户对象实现类：UserInterfaceImplDao

2、Service 层

I、接口类：采用 Xxx+Interface+Service 的形式来定义,即，模块 +Interface+Service。如：用户管理接口类：UserMsgInterfaceServiec

II、实现类：采用 Xxx+Interface+Impl+Service 的形式来定义,即，模块 +Interface+Impl+Servic。如：用户管理实现类： UserMsgInterfaceImplServiec

3、Web 层（action 类）

I、实现类：采用县 Xxx+Operator+Action 的形式来定义,即，模块+操作 +Action 。如用户模块 User+ 删除操作Delete+Action = UserDeleteAction

**5.3 方法的命名**

方法的名字的第一个单词应以小写字母作为开头，后面的单词则用大写字 母开头。例如：drawImage

**5.3.1 一般要求**

1、选择有意义的名字，能快速地传达该方法的用途。

2、参照 java 驼峰命名法，首字母以小写开头，每个单词首字母大写（第一个 单词除外）。

**5.3.2 实际要求**

1、方法表示一种行为，它代表一种动作，最好是一个动词或者动词词组或者 第一个单词为一个动词。

2、属性方法：以 get/set 开头，其后跟字段名称，字段名称首字母大写。如： getUserName()

3、数据层方法：只能以 insert（插入）， delete（删除），update（更新）， select（查找），count（统计）开头，其他层方法避免以这个 5 个开头，以 免造成误解。

4、服务层方法，根据方法的行为命名，只描述方法的意义，而不采用方法的目 的命名。比如系统的添加新用户，用户可以前台注册，也可以管理员后台添 加，方法会被重用，所以最好不要用使用 register，采用 add 会更好写。避 免使用与 web 层相关的方法。

5、Web 层方法最好是贴近 web 的语言，如 register，login，logout 等方法。

**5.4 常量的命名**

常量的名字应该都是用大写字母，并指出该常量完整含义。如果一个常量 名称由多个单词组成，则应该用下划线来分割这些单词。例如：MAX\_VALUE。

**5.4.1 一般要求**

1、选择有意义的名字，能快速地传达该变量的用途。

2、参照 java 驼峰命名法，采用全部大写的形式来书写，对于采用多词合成的 变量采用“\_”来连接各单词。如：USER\_LIST

**5.5 参数的命名**

参数的命名规范和方法的命名规范相同，而且为了避免阅读程序时造成迷 惑， 、请在尽量保证参数名称为一个单词的情况下使参数的命名尽可能明确。

**5.5 Javadoc的注释**

Java 除了可以采用我们常见的注释方式（//、/\* \*/）之外，Java 语言规 范还定义了一种特殊的注释，也就是我们所说的 Javadoc 注释，以/\*\*开头，而 以\*/结束，Javadoc 注释可以被自动转为在线文档，省去了单独编写程序文档 的麻烦。 推荐使用。Javadoc 注释主要涉及范围：类、属性、方法。 例如：

**package** pioneer.seahorse.goods.dao;  
**import** org.apache.logging.log4j.LogManager;  
**import** org.apache.logging.log4j.Logger;  
**import** org.junit.Test;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** pioneer.seahorse.base.dao.GenericTreeDaoTestCase;  
**import** pioneer.seahorse.goods.domain.Management;  
**import** java.util.List;  
**import static** org.junit.Assert.*assertEquals*;  
  
*/\*\*  
 \* Management测试  
 \** ***@author*** *lpl  
 \** ***@Date*** *2019/4/13 09:32  
 \*/***public class** ManagementDaoTest **extends** GenericTreeDaoTestCase<Long, Management, ManagementDao> {  
 */\*\*  
 \* Logger for this class  
 \*/* **private static final** Logger ***logger*** = LogManager.*getLogger*(ManagementDaoTest.**class**.getName());  
  
 ManagementDao **managementDao**;  
  
 @Autowired  
 **public void** setManagementDao(ManagementDao managementDao) {  
 **this**.**managementDao** = managementDao;  
 **this**.**dao** = **this**.**managementDao**;  
 }  
  
 @Test  
 **public void** testGetRoot() {  
 **int** root\_size = 3;  
 */\* for (int i = 0; i < root\_size; i++) {  
 Management management = new Management();  
 management.setName("management\_" + i);  
 for (int j = 0; j < 3; j++) {  
 Management s = new Management();  
 s.setName("management\_" + i + "\_" + j);  
 s.setParent(management);  
 }  
 this.managementDao.save(management);  
 }\*/* List<Management> roots = **this**.**managementDao**.getRoot();  
 *//assertEquals(root\_size, roots.size());* **if** (***logger***.isInfoEnabled()) {  
 ***logger***.info(**"testGetRoot() - List<Management> roots={}"**, roots); *//$NON-NLS-1$* }  
 }  
}

**5.7 变量的命名**

**5.7.1 一般要求**

1、选择有意义的名字，能快速地传达该变量的用途。 2、参照 java 驼峰命名法，首字母以小写开头，每个单词首字母大写（第一个 单词除外）。

**5.7.2 实际要求**

1、变量命名采用基本结构为 typeVariableName，使用 3 字符前缀来表示数据类 型。例如，定义一个整形变量：intDocCount，其中 int 表明数据类型，后面、 为表意的英文名，每个单词首字母大写。

2、变量使用技巧：

a、在一段函数中不使用同一个变量表示前后意义不同的两个数值。

b、除非是在循环中，否则一般不推荐使用单个字母作为变量名，i、j、k 等只 作为小型循环的循环索引变量。

c、避免用 Flag 来命名状态变量。

d、用 Is 来命名逻辑变量，如：blnFileIsFound。通过这种给布尔变量肯定形 式的命名方式，使得其它开发人员能够更为清楚的理解布尔变量所代表的意 义。

e、如果需要对变量名进行缩写时，一定要注意整个代码中缩写规则的一致性。 例如，如果在代码的某些区域中使用 intCnt，而在另一些区域中又使用 intCount，就会给代码增加不必要的复杂性。建议变量名中尽量不要出现缩写。

