# 1.Mybatis自动生成

详解：<https://www.cnblogs.com/smileberry/p/4145872.html>

# 2.Maven

https://ke.qq.com/webcourse/index.html#course\_id=303156&term\_id=100359247&taid=2222959108595764&vid=w142634hes2

## 1.maven的作用

[1].管理jar包

I:增加第三方jar包

II: jar包之间的依赖关系(自动关联下载所有依赖的jar，并且不会冲突)

[2].将项目拆分成若干个模块 dao service controlle

## 2.maven概念

是一个基于java平台的，自动化构建工具。

Make🡺ant 🡺maven 🡺gradle

1. 清理：删除编译的结果，为重新编译做准备
2. 编译：java🡺class
3. 测试：针对项目中的重点进行测试，亦可用测试代码 测试 开发代码
4. 报告：将测试结果进行显示
5. 打包：将项目中包含的多个文件压缩成一个文件，用于安装或部署。

-java项目🡺jar包

-web项目🡺war包

1. 安装：将打成的包 放到本地仓库 ，供其他项目使用
2. 部署：将打成的包 放到服务器

通常下载一个jar，先在本地仓库下载，如果本地仓库没有，在联网到远程仓库（镜像）下载。

Eclipse中的项目在部署时，会生成一个对应的部署项目。区别在于，部署项目没有源码文件src(java),只有编译后的class文件和jsp文件，因为二者的目录结构不同，因此Tomcat无法直接运行Eclipse复制过来的项目(因为要在Tomcat运行一个项目，该项目就必须严格遵循Tomcat 的目录结构 )