**第六章 springmvc框架技术**

**6.1Spring MVC 入门**

**6.1.1、Spring MVC 简介**

> 简化Spring应用开发的一个框架；

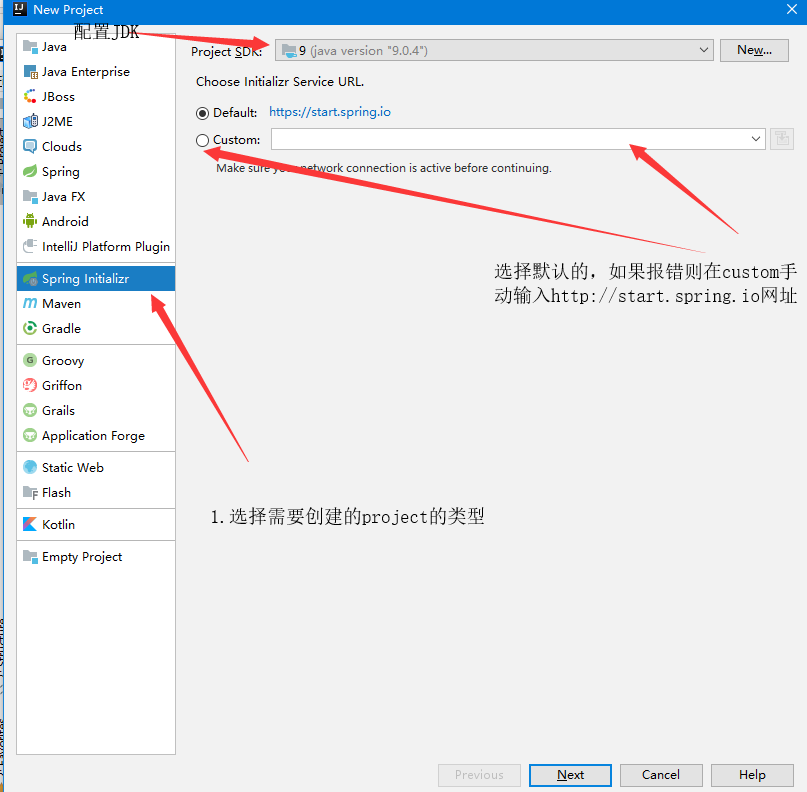
>

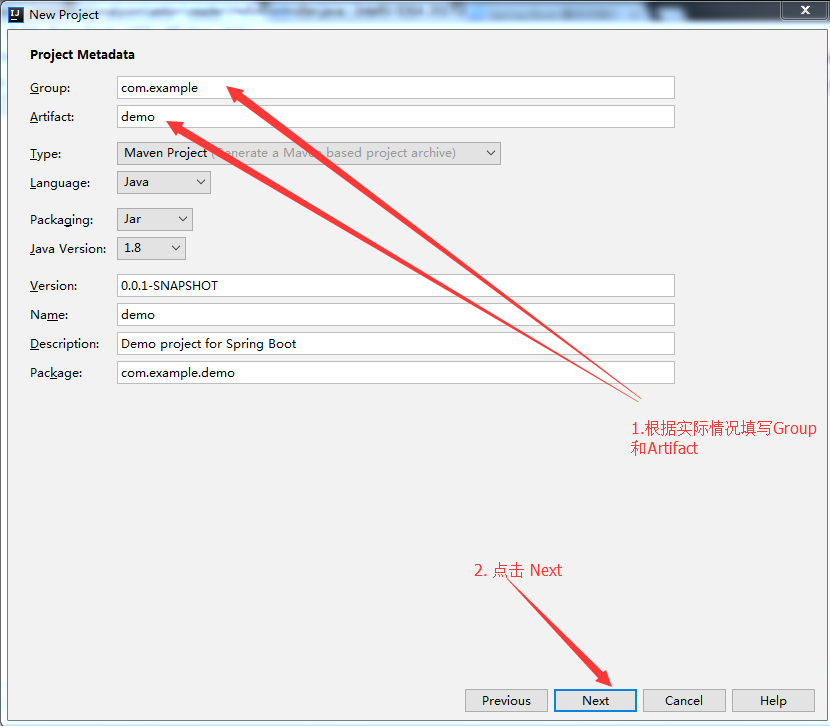
> 整个Spring技术栈的一个大整合；

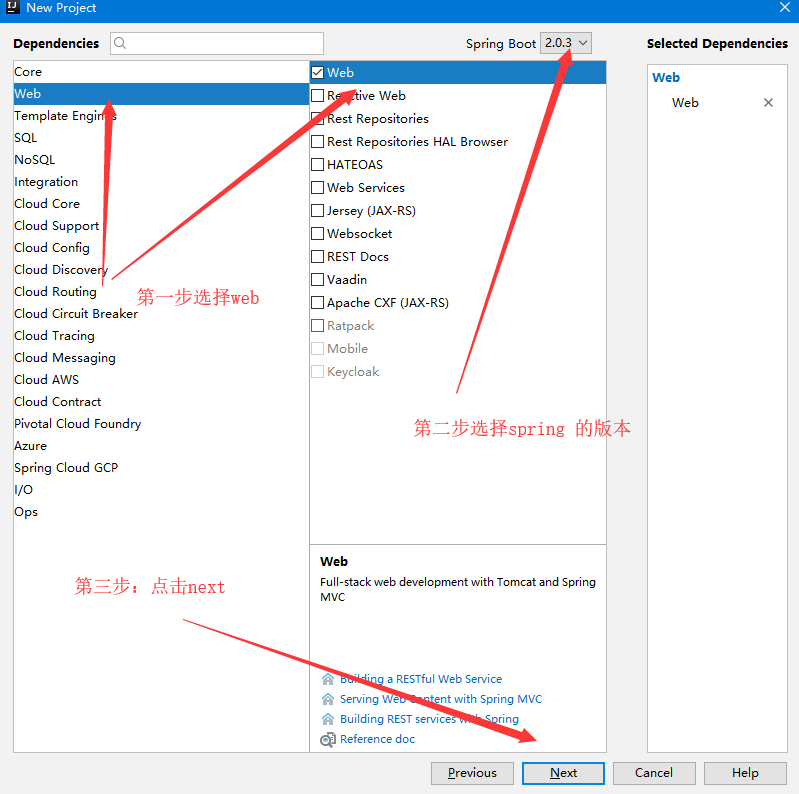
>

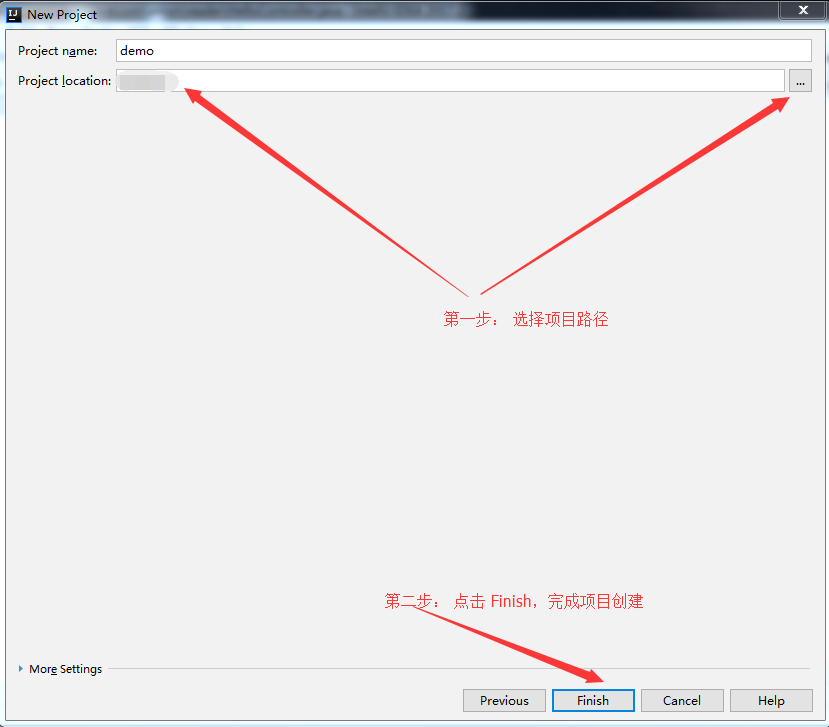
> J2EE开发的一站式解决方案；

**6.2 springmvc创建项目**







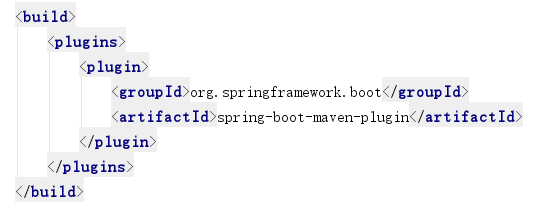


**6.2 pom.xml配置**

1.在新建的Springmvc项目中找到pom.xml添加以下依赖：



2. 加入如下插件



**6.3 [Spring mvc配置文件application](https://www.cnblogs.com/aston/p/7237039.html)**

配置文件有两种application.properties 和 application.xml。便于观看，选择application.xml树形结构，本次以修改端口号为例，展示如何使用，建立application.xml文件，springmvc有默认读取resources文件下的config文件夹下的application.xml，如果没有，请自行建立

application.xml

server:

        port: 8080

**6.3.1xml的语法格式**

标记语言：

以前的配置文件；大多都使用的是 \*\*xxxx.xml\*\*文件；

XAML：\*\*以数据为中心\*\*，比json、xml等更适合做配置文件；

1.XAML：配置例子

```xaml

server:

port: 8081

```

2.XML：配置例子

```xml

<server>

<port>8081</port>

</server>

```

2、XAML语法：

**1、基本语法**

1.k:(空格)v：表示一对键值对（空格必须有）；

以\*\*空格\*\*的缩进来控制层级关系；只要是左对齐的一列数据，都是同一个层级的

*```xaml*

*server:*

*port: 8081*

*path: /hello*

*```*

2.属性和值也是大小写敏感；

**2、值的写法**

1.字面量：普通的值（数字，字符串，布尔）

2.k: v：字面直接来写；

3.字符串默认不用加上单引号或者双引号；

4.""：双引号；不会转义字符串里面的特殊字符；特殊字符会作为本身想表示的意思

5.''：单引号；会转义特殊字符，特殊字符最终只是一个普通的字符串数据

6对象、Map（属性和值）（键值对）：

7: v：在下一行来写对象的属性和值的关系；注意缩进

8对象还是k: v的方式

*```Xaml*

*friends:*

*lastName:*

*age: 20*

*```*

10.行内写法：

*```xaml*

*friends: {lastName: ,age: 18}*

*```*

数组（List、Set）：

用- 值表示数组中的一个元素

```xaml

pets:

- cat

- dog

- pig

```

11.行内写法

```xaml

pets: [cat,dog,pig]

```

**6.4 配置SpringDataJPA**

**6.4.1 springmvc-data-jpa配置**

1.在pom.xml中添加如下依赖

<!-- JPA依赖 -->

<dependency>

<groupId>org.springframework.mvc</groupId>

<artifactId>spring-mvc-starter-data-jpa</artifactId>

</dependency>

<!-- mysql依赖 -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

2.在src/main/resources下配置数据库信息：

spring:

datasource:

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3305/test

username: root

password: root

jpa:

hibernate:

ddl-auto: update

show-sql: true

 a.create：每次加载hibernate时都会删除上一次的生成的表，然后根据你的model类再重新来生成新表，哪怕两次没有任何改变也要这样执行，这就是导致数据库表数据丢失的一个重要原因

b.create-drop：每次加载hibernate时根据model类生成表，但是sessionFactory一关闭,表就自动删除

c.update：最常用的属性，第一次加载hibernate时根据model类会自动建立起表的结构（前提是先建立好数据库），以后加载hibernate时根据model类自动更新表结构，即使表结构改变了但表中的行仍然存在不会删除以前的行。要注意的是当部署到服务器后，表结构是不会被马上建立起来的，是要等应用第一次运行起来后才会

d.validate：每次加载hibernate时，验证创建数据库表结构，只会和数据库中的表进行比较，不会创建新表，但是会插入新值

e.none：没有配置

3.建立实体类

@Table(name="UserRole\_Table")

@Entity

public class UserRole extends BaseEntity {

@Column(name="Authority" ,nullable=false)

private String authoritya;

@Column(name="UserName",unique = true,nullable=false)

private String userName;

@Column(name="Password", nullable=false)

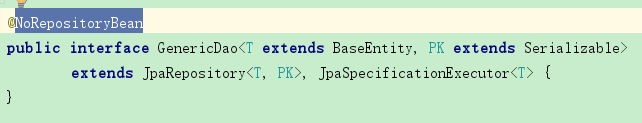
private String password;

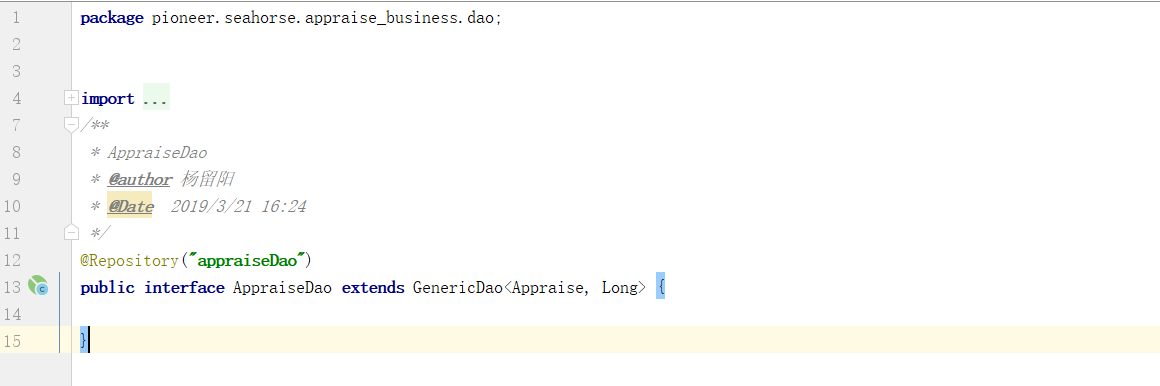
@Column(name="Descrip", nullable=false)

private String descrip;在调用 get//set//方法

启动项目，会自动创建数据表。

**6.4.2 应用**





代码解释：

1. 注解：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 注解名 | 注解（属性）说明 | 备注 |
| 1 | @NoRepositoryBean | 一般用作父类的repository，有这个注解，spring不会去实例化该repository。 |  |
| 2 | @Repository | 只能标注在 DAO 类上。该注解的作用不只是将类识别为 Bean，同时它还能将所标注的类中抛出的数据访问异常封装为 Spring 的数据访问异常类型 |  |
| 3 | @Query | 自定义数据库操作语句具体用法可以参见https://www.imooc.com/video/14548 |  |
| 4 | @Table | 所写参数为映射到数据库的表名 |  |
| 5 | @Entity | 实体类 |  |
| 5 | @Column | 映射到数据库的列名，后跟多个参数用于规范数据 |  |

1. 说明：

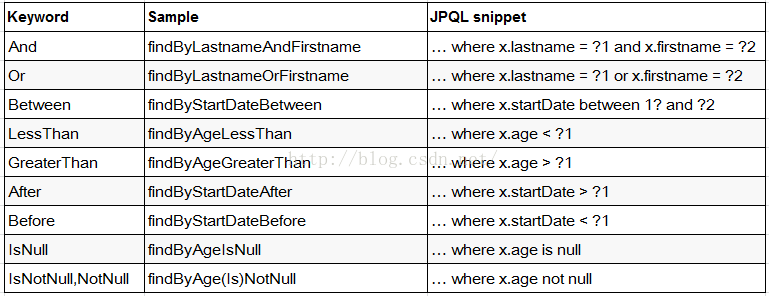
BaseEntity是小组提取的封装好的一个公共类，JpaRepository接口继承于PagingAndSortingRespository,实现一组JPA规范相关的方法 ，即已经封装好的方法，按照一定的规范就可以直接使用，避免重复写sql方法操作，同时将实例映射到数据库中，进行持久化。JpaSpecificationExecutor接口,它是一个单独的接口,现实中也经常使用到它.使用的它的原因是 在执行分页排序的查询时,可以附加查询条件,增加了灵活性.

1. Pojo：

POJO（Plain Old Java Objects）简单的Java对象，实际就是普通JavaBeans，是为了避免和EJB混淆所创造的简称。

使用POJO名称是为了避免和 [EJB](http://baike.baidu.com/view/3542.htm" \t "https://www.cnblogs.com/chengzixin/p/_blank)混淆起来, 而且简称比较直接. 其中有一些属性及其getter setter方法的类,没有业务逻辑，有时可以作为 [VO](http://baike.baidu.com/view/1097386.htm" \t "https://www.cnblogs.com/chengzixin/p/_blank)(value -object)或 [dto](http://baike.baidu.com/view/160599.htm" \t "https://www.cnblogs.com/chengzixin/p/_blank)(Data Transform Object)来使用.当然,如果你有一个简单的运算属性也是可以的,但不允许有业务方法,也不能携带有connection之类的方法。

1. Springdata方法使用规范



假如我们创建如下的查询：findByUserDepUuid(),框架在解析该方法时，首先剔除findBy，然后对剩下的属性进行解析，假设查询实体为Doc。

  --先判断userDepUuid (根据POJO（Plain Ordinary Java Object简单java对象，实际就是普通java bean）规范，首字母变为小写。)是否是查询实体的一个属性，如果根据该属性进行查询；如果没有该属性，继续第二步。

  --从右往左截取第一个大写字母开头的字符串(此处为Uuid)，然后检查剩下的字符串是否为查询实体的一个属性，如果是，则表示根据该属性进行查询；如果没有该属性，则重复第二步，继续从右往左截取；最后假设 user为查询实体的一个属性。

  --接着处理剩下部分（DepUuid），先判断user所对应的类型是否有depUuid属性，如果有，则表示该方法最终是根据 “ Doc.user.depUuid” 的取值进行查询；否则继续按照步骤 2的规则从右往左截取，最终表示根据“Doc.user.dep.uuid” 的值进行查询。

  --可能会存在一种特殊情况，比如 Doc包含一个user的属性，也有一个 userDep 属性，此时会存在混淆。可以明确在属性之间加上 "\_"以显式表达意图，比如"findByUser\_DepUuid()"或者"findByUserDep\_uuid()"。

  --特殊的参数： 还可以直接在方法的参数上加入分页或排序的参数，比如：Page<UserModel>findByName(String name, Pageable pageable);List<UserModel>findByName(String name, Sort sort);

1. **项目业务层封装**

本层主要接收控制层参数，进行业务方面的处理，之后访问dao数据层，但因为本项目初始较为简单，只做了单纯的增删改查的调用，所以本层不做说明

1. **展示层封装**

**8.1 配置跨域**

# 8.1.1什么是跨域

跨域是指不同域名之间相互访问。跨域，指的是浏览器不能执行其他网站的脚本，源[origin]就是协议、域名和端口号。它是由浏览器的同源策略造成的，是浏览器对JavaScript施加的安全限制。

详情参考下面这篇博客https://www.cnblogs.com/yuansc/p/9076604.html

**8.1.2 [Springmvc添加支持CORS跨域访问](https://www.cnblogs.com/shihaiming/p/8716830.html)**

本项目采用配置过滤器的方法来进行跨域访问，代码如下

**package** pioneer.seahorse;  
**import** org.springframework.web.servlet.HandlerInterceptor;  
**import** org.springframework.web.servlet.ModelAndView;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
*/\*\*  
 \* ProcessInterceptor 类  
 \*  
 \** ***@author*** *任鑫意  
 \** ***@date*** *2019/6/25 9:53  
 \*/***public class** ProcessInterceptor **implements** HandlerInterceptor {  
 @Override  
 **public boolean** preHandle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o) **throws** Exception {  
 httpServletResponse.setHeader(**"Access-Control-Allow-Origin"**, **"\*"**);  
 httpServletResponse.setHeader(**"Access-Control-Allow-Headers"**, **"Content-Type,Content-Length, Authorization, Accept,X-Requested-With"**); httpServletResponse.setHeader(**"Access-Control-Allow-Methods"**,**"PUT,POST,GET,DELETE,OPTIONS"**);  
 httpServletResponse.setHeader(**"X-Powered-By"**,**"Jetty"**);  
 String method= httpServletRequest.getMethod();  
 **if** (method.equals(**"OPTIONS"**)){  
 httpServletResponse.setStatus(200);  
 **return false**;  
 }  
 System.***out***.println(method);  
 **return true**;  
 }  
 @Override  
 **public void** postHandle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, ModelAndView modelAndView) **throws** Exception {  
 }  
 @Override  
 **public void** afterCompletion(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, Exception e) **throws** Exception { }}

**8.2编写springmvcController**

**8.2.1springmvcContrller三种编写方法**

1. **String**

@RequestMapping("test")

public String list(Map<String, Object> map){

////创建一个数组对象

List<Map<String, Object>> userList = new ArrayList<Map<String, Object>>();

for (int i = 0; i <10; i++) {

//存储对象map<K, V>

Map<String, Object> map1 = new HashMap<String, Object>();

map.put("name", "HelloKitty"+i);

map.put("age", i);

map.put("dept", "技术部"+i);

//将map对象存储在userList数组中 userList.add(map);

}

//[{age=Avenstar4, name=Avenstar4}]

map.put("elList", userList);

//返回视图

return "system/userList";

}

1. **Ajax**

//接收参数：@RequestParam(value="username", required=true, defaultValue="hello") String username

@RequestMapping(value="/test3", method=RequestMethod.POST)

@ResponseBody

public Map<String, Object> list3(HttpServletRequest request, @RequestParam(value="username", required=true, defaultValue="hello") String username){

Map<String, Object> map = new HashMap<>();

//创建一个数组对象

List<Map<String, Object>> userList = new ArrayList<Map<String, Object>>();

for (int i = 0; i < 5; i++) {

//存储对象map<K, V>

Map<String, Object> map1 = new HashMap<String, Object>();

map1.put("name", "HelloKitty"+i);

map1.put("age", i);

map1.put("dept", "技术部"+i);

userList.add(map1);

}

map.put("userList", userList);

return map;

}

}

1. **ModelandView**

@RequestMapping(value="test2", method=RequestMethod.GET)

public ModelAndView list2(Model model){

//创建一个视图对象

ModelAndView view = new ModelAndView();

//创建一个数组对象

List<Map<String, Object>> userList = new ArrayList<Map<String, Object>>();

for (int i = 0; i < 5; i++) {

//存储对象map<K, V>

Map<String, Object> map = new HashMap<String, Object>();

map.put("name", "HelloKitty"+i);

map.put("age", i);

map.put("dept", "技术部"+i);

userList.add(map);

}

//绑定数据

model.addAttribute("userList", userList);

//设置视图

view.setViewName("system/userList");

return view;

}

**第九章 项目后台开发步骤**

**9.1 准备工作**

**9.1.1 创建项目**

在总工程sea.horse下创建sea.horse的子项目

见4.3.4节

**9.1.2 pom文件**

sea.horse的pom文件如下

*<?***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**>pioneer</**groupId**>  
 <**artifactId**>sea.horse</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**packaging**>pom</**packaging**>  
 <**modules**>  
 <**module**>sea.horse-base</**module**>  
 <**module**>sea.horse-dao</**module**>  
 <**module**>sea.horse-service</**module**>  
 <**module**>sea.horse-web</**module**>  
 <**module**>sea.horse-client</**module**>  
 </**modules**>  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**java.version**>1.8</**java.version**>  
 <**spring.version**>5.1.0.RELEASE</**spring.version**>  
 <**spring-data-jpa.version**>2.1.0.RELEASE</**spring-data-jpa.version**>  
 <**hibernate.version**>5.2.16.Final</**hibernate.version**>  
 </**properties**>  
  
 <**dependencies**>  
 *<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->* <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-webmvc</**artifactId**>  
 <**version**>5.1.5.RELEASE</**version**>  
 </**dependency**>  
 *<!-- log4j -->* <**dependency**>  
 <**groupId**>org.apache.logging.log4j</**groupId**>  
 <**artifactId**>log4j-api</**artifactId**>  
 <**version**>2.11.0</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.logging.log4j</**groupId**>  
 <**artifactId**>log4j-core</**artifactId**>  
 <**version**>2.11.0</**version**>  
 </**dependency**>  
 *<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-slf4j-impl -->* <**dependency**>  
 <**groupId**>org.apache.logging.log4j</**groupId**>  
 <**artifactId**>log4j-slf4j-impl</**artifactId**>  
 <**version**>2.11.2</**version**>  
 </**dependency>**  
 *<!-- ============== test begin ============== -->* <**dependency**>*<!-- JUnit单元测试框架 -->* <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.12</**version**>  
 </**dependency**>  
 <**dependency**>*<!-- spring对测试框架的简单封装功能 -->* <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-test</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 </**dependencies**>  
 <**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**version**>3.7.0</**version**>  
 <**configuration**>  
 <**source**>${java.version}</**source**>  
 <**target**>${java.version}</**target**>  
 <**encoding**>UTF-8</**encoding**>  
 </**configuration**>  
 </**plugin**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-project-info-reports-plugin</**artifactId**>  
 <**version**>2.9</**version**>  
 </**plugin**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-surefire-plugin</**artifactId**>  
 <**version**>2.21.0</**version**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
</**project**>