Eclipse中的项目要在Tomcat中运行，需要部署：

-Eclipse Add and Remove 部署项目

-将项目打包成一个.war包，直接复制到tomcat/webapps

## **3.Maven应用**

#### [1].下载配置maven



Binary 二进制 zip🡺Windows

Binary 二进制 gz 🡺Linux

#### [2].-配置java\_home

-配置maven\_home E:\PekingNN\apache-maven-3.5.2-bin\apache-maven-3.5.2

M2\_home

-配置Path % maven\_home%\bin

#### [3].验证 mvn –v

#### [4].配置本地仓库

- E:\PekingNN\apache-maven-3.5.2-bin\apache-maven-3.5.2\conf\settings.xml

-默认本地仓库：C:/Users/Administrator /.m2/repository

-修改本地仓库：<localRepository> E:/work /.m2/repository </localRepository>

## 4.Maven使用 （约定优于配置）

#### [1].Maven约定的目录结构

\* 项目

-src

--main 程序功能代码

---java java代码

---resource 资源代码，配置代码

--test 测试代码

---java java代码

---resource 资源代码，配置代码

-pom.xml 项目对象模型

#### [2.]坐标

<groupId>com.ifp.ifp</groupId> <groupId>域名反转+项目名</groupId>

<artifactId>ifp-web-parent</artifactId> <artifactId>模块名</artifactId>

<version>0.0.1</version> <version>版本号</version>

#### [3].依赖

A中的某些类，需要使用B中的某些类 称为A依赖B

在maven项目中，如果需要使用一个当时不存在的jar或者模块，可以通过依赖实现（去本地仓库、远程仓库去寻找）

A jar🡺 B jar

<dependency>

<groupId>joda-time</groupId>

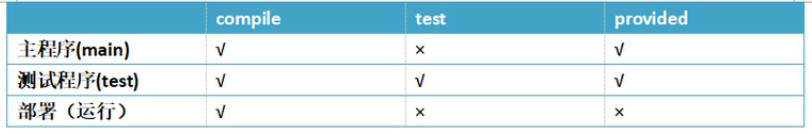
<artifactId>joda-time</artifactId>

<version>2.1</version>

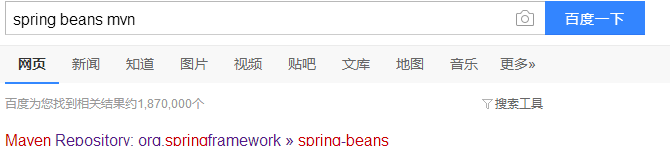
<scope>compile</scope>

</dependency>

**-依赖的范围、有效性：**



百度中搜索jar包



-**依赖排除**

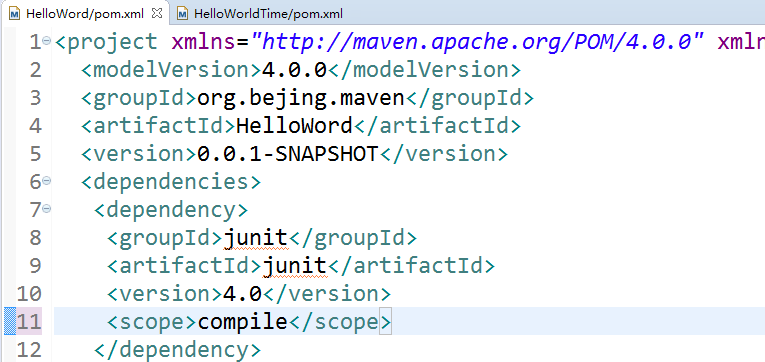


**-依赖的传递性**

A.jar🡺B.jar🡺C.jar

要使A.jar 🡺C.jar 当且仅当B.jar依赖于C.jar的范围是compile

HelloWorldTime 🡺 HelloWorld🡺Junit.jar



**-多个Maven项目如何依赖 A项目 依赖于 B项目**

--A项目 install 安装到本地仓库

--B项目依赖：

<!--本项目依赖 HelloWord项目-->

<dependency>

<groupId>org.bejing.maven</groupId>

<artifactId>HelloWord</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>

**-依赖原则**

a.路径最短优先原则

b.路径长度相同

i：（覆盖）在同一个pom文件中，有两个相同的依赖，后面声明的依赖会覆盖前面的依赖。

ii：（优先）在 不同一个pom文件中，有两个相同的依赖，先声明的依赖会覆盖后声明的依赖。

常见命令：执行mvn命令 ，必须在pom.xml所在目录

编译 （maven基础组件 基础jar）

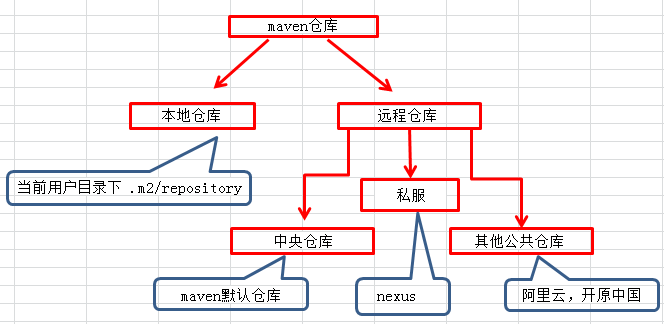
mvn compile (只编译main目录下的java文件)

mvn test 测试

mvn package 打包 jar/war

mvn install 安装 将开发的模块放到本地仓库，供其他模块使用 （放入的位置 通过gav决定）

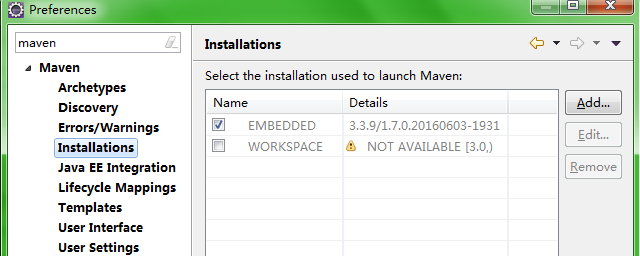
mvn clean 删除target文件 （删除编译文件的目录）



## 5.在Eclipse中创建Maven项目

#### [1].配置Maven

-配置Maven版本 

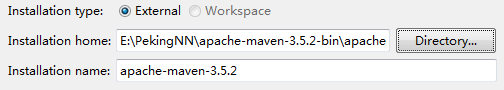


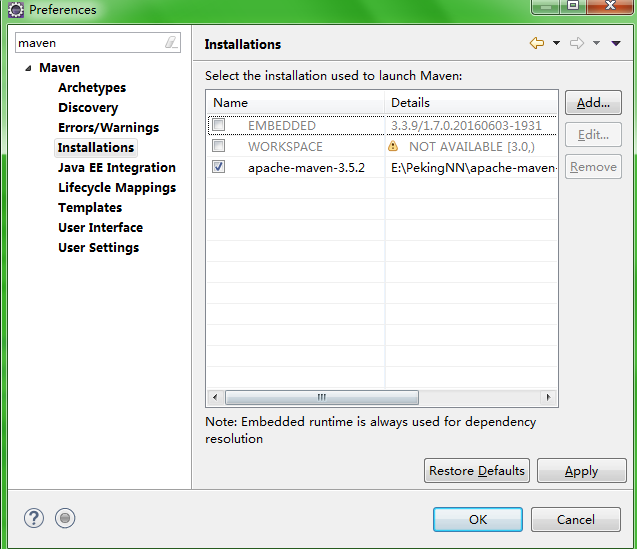
Eclipse默认版本

3

2

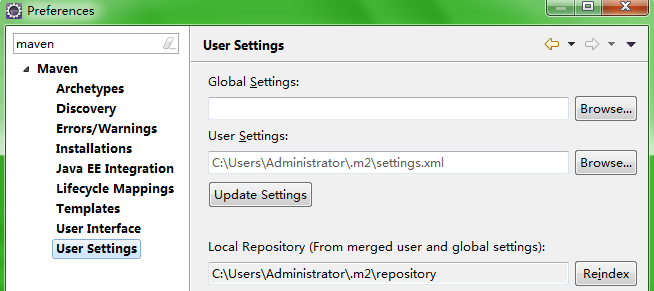
1



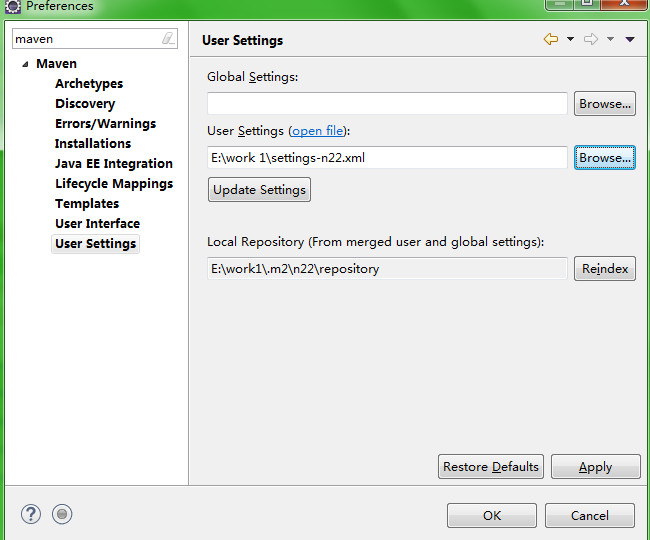