base层的pom文件

*<?***xml version="1.0"***?>*<**project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**parent**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 </**parent**>  
 <**artifactId**>sea.horse-base</**artifactId**>  
 <**name**>sea.horse-base</**name**>  
 <**url**>http://maven.apache.org</**url**>  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 </**properties**>  
 <**dependencies**>  
 *<!-- https://mvnrepository.com/artifact/org.hibernate.javax.persistence/hibernate-jpa-2.1-api -->* <**dependency**>  
 <**groupId**>org.hibernate.javax.persistence</**groupId**>  
 <**artifactId**>hibernate-jpa-2.1-api</**artifactId**>  
 <**version**>1.0.2.Final</**version**>  
 </**dependency**>  
 *<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->* <**dependency**>  
 <**groupId**>org.apache.commons</**groupId**>  
 <**artifactId**>commons-lang3</**artifactId**>  
 <**version**>3.8.1</**version**>  
 </**dependency**>  
 *<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations -->* <**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-annotations</**artifactId**>  
 <**version**>2.9.8</**version**>  
 </**dependency**>  
 *<!-- https://mvnrepository.com/artifact/org.springframework.data/spring-data-jpa -->* <**dependency**>  
 <**groupId**>org.springframework.data</**groupId**>  
 <**artifactId**>spring-data-jpa</**artifactId**>  
 <**version**>2.1.6.RELEASE</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.12</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

dao层的pom文件

*<?***xml version="1.0"***?>*<**project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**parent**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 </**parent**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse-dao</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**name**>sea.horse-dao</**name**>  
 <**url**>http://maven.apache.org</**url**>  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 </**properties**>  
 <**dependencies**>  
 *<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager -->* <**dependency**>  
 <**groupId**>org.hibernate</**groupId**>  
 <**artifactId**>hibernate-entitymanager</**artifactId**>  
 <**version**>5.4.2.Final</**version**>  
 </**dependency**>  
 *<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-dbcp2 -->* <**dependency**>  
 <**groupId**>org.apache.commons</**groupId**>  
 <**artifactId**>commons-dbcp2</**artifactId**>  
 <**version**>2.6.0</**version**>  
 </**dependency**>  
 *<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->* <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>8.0.15</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-orm</**artifactId**>  
 <**version**>5.1.6.RELEASE</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse-base</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.12</**version**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 *<!-- read excel start -->* <**dependency**>  
 <**groupId**>org.apache.poi</**groupId**>  
 <**artifactId**>poi</**artifactId**>  
 <**version**>3.17</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.poi</**groupId**>  
 <**artifactId**>poi-ooxml</**artifactId**>  
 <**version**>3.17</**version**>  
 </**dependency**>  
 *<!-- read excel start -->* </**dependencies**>  
 <**build**>  
 <**plugins**>  
 *<!-- To use the plugin goals in your POM or parent POM -->* <**plugin**>  
 <**groupId**>io.github.divinespear</**groupId**>  
 <**artifactId**>jpa-schema-maven-plugin</**artifactId**>  
 <**version**>0.2.1</**version**>  
 <**configuration**>  
 <**format**>true</**format**>  
 <**scriptAction**>drop-and-create</**scriptAction**>  
 <**databaseAction**>drop-and-create</**databaseAction**>  
 <**vendor**>hibernate</**vendor**>  
 <**packageToScan**>pioneer.seahorse</**packageToScan**>  
 <**jdbcDriver**>com.mysql.cj.jdbc.Driver</**jdbcDriver**>  
 <**jdbcUrl**>jdbc:mysql://202.196.37.168:3306/seahorse</**jdbcUrl**>  
 <**jdbcUser**>student</**jdbcUser**>  
 <**jdbcPassword**>Student\_123456</**jdbcPassword**>  
 *<!--<hibernate.dialect>org.hibernate.dialect.MySQL5InnoDBDialect</hibernate.dialect>-->* <**properties**>  
 <**hibernate.dialect**>org.hibernate.dialect.MySQL5InnoDBDialect</**hibernate.dialect**>  
 </**properties**>  
 </**configuration**>  
 <**dependencies**>  
 *<!-- https://mvnrepository.com/artifact/org.hibernate.javax.persistence/hibernate-jpa-2.1-api -->* <**dependency**>  
 <**groupId**>org.hibernate.javax.persistence</**groupId**>  
 <**artifactId**>hibernate-jpa-2.1-api</**artifactId**>  
 <**version**>1.0.2.Final</**version**>  
 </**dependency**>  
 *<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager -->* <**dependency**>  
 <**groupId**>org.hibernate</**groupId**>  
 <**artifactId**>hibernate-entitymanager</**artifactId**>  
 <**version**>5.2.16.Final</**version**>  
 </**dependency**>  
 *<!-- https://mvnrepository.com/artifact/javax.xml.bind/jaxb-api -->* <**dependency**>  
 <**groupId**>javax.xml.bind</**groupId**>  
 <**artifactId**>jaxb-api</**artifactId**>  
 <**version**>2.3.0</**version**>  
 </**dependency**>  
 *<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->* <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>8.0.15</**version**>  
 </**dependency**>  
 </**dependencies**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
</**project**>

server层的pom文件

*<?***xml version="1.0"***?>*<**project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**parent**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 </**parent**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse-service</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**name**>sea.horse-service</**name**>  
 <**url**>http://maven.apache.org</**url**>  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 </**properties**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.12</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse-dao</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

web层的pom文件

*<?***xml version="1.0"***?>*<**project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**parent**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 </**parent**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse-web</**artifactId**>  
 <**packaging**>war</**packaging**>  
 <**name**>sea.horse-web Maven Webapp</**name**>  
 <**url**>http://maven.apache.org</**url**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.12</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse-dao</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse-service</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>pioneer</**groupId**>  
 <**artifactId**>sea.horse-client</**artifactId**>  
 <**type**>war</**type**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 </**dependency**>  
 *<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->* <**dependency**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>jstl</**artifactId**>  
 <**version**>1.2</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.itextpdf</**groupId**>  
 <**artifactId**>itext-asian</**artifactId**>  
 <**version**>5.1.0</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.lowagie</**groupId**>  
 <**artifactId**>itext</**artifactId**>  
 <**version**>2.1.7</**version**>  
 </**dependency**>  
 *<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->* <**dependency**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>javax.servlet-api</**artifactId**>  
 <**version**>4.0.1</**version**>  
 <**scope**>provided</**scope**>  
 </**dependency**>  
  
 *<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->* <**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-databind</**artifactId**>  
 <**version**>2.9.7</**version**>  
 </**dependency**>  
  
 *<!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->* <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 <**version**>1.2.58</**version**>  
 </**dependency**>  
  
 *<!-- https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper -->* <**dependency**>  
 <**groupId**>com.github.pagehelper</**groupId**>  
 <**artifactId**>pagehelper</**artifactId**>  
 <**version**>5.1.8</**version**>  
 </**dependency**>  
 </**dependencies**>  
 <**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.eclipse.jetty</**groupId**>  
 <**artifactId**>jetty-maven-plugin</**artifactId**>  
 <**version**>9.4.17.v20190418</**version**>  
 <**configuration**>  
 <**scanIntervalSeconds**>10</**scanIntervalSeconds**>  
 <**webApp**>  
 <**contextPath**>/seahorse</**contextPath**>  
 </**webApp**>  
 </**configuration**>  
 </**plugin**>  
  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**version**>3.3</**version**>  
 <**configuration**>  
 <**source**>1.8</**source**>  
 <**target**>1.8</**target**>  
 <**encoding**>utf8</**encoding**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
 <**finalName**>sea.horse-web</**finalName**>  
 </**build**>  
</**project**>

**9.1.3 配置资源文件**

1、sea.horse-dao\src\main\resources\application.dao.xml

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:jpa="http://www.springframework.org/schema/data/jpa"  
 xmlns="http://www.springframework.org/schema/beans"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  
  
 http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/jpa/spring-jpa.xsd"  
 default-lazy-init="true"**>  
  
  
 *<!-- Activates scanning of @Repository -->* <**jpa:repositories  
 base-package="pioneer.seahorse.\*\*.dao" entity-manager-factory-ref="emf"**/>  
</**beans**>

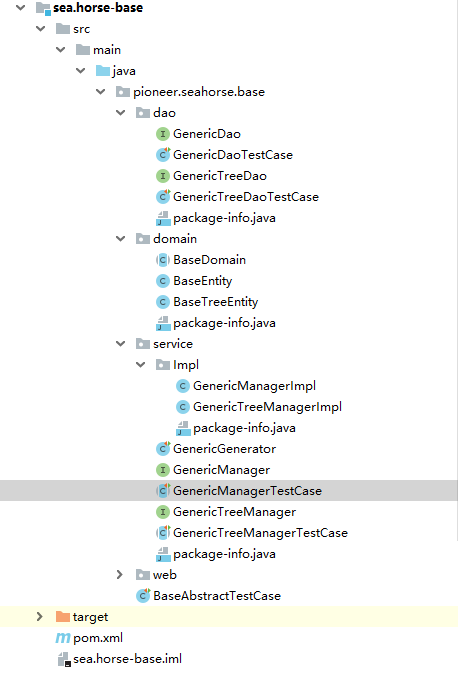
1. sea.horse-web\src\main\resources\config\application.server.xml

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans 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="http://www.springframework.org/schema/beans"  
 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"  
 default-lazy-init="true"**>  
  
 *<!-- Activates scanning of @Service -->* <**context:component-scan  
 base-package="pioneer.seahorse.\*.service.impl"** />  
  
</**beans**>

**9.2 src源文件夹**

**9.2.1 base层**

目录结构



项目注解说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 注解名 | 注解（属性）说明 | 备注 |
| 1 | @Configuration | 声明配置类 |  |
| 2 | @EnableJpaRepositories(basePackages = "edu.zut.cs.sea.horse.\*") | 开启JPA存储库扫描并声明扫描的包 |  |
| 3 | @Bean | 将实例化的对象注册成Bean |  |

1、controller/GenericController

**package** pioneer.seahorse.base.web;  
**import** org.apache.commons.lang3.StringUtils;  
**import** org.springframework.data.domain.Page;  
**import** org.springframework.data.domain.PageRequest;  
**import** org.springframework.data.domain.Pageable;  
**import** org.springframework.data.domain.Sort;  
**import** org.springframework.data.domain.Sort.Direction;  
**import** org.springframework.web.bind.annotation.\*;  
**import** java.io.IOException;  
**import** java.io.Serializable;  
**import** java.util.Date;  
**import** pioneer.seahorse.base.domain.BaseEntity;  
**import** pioneer.seahorse.base.service.GenericManager;  
  
  
**public abstract class** GenericController<T **extends** BaseEntity, PK **extends** Serializable, M **extends** GenericManager<T, PK>> {  
 **protected** PK **id**;  
 **protected** M **manager**;  
 **protected** T **model**;  
 **protected** Page<T> **page**;  
  
 **protected** Pageable **pageable**;  
 **protected int pageNumber** = 0;  
 **protected int pageSize** = 20;  
  
 */\*\*  
 \** ***@param model*** *\** ***@return*** *\*/* @RequestMapping(value = **"/"**, method = RequestMethod.***POST***, produces = **"application/json"**, consumes = **"application/json"**)  
 @ResponseBody  
 **public** T create(@RequestBody T model) {  
 **this**.**model** = model;  
 Date date = **new** Date();  
 **this**.**model**.setDateCreated(date);  
 **this**.**model**.setDateModified(date);  
 **this**.**model** = **this**.**manager**.save(**this**.**model**);  
 **return this**.**model**;  
 }  
  
 */\*\*  
 \** ***@param id*** *\** ***@throws*** *IOException  
 \*/* @RequestMapping(value = **"/{id}"**, method = RequestMethod.***DELETE***, produces = **"application/json"**)  
 @ResponseBody  
 **public void** delete(@PathVariable PK id) **throws** IOException {  
 **this**.**manager**.delete(id);  
 }  
  
 */\*\*  
 \* 根据输入，返回分页结果中的当前页，包括当前页信息和其中的实体对象集合  
 \*  
 \** ***@param request*** *\** ***@param response*** *\** ***@return*** *\*/* @RequestMapping(value = **"/"**, method = RequestMethod.***GET***, produces = **"application/json"**)  
 @ResponseBody  
 **public** Page<T> get(@RequestParam(name = **"page"**, defaultValue = **"0"**) String pageNumber,  
 @RequestParam(name = **"limit"**, defaultValue = **"20"**) String pageSize) {  
 **if** (StringUtils.*isNotBlank*(pageNumber)) {  
 **this**.**pageNumber** = Integer.*valueOf*(pageNumber) - 1;  
 }  
 **if** (StringUtils.*isNotBlank*(pageSize)) {  
 **this**.**pageSize** = Integer.*valueOf*(pageSize);  
 }  
 **this**.**pageable** = PageRequest.*of*(**this**.**pageNumber**, **this**.**pageSize**, **new** Sort(Direction.***ASC***, **"id"**));  
 **this**.**page** = **this**.**manager**.findAll(**this**.**pageable**);  
 *//logger.info(this.page);* **return this**.**page**;  
 }  
  
 */\*\*  
 \** ***@param id*** *\** ***@return*** *\*/* @RequestMapping(value = **"/{id}"**, method = RequestMethod.***GET***, produces = **"application/json"**)  
 @ResponseBody  
 **public** T getOne(@PathVariable PK id) {  
 **return this**.**manager**.findById(id);  
  
 }  
  
 */\*\*  
 \** ***@param id*** *\** ***@param model*** *\** ***@return*** *\*/* @RequestMapping(value = **"/{id}"**, method = RequestMethod.***PUT***, produces = **"application/json"**, consumes = **"application/json"**)  
 @ResponseBody  
 **public** T update(@PathVariable PK id, @RequestBody T model) {  
 model.setId(Long.*valueOf*(id.toString()));  
 model.setDateModified(**new** Date());*// 更新修改时间* **this**.**model** = **this**.**manager**.save(model);  
 **return this**.**model**;  
 }  
  
}

注解说明：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 注解名 | 属性 | 注解（属性）说明 | 备注 |
| 1 | @RequestMapping | Value | 指定请求的实际地址 |  |
| consumes | 指定处理请求的提交内容类型 |
| Method | 指定请求的method类型 |
| produces | 指定返回的内容类型 |
| 2 | @ResponseBody |  | 将返回的结果直接写入 HTTP 响应正文 |  |
| 3 | @RequestBody |  | 将 HTTP 请求正文插入方法中 |  |
| 4 | @PathVariable |  | 获得请求url中的动态参数的 |  |
| 5 | @RequestParam | Value | 参数名字 |  |
| required | 默认是true，表示请求中一定要有相应的参数 |
| defaultValue | 默认值，表示如果请求中没有同名参数时的默认值 |

2、dao/GenericDao

**package** pioneer.seahorse.base.dao;  
  
**import** org.springframework.data.jpa.repository.JpaRepository;  
**import** org.springframework.data.jpa.repository.JpaSpecificationExecutor;  
**import** org.springframework.data.repository.NoRepositoryBean;  
**import** pioneer.seahorse.base.domain.BaseEntity;  
  
**import** java.io.Serializable;  
  
*/\*\*  
 \* Generic DAO (Data Access Object) with common methods to CRUD POJOs.  
 \* Implemented with Spring-Data-JPA Repository  
 \* <p>  
 \* Extend this interface if you want type safe (no casting necessary) DAO's for  
 \* your domain objects.  
 \*  
 \** ***@param <T>*** *\* a type variable, Entity  
 \** ***@param <PK>*** *\* the primary key for that type，Entity Id  
 \** ***@author*** *<a href="mailto:ming616@gmail.com">Liu Xiaoming</a>  
 \*/*@NoRepositoryBean  
**public interface** GenericDao<T **extends** BaseEntity, PK **extends** Serializable>  
 **extends** JpaRepository<T, PK>, JpaSpecificationExecutor<T> {  
}

注解说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 注解名 | 注解（属性）说明 | 备注 |
| 1 | @ NoRepositoryBean | spring不会去实例化该repository |  |

3、domain/BaseDomain

**package** pioneer.seahorse.base.domain;  
  
**import** java.io.Serializable;  
  
*/\*\*  
 \* Base abstract class for Model objects. Child objects should implement  
 \* toString(), equals() and hashCode().  
 \*  
 \** ***@author*** *<a href="mailto:ming616@gmail.com">Liu Xiaoming</a>  
 \*/*@SuppressWarnings(**"serial"**)  
**public abstract class** BaseDomain **implements** Serializable {  
  
 */\*\*  
 \* Compares object equality. When using Hibernate, the primary key should not be  
 \* a part of this comparison.  
 \*  
 \** ***@param o*** *\* object to compare to  
 \** ***@return*** *true/false based on equality tests  
 \*/* **public abstract boolean** equals(Object o);  
  
 */\*\*  
 \* When you override equals, you should override hashCode. See "Why are equals()  
 \* and hashCode() importation" for more information:  
 \* http://www.hibernate.org/109.html  
 \*  
 \** ***@return*** *hashCode  
 \*/* **public abstract int** hashCode();  
  
 */\*\*  
 \* Returns a multi-line String with key=value pairs.  
 \*  
 \** ***@return*** *a String representation of this class.  
 \*/* **public abstract** String toString();  
}

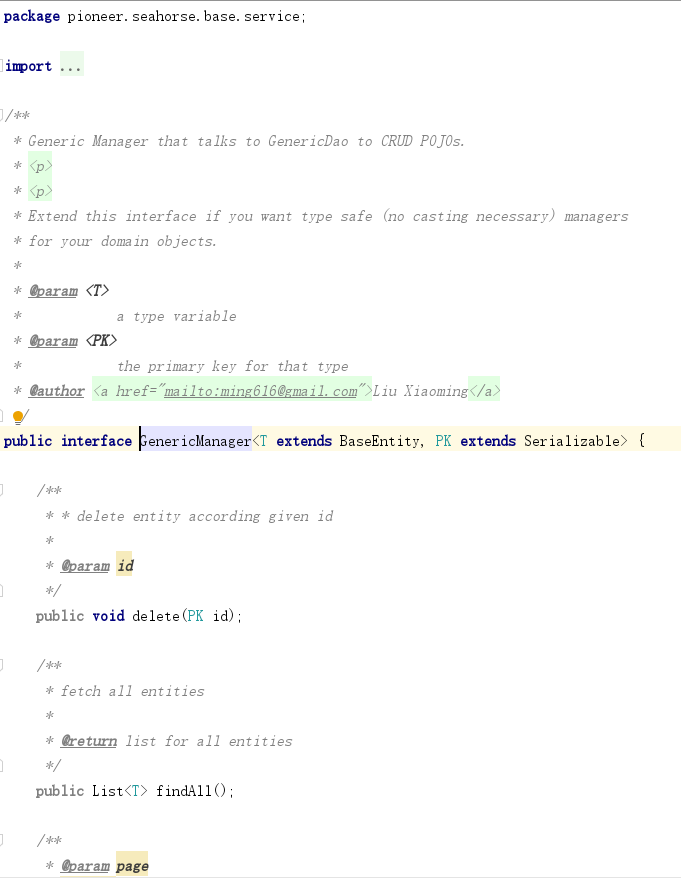
4、domain/BaseEntity

**package** pioneer.seahorse.base.domain;  
**import** com.fasterxml.jackson.annotation.JsonIgnoreProperties;  
**import** org.apache.commons.lang3.builder.HashCodeBuilder;  
**import** org.apache.commons.lang3.builder.ToStringBuilder;  
**import** org.apache.commons.lang3.builder.ToStringStyle;  
  
**import** javax.persistence.\*;  
**import** java.util.Date;  
@MappedSuperclass  
@JsonIgnoreProperties(value = { **"hibernateLazyInitializer"**, **"handler"** })  
@Access(value=AccessType.***FIELD***)  
**public class** BaseEntity **extends** BaseDomain {  
 */\*\*  
 \*  
 \*/* **private static final long *serialVersionUID*** = -6163675075289529459L;  
   
 */\*\*  
 \* 实体创建时间  
 \*/* @Temporal(TemporalType.***TIMESTAMP***)  
*// @Temporal(TemporalType.DATE)* @Column(name = **"DATECREATED"**)  
 **protected** Date **dateCreated** = **new** Date();  
  
 */\*\*  
 \* 实体修改时间  
 \*/* @Temporal(TemporalType.***TIMESTAMP***)  
*// @Temporal(TemporalType.TIME)* @Column(name = **"DATEMODIFIED"**)  
 **protected** Date **dateModified** = **new** Date();  
  
 */\*\*  
 \* 实体是否被删除  
 \*/* @Column(name = **"DELETED"**)  
 **protected** Boolean **deleted**;  
 @Column(name = **"ENTITY\_NAME"**)  
 **protected** String **entityName**;  
 @Id  
 @Column(name = **"ID"**)  
 @GeneratedValue(strategy = GenerationType.***AUTO***)  
 **protected** Long **id**;  
 @Version  
 @Access(AccessType.***FIELD***)  
 **private long version**;  
 @Override  
 **public boolean** equals(Object obj) {  
 **if** (**null** != obj) {  
 **if** (obj **instanceof** BaseEntity) {  
 BaseEntity domain = (BaseEntity) obj;  
 **if** (**this**.**id** == domain.**id**) {  
 **return true**;  
 }  
 }  
 }  
 **return false**;  
 }  
 **public** Date getDateCreated() {  
 **return dateCreated**;  
 }  
 **public** Date getDateModified() {  
 **return dateModified**;  
 }  
 **public** Boolean getDeleted() {  
 **return deleted**;  
 }  
 **public** String getEntityName() {  
 **return entityName**;  
 }  
 **public** Long getId() {  
 **return id**;  
 }  
 @Override  
 **public int** hashCode() {  
 **if** (**this**.**id** == **null**) {  
 **this**.**id** = Long.*valueOf*(0);  
 }  
 **return** HashCodeBuilder.*reflectionHashCode*(**this**.**id**);  
 }  
 **public void** setDateCreated(Date dateCreated) {  
 **this**.**dateCreated** = dateCreated;  
 }  
 **public void** setDateModified(Date dateModified) {  
 **this**.**dateModified** = dateModified;  
 }  
 **public void** setDeleted(Boolean deleted) {  
 **this**.**deleted** = deleted;  
 }  
 **public void** setEntityName(String entityName) {  
 **this**.**entityName** = entityName;  
 }  
 **public void** setId(Long id) {  
 **this**.**id** = id;  
 }  
 @Override  
 **public** String toString() {  
 **return** ToStringBuilder.*reflectionToString*(**this**, ToStringStyle.***MULTI\_LINE\_STYLE***);  
 }  
}