-配置本地仓库 （设置Setting.xml 自动识别本地仓库位置）

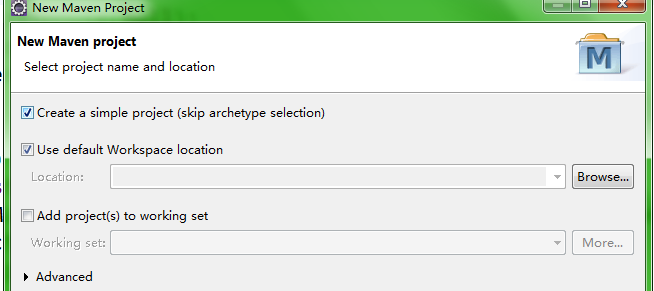


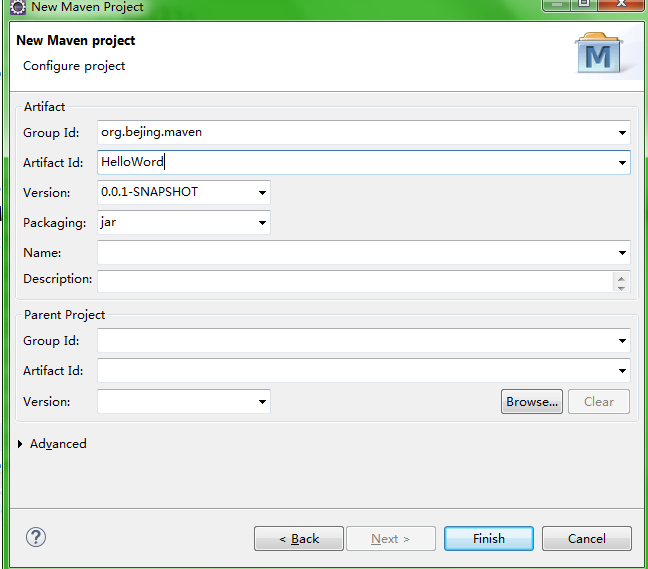


默认



#### [2].创建项目



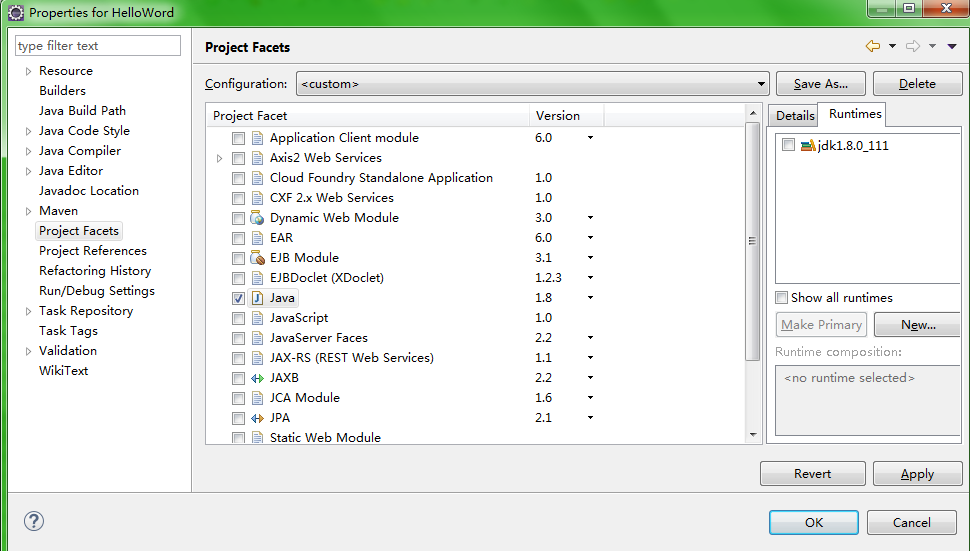


-在Eclipse中编写完pom.xml文件后 maven 🡺 update project

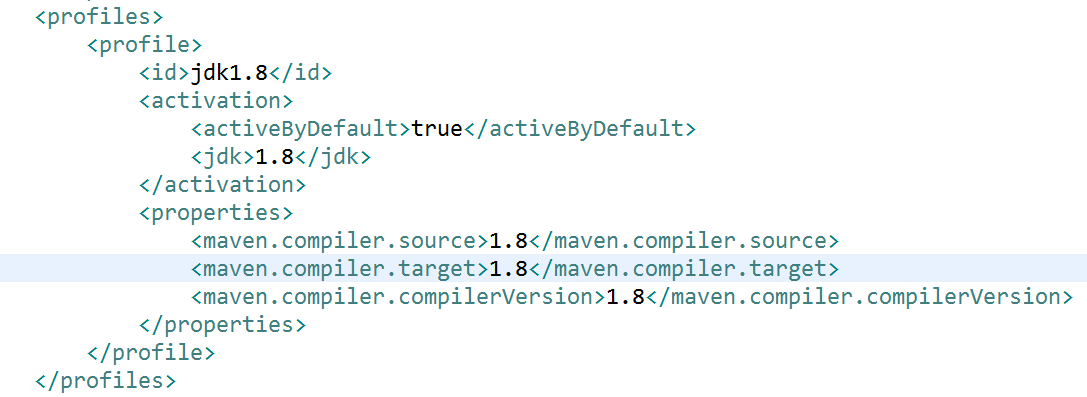
#### [3].统一项目中的jdk

-i：build path 删除旧版本 增加新版本

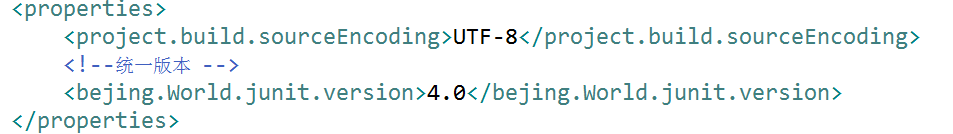
-ii:右键项目-属性-project facets java version 改版本