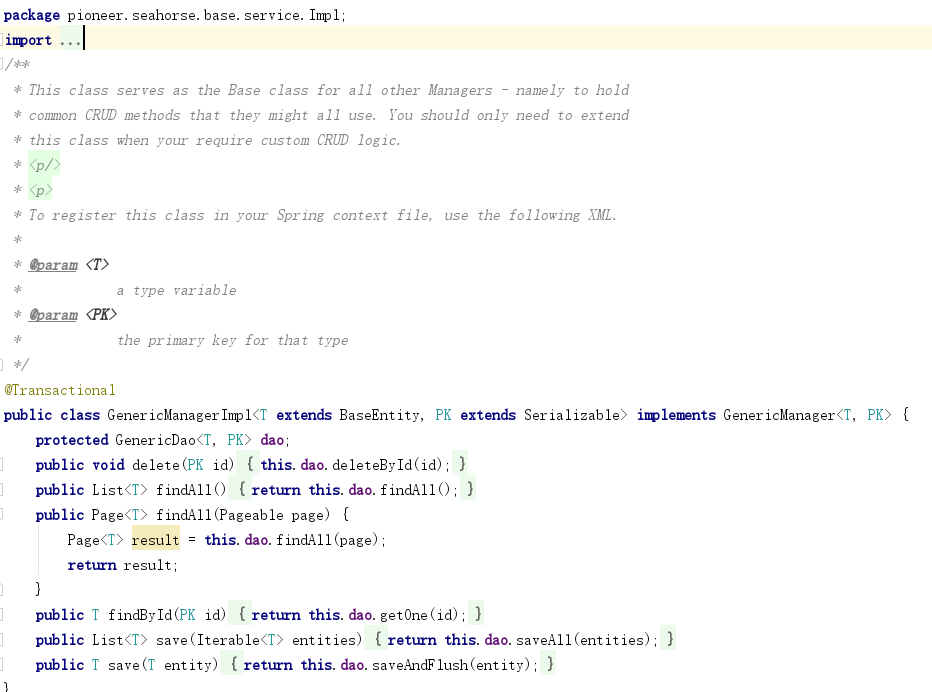
注解说明：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 注解名 | 属性 | 注解（属性）说明 | 备注 |
| 1 | @ MappedSuperclass |  | 基类实体，继承它的子类实体在隐射时会自动扫描该基类实体的隐射属性 |  |
| 2 | @ JsonIgnoreProperties | Value | 忽略类中不存在的字段 |  |
| 3 | @ Temporal |  | 格式化Date、TIMESTAMP、TIME类型数据 |  |
| 4 | @ Column | Name | 持久化属性，映射表中的字段 |  |
| 5 | @Id |  | 声明一个实体类的属性映射为数据库的主键列 |  |
| 6 | @ GeneratedValue | Strategy | 用于标注主键的生成策略 |  |

5、service/GenericManager



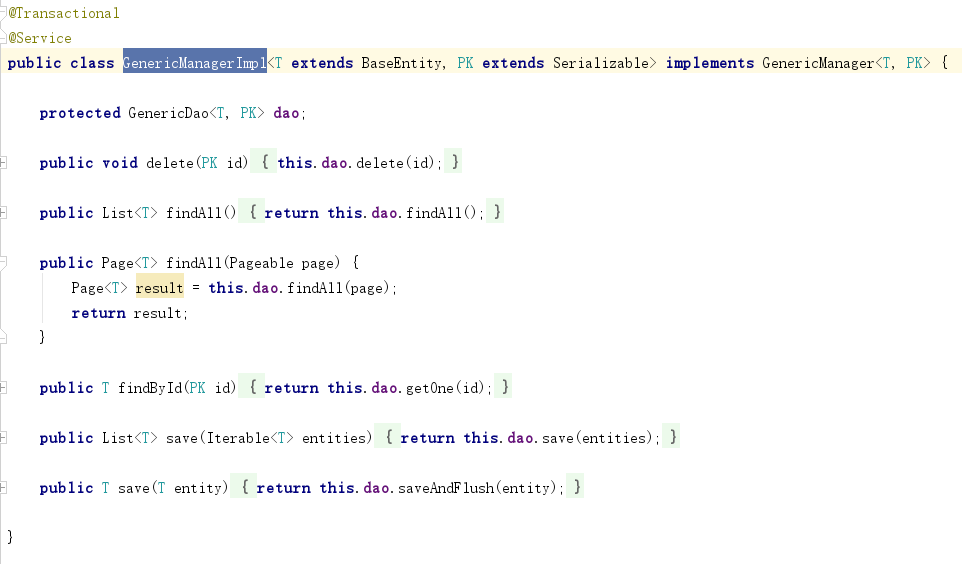
6、service/impl/GenericGenerator



注解说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 注解名 | 注解（属性）说明 | 备注 |
| 1 | @ RunWith | 运行器 |  |
| 2 | @ SpringmvcTest | 单元测试 |  |
| 3 | @ ContextConfiguration | 引入多个配置文件 |  |

1. service/impl/GenericManagerImpl



注解说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 注解名 | 注解（属性）说明 | 备注 |
| 1 | @ Transactional | 管理事务 |  |
| 2 | @ Service | 标注业务层组件 |  |

8、service/impl/TestApplication

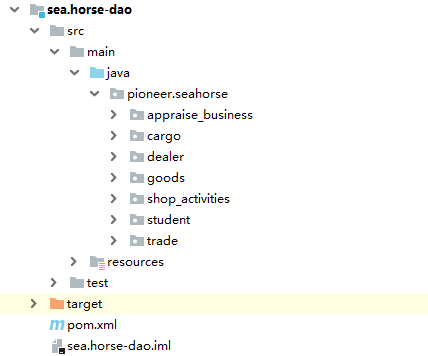
**package** pioneer.seahorse.base.service;  
  
**import** org.junit.Before;  
**import** org.junit.Test;  
**import** org.springframework.test.context.ContextConfiguration;  
**import** pioneer.seahorse.base.BaseAbstractTestCase;  
**import** pioneer.seahorse.base.domain.BaseEntity;  
  
**import** java.io.Serializable;  
**import** java.lang.reflect.InvocationTargetException;  
**import** java.util.List;  
  
@ContextConfiguration(locations = { **"classpath:/applicationContext-resources.xml"**,  
 **"classpath:/applicationContext-dao.xml"**, **"classpath:/applicationContext-service.xml"** })  
**public abstract class** GenericManagerTestCase<PK **extends** Serializable, T **extends** BaseEntity, M **extends** GenericManager<T, PK>>  
 **extends** BaseAbstractTestCase {  
 **protected** T **entity**;  
 **protected** List<T> **list**;  
 **protected** M **manager**;  
  
 **private** Class<T> **persistentClass**;  
  
 **public** GenericManagerTestCase(**final** Class<T> persistentClass) {  
 **this**.**persistentClass** = persistentClass;  
 }  
 **private void** assertEquals(T entity2, T foundEntity) {  
 *//* ***TODO Auto-generated method stub*** }  
 @Before  
 **public void** setUp() **throws** Exception {  
 **this**.**entity** = **this**.**persistentClass**.getDeclaredConstructor().newInstance();  
 }  
 @Test  
 **public void** testCRUD() {  
 **if** (**this**.**entity** == **null**) {  
 **try** {  
 **this**.**entity** = **this**.**persistentClass**.getDeclaredConstructor().newInstance();  
 T newEntity = **this**.**manager**.save(**this**.**entity**);  
 @SuppressWarnings(**"unchecked"**)  
 PK entity\_id = (PK) newEntity.getId();  
 T foundEntity = **this**.**manager**.findById(entity\_id);  
 assertEquals(newEntity, foundEntity);  
  
 } **catch** (InstantiationException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 } **catch** (IllegalAccessException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 } **catch** (IllegalArgumentException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 } **catch** (InvocationTargetException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 } **catch** (NoSuchMethodException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 } **catch** (SecurityException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 } } }}

注解说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 注解名 | 注解（属性）说明 | 备注 |
| 1 | @ SpringmvcApplication | 快捷配置启动类 | 相当于@Target(ElementType.TYPE)  @Retention(RetentionPolicy.RUNTIME)  @Documented  @Inherited  @SpringmvcConfiguration  @EnableAutoConfiguration  @ComponentScan(excludeFilters |
| 2 | @ RestController | @ResponseBody和@Controller的组合注解 |  |

**9.2.2 dao层**

目录结构如下



1. /dao/CaogoDao

**import** org.springframework.stereotype.Repository;  
**import** pioneer.seahorse.base.dao.GenericDao;  
**import** pioneer.seahorse.cargo.domain.Cargo;  
  
@Repository(**"cargoDao"**)  
**public interface** CargoDao **extends** GenericDao<Cargo, Long> {  
}

2./domain/Cargo

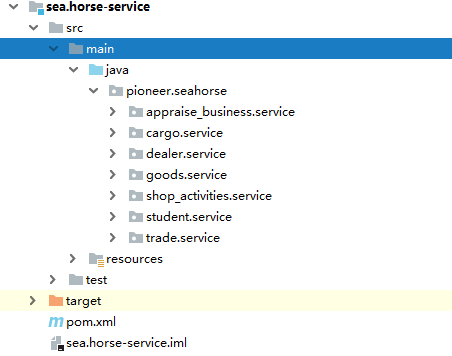
**package** pioneer.seahorse.cargo.domain;  
  
**import** com.fasterxml.jackson.annotation.JsonIdentityInfo;  
**import** com.fasterxml.jackson.annotation.ObjectIdGenerators;  
**import** org.hibernate.annotations.NotFound;  
**import** org.hibernate.annotations.NotFoundAction;  
**import** pioneer.seahorse.base.domain.BaseEntity;  
  
**import** javax.persistence.\*;  
  
*/\*\*  
 \* Cargo 类  
 \*  
 \** ***@author*** *任鑫意  
 \** ***@date*** *2019/6/20 17:07  
 \*/*@Entity  
@Table(name = **"SH\_Cargo"**)  
  
**public class** Cargo **extends** BaseEntity {  
 **private static final long *serialVersionUID*** = -6833228937585333122L;  
 @Column(name=**"name"**)  
 String **name**;  
 @Column(name=**"num"**)  
 String **num**;  
 @Column(name=**"price"**)  
 Integer **price**;  
 @Column(name = **"number"**)  
 Integer **number**;  
 @ManyToOne(optional = **false**)  
 @JoinColumn(name = **"shelves\_ID"**)  
 Shelves **shelves**;  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 **public** String getNum() {  
 **return num**;  
 }  
  
 **public void** setNum(String num) {  
 **this**.**num** = num;  
 }  
  
 **public** Integer getPrice() {  
 **return price**;  
 }  
  
 **public void** setPrice(Integer price) {  
 **this**.**price** = price;  
 }  
  
 **public** Integer getNumber() {  
 **return number**;  
 }  
  
 **public void** setNumber(Integer number) {  
 **this**.**number** = number;  
 }  
  
 **public** Shelves getShelves() {  
 **return shelves**;  
 }  
  
 **public void** setShelves(Shelves shelves) {  
 **this**.**shelves** = shelves;  
 }  
  
}

注解说明：

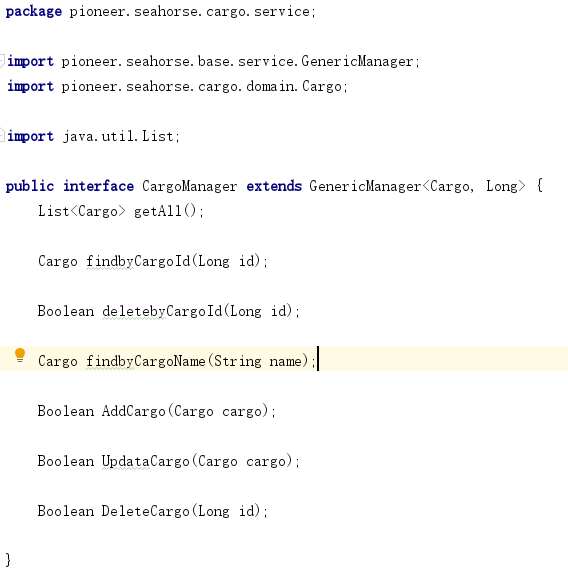
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 注解名 | 注解 | 注解（属性）说明 | 备注 |
| 1 | @ Table | Name | 映射的表名 |  |
| 2 | @ Entity |  | 声明实体类 | 默认orm规则 |

**9.2.3 service层**

目录结构如下



1. CargoManager.java

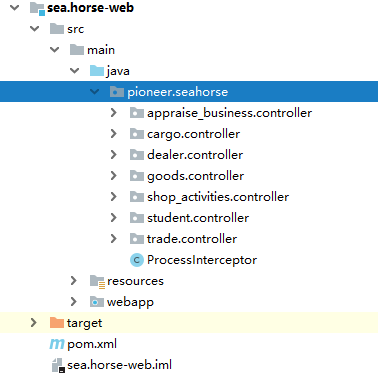


1. impl\CargoManagerImpl.java

**package** pioneer.seahorse.cargo.service.impl;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Service;  
**import** pioneer.seahorse.base.service.Impl.GenericManagerImpl;  
**import** pioneer.seahorse.cargo.dao.CargoDao;  
**import** pioneer.seahorse.cargo.domain.Cargo;  
**import** pioneer.seahorse.cargo.domain.Shelves;  
**import** pioneer.seahorse.cargo.service.CargoManager;  
**import** pioneer.seahorse.cargo.service.ShelvesManager;  
  
**import** java.util.Date;  
**import** java.util.Iterator;  
**import** java.util.List;  
**import** java.util.Optional;  
  
*/\*\*  
 \* CargoManager 类  
 \*  
 \** ***@author*** *任鑫意  
 \** ***@date*** *2019/6/21 19:15  
 \*/*@Service(**"cargoManager"**)  
  
**public class** CargoManagerImpl **extends** GenericManagerImpl<Cargo, Long> **implements** CargoManager {  
  
 CargoDao **cargoDao**;  
 ShelvesManager **shelvesManager**;  
  
 @Autowired  
 **public void** setCargoDao(CargoDao cargoDao) {  
 **this**.**cargoDao** = cargoDao;  
 **this**.**dao** = **this**.**cargoDao**;  
 }  
 @Autowired  
 **public void** setShelvesManager(ShelvesManager shelvesManager) {  
 **this**.**shelvesManager**=shelvesManager;  
 }  
  
  
 @Override  
 **public** List<Cargo> getAll() {  
 **this**.**shelvesManager**.updata();  
 List<Cargo> result = **this**.**dao**.findAll();  
 **return** result;  
 }  
  
 @Override  
 **public** Cargo findbyCargoId(Long id) {  
 Cargo cargo = **null**;  
 List<Cargo> all=**this**.**dao**.findAll();  
 **for**(**int** i=0;i<all.size();i++)  
 {  
 **if** (all.get(i).getId().equals(id))  
 cargo=all.get(i);  
 }  
 **return** cargo;  
 }  
  
 @Override  
 **public** Cargo findbyCargoName(String name) {  
 Cargo cargo = **null**;  
 List<Cargo> all=**this**.**dao**.findAll();  
 **for**(**int** i=0;i<all.size();i++)  
 {  
 **if** (all.get(i).getName().equals(name))  
 cargo=all.get(i);  
 }  
 **return** cargo;  
 }  
  
 @Override  
 **public** Boolean deletebyCargoId(Long id) {  
 Optional<Cargo> stuList=**this**.**dao**.findById(id);  
 **if**(stuList.equals(**null**)) {  
 **return false**;  
 } **else** {  
 **this**.**dao**.delete(stuList.get());  
 **return true**;  
 }  
 }  
  
 @Override  
 **public** Boolean AddCargo(Cargo cargo){  
 String name=cargo.getShelves().getName();  
 Shelves shelves=**this**.**shelvesManager**.findbyShelvesName(name);  
 cargo.setShelves(shelves);  
 **this**.save(cargo);  
 shelves.getCargos().add(cargo);  
 **this**.**shelvesManager**.reduce(shelves);  
 **return true**;  
 }  
  
 @Override  
 **public** Boolean UpdataCargo(Cargo cargo){  
 Shelves shelves=**this**.**shelvesManager**.findbyShelvesName(cargo.getShelves().getName());  
 Cargo cg=**this**.findbyCargoId(cargo.getId());  
 cg.setName(cargo.getName());  
 cg.setNum(cargo.getNum());  
 cg.setPrice(cargo.getPrice());  
 cg.setNumber(cargo.getNumber());  
 cg.setShelves(shelves);  
 cg.setDateModified(**new** Date());  
 **this**.save(cg);  
 shelves.getCargos().add(cg);  
 **return true**;  
 }  
  
 @Override  
 **public** Boolean DeleteCargo(Long id){  
 Cargo cargo=**this**.findbyCargoId(id);  
 Shelves shelves=**this**.**shelvesManager**.findbyShelvesName(cargo.getShelves().getName());  
 Boolean bool=**this**.deletebyCargoId(id);  
 Iterator<Cargo> it = shelves.getCargos().iterator();  
 **while** (it.hasNext()) {  
 Cargo str = it.next();  
 **if** (cargo.getId().equals(str.getId())) {  
 it.remove();  
 }  
 }  
 **this**.**shelvesManager**.reduce(shelves);  
 **return** bool;  
 }  
}

**9.2.4 web层**

目录结构如下



1. web\controller\CargoController.java

**package** pioneer.seahorse.cargo.controller;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Controller;  
**import** org.springframework.web.bind.annotation.RequestBody;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.ResponseBody;  
**import** pioneer.seahorse.base.web.GenericController;  
**import** pioneer.seahorse.cargo.domain.Cargo;  
**import** pioneer.seahorse.cargo.domain.Shelves;  
**import** pioneer.seahorse.cargo.service.CargoManager;  
**import** pioneer.seahorse.cargo.service.ShelvesManager;  
  
**import** java.util.Date;  
**import** java.util.Iterator;  
**import** java.util.List;  
  
*/\*\*  
 \* CargoController 类  
 \*  
 \** ***@author*** *任鑫意  
 \** ***@date*** *2019/6/21 19:17  
 \*/*@Controller  
@RequestMapping(**"/cargo"**)  
**public class** CargoController **extends** GenericController<Cargo,Long, CargoManager> {  
  
 @Autowired  
 CargoManager **cargoManager**;  
  
 @Autowired  
 **public void** setCargoManager(CargoManager cargoManager) {  
 **this**.**cargoManager**=cargoManager;  
 **this**.**manager**=**this**.**cargoManager**;  
 }  
  
 @RequestMapping(**"/car"**)  
 **public** String index() {  
 **return "cargo"**;  
 }  
  
 @ResponseBody*//标识转换成JSON处理* @RequestMapping(value = **"all"**,produces = **"application/json;charset=utf-8"**)  
 **public** List<Cargo> findAllCargo(){  
 **return this**.**manager**.getAll();  
 }  
  
 @ResponseBody*//标识转换成JSON处理* @RequestMapping(value = **"deleteCargo"**,produces = **"application/json;charset=utf-8"**)  
 **public** Boolean deleteCargo(@RequestBody Long id){  
 **return this**.**cargoManager**.DeleteCargo(id);  
 }  
  
 @ResponseBody*//标识转换成JSON处理* @RequestMapping(value = **"addCargo"**,produces = **"application/json;charset=utf-8"**)  
 **public** Boolean addCargo(@RequestBody Cargo cargo){  
 **return this**.**cargoManager**.AddCargo(cargo);  
 }  
  
 @ResponseBody*//标识转换成JSON处理* @RequestMapping(value = **"updateCargo"**,produces = **"application/json;charset=utf-8"**)  
 **public** Boolean updateCargo(@RequestBody Cargo cargo){  
 **return this**.**cargoManager**.UpdataCargo(cargo);  
 }  
  
 @ResponseBody*//标识转换成JSON处理* @RequestMapping(value = **"searchCargo"**,produces = **"application/json;charset=utf-8"**)  
 **public** Cargo search(@RequestBody String name) {  
 **return this**.**manager**.findbyCargoName(name);  
 }  
}

注解说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 注解名 | 注解（属性）说明 | 备注 |
| 1 | @ Controller | 标记一个SpringMVC Controller 对象 |  |
| 2 | @ Autowired | 按照类型装配注入 |  |

**9.3 运行项目**

运行入口程序即可，打开浏览器

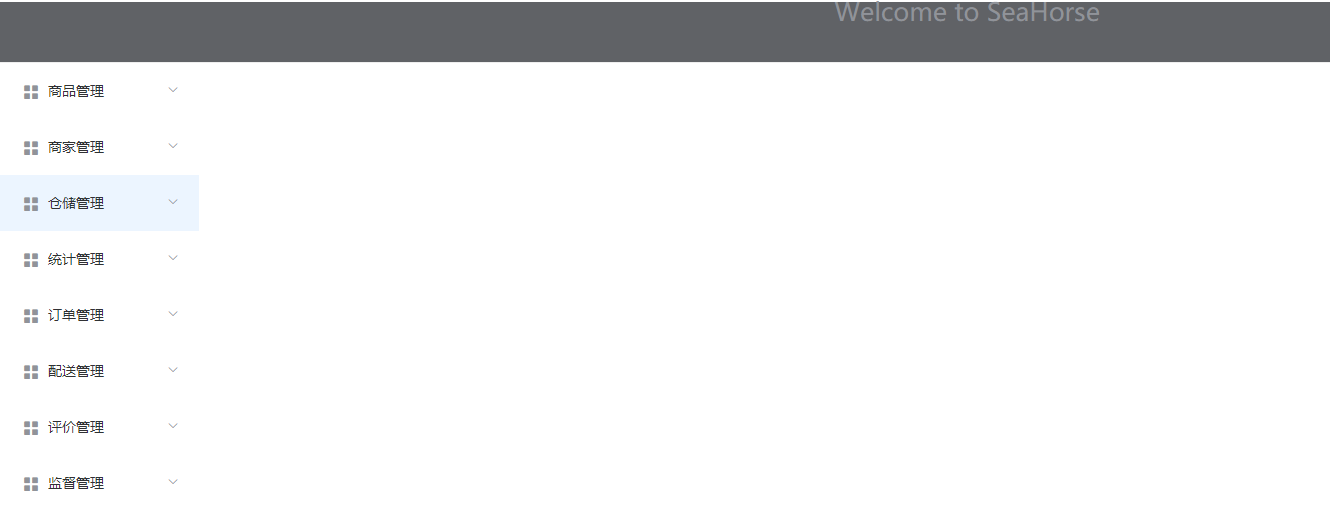


**第十章 项目前端开发步骤**

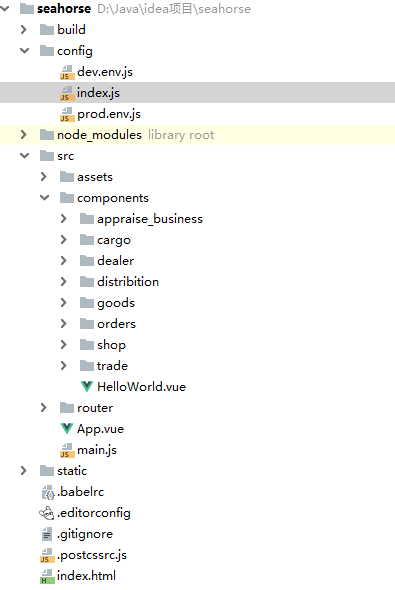
**10.1准备工作**

**10.1.1项目布局**

布局



因此建立如下组件



组件说明：

|  |  |  |
| --- | --- | --- |
| 序号 | 组件作用 | 备注 |
| 1 | 侧边栏 |  |
| 2 | 头部界面 |  |
| 3 | 主界面 |  |
| 4 | 内容管理 |  |
| 5 | 个性化页面设置 |  |

**10.1.2组件容器：**

运用Container的相关知识建立侧边栏，每个组件用div包裹，然后用下列标签讲组件进行包裹，可以直接在页面显示，如Layout页面中需要包含公共侧边栏和公共顶部以及用户管理的组件，具体参考element-ui的官方文档http://element.eleme.io/#/zh-CN/component/container



标签说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 标签名 | 标签作用 | 备注 |
| 1 | <el-container> | 外层容器。 |  |
| 2 | <el-header> | 顶栏容器 |  |
| 3 | <el-aside> | 侧边栏容器 |  |
| 4 | <el-main> | 主要区域容器 |  |
| 5 | <el-footer> | 底栏容器 |  |

**10.1.2组件路由**

如果一个组件被重复点击，例如配置一个changePassword的组件，详细知识请访问官方文档https://router.vuejs.org/

下面会演示一个例子，请按照如下操作配置路由

首先在router下建立一个js文件，默认会出现一个index.js，在这里面配置也是可以的

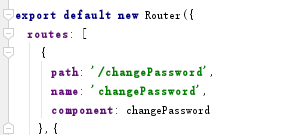
**第一步导入：**



**第二步：**

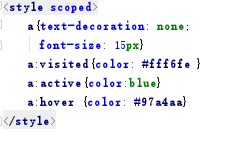
IMG_256

**第三步输入路径：**path后跟的是在导航栏输入的地址，component是组件的名字



**第四步使用：**这里是在侧边栏Aside下使用的，element-ui默认有超链接的样式，可用用css更改，

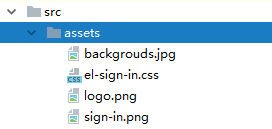
IMG_256



**10.1.3背景设置：**

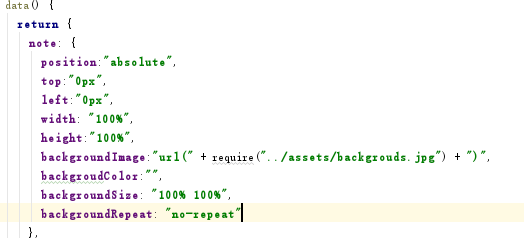
Vue.jss可以在js中设置背景颜色，背景图片等等，本项目中的Login页面使用的这个方法

**第一步：**在某路径下导入一张图片，并在标签后设置class或者id



IMG_256

**第二步：**在js中设置如下，require后跟项目中存储的图片



**10.1.4引入iconfont图标：**

iconfont网站是阿里巴巴矢量图标库网站，在这里你可以使用很多图标来美化你的网站，方法有多种，这里演示比较常用的，能够动态更新图标的方法。下面进行简单演示

**第一步：**在网站上选取你需要的图标并添加至项目，如果没有需要创建项目



**第二步：**添加至项目后会出现一个链接，如下图，复制这个连接，在index.js中添加<**link rel="stylesheet" href="链接"**>，在图表信息中，上面有三个选项，从左往右第一个选项是控制类名的、第二个是生成在线代码链接地址的、第三个就是下载至本地的图标代码文件包；



**第三步：**在assets文件夹中新建一个css文件，class后跟FontClass文件名，并添加如下代码，可以在iconfont网站上的项目中的更多操作图标中修改FontClass。

**第四步：**准备工作做好后，将需要该图标的地方用<i class="iconfont xxxx"></i>的形式引入即可，其中的类名复制图标下的“复制代码”即可得来！

**10.1.5前端获取数据并显示到页面：**

这里是利用axios在前后端进行数据传输，这里前面已经介绍过如何安装了，下面进行简单说明。Vue.js封装好了get和post方法，后台需要封装好了一个url进行数据传送，前端访问这个url并获取和传输相应的数据得到需要的数据并赋值给在前端的数据变量中，进而渲染在页面上。这里有三个变量：

1.url：访问得到传输数据的url、

2.params：你所需要传输的数据、

其中可以fetch中设置请求的公共参数

在各个视图和组件的方法中，直接调用接口并对返回的数据进行处理。

**10.1.6Vue.js的生命周期钩子函数和常用指令：**

简单的说分为以下8个阶段：

1.beforeCreate 创建前执行（data和el都还未初始化）  
2.create 完成创建 （完成了data数据初始化，el还未初始化）  
3.beforeMount 载入前（完成了data和el数据初始化）  
4.mounted 载入后html已经渲染(ajax请求可以放在这个函数中)  
5.beforeUpdate 更新前状态（view层的数据变化前，不是data中的数据改变前）  
6.update 更新状态后  
7.beforeDestroy 销毁前  
8.destroy 销毁后 （Dom元素存在，只是不再受vue控制）

常用指令

    1.v-model  接收用户输入的一些数据，直接就可以将这些数据 挂载到data属性里面  
    2.v-if  判断加载固定的内容，若为真加载，为假时删除元素,常用在用在权限管理，页面模块条件加载  
    3.v-show 元素会始终渲染并保持在dom中，v-show用法和v-if相同，安全性没有v-if高，v-show将元素display设置成none，并不是将元素直接移除。  
    4.v-else 元素必须紧跟在v-if或v-show元素的后面——否则它不能被识别  
    5.v-bind 给页面中html属性进行绑定  
    6.v-on 对页面中的事件进行绑定  
      事件修饰符 .stop (阻止单击事件冒泡)   .prevent (提交事件不再重载页面) .capture (使用事件捕获模式) .self(只当事件在该元素本身（而不是子元素）触发时触发回调)

7.v-for 控制html元素的循环，实现数据列表的生成  
    8.v-once 只渲染元素和组件**一次**。随后的重新渲染，元素/组件及其所有的子节点将被视为静态内容并跳过。这可以用于优化更新性能（源自博客https://blog.csdn.net/xiaoxiaoshuai\_\_/article/details/79625951 例子在上面）

**10.1.7组件之间的通信：**

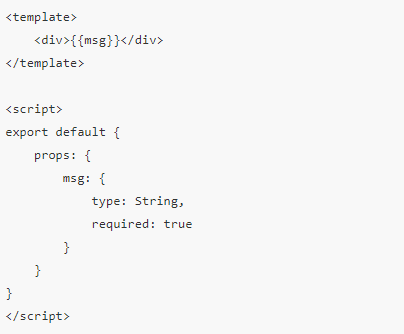
1. 父组件传到子组件之间的通信：

使用[props](https://cn.vuejs.org/v2/guide/components.html" \l "Prop)，父组件可以使用props向子组件传递数据。

父组件：



子组件：



2.子组件传到父组件的通信：利用$emit

父组件：





非父子组件，兄弟之间的通信：创建一个event事件，利用$emit和$on进行接收和传输.

部分源自博客：<https://www.cnblogs.com/answershuto/p/7711750.html>

**10.1.8 善用官网：**

Vue.js和element-ui的官网的学习资料非常详细，有问题先找官网。这里推荐个比较完备的视频，链接可能会失效，仅作参考

Vue.js的官网：<https://cn.vuejs.org/v2/guide/conditional.html>

element-ui的官网：[http://element-cn.eleme.io/#/zh-CN](http://element-cn.eleme.io/" \l "/zh-CN)

Vue.js的学习视频：<https://www.bilibili.com/video/av24826984/?p=73>

Element-UI的学习：基本使用方法和html类似，官方网站给的有代码。