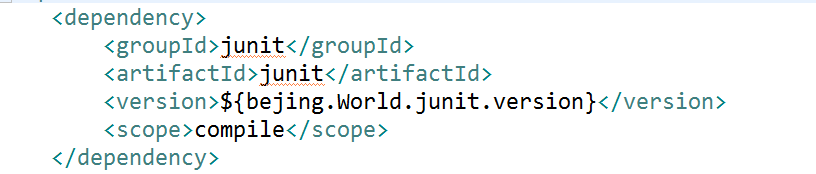


-iii:通过Maven统一



#### [4].在pom中统一版本

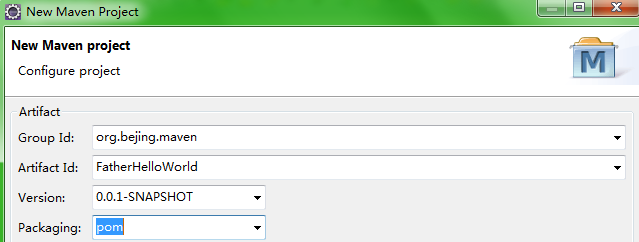




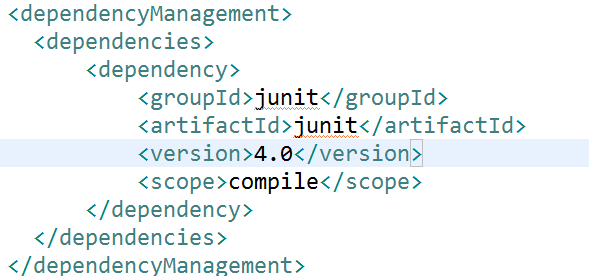
## 6.Maven继承实现

#### [1].继承实现步骤

I:建立父工程，父工程的打包方式是 pom



II:在父工程的pom文件中编写依赖:



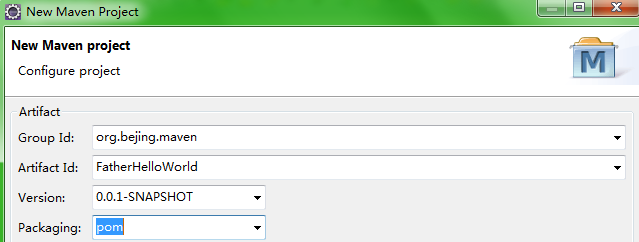
III:编写子类工程



IV:在子类中需要声明：使用哪些父类的依赖

## 7.聚合的使用

在一个总工程中配置聚合，（聚合的配置 ，只能配置在（打包方式为pom）的maven工程中）



<!-- 配置聚合 避免前置工程的install 操作 -->

<modules>

<!--项目的根路径 -->

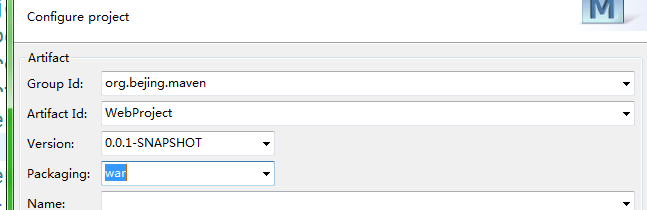
<module>../HelloWord</module>

<module>../HelloWorldTime</module>

</modules>

配置完聚合之后，以后只要操作总工程，则会自动操作聚合中配置过的工程

## 8.创建web项目



## 9.Maven生命周期

生命周期和构建关系：

生命周期中的顺序：a b c d e

当我们执行c命令，实际执行的是a b c

Maven生命周期是抽象的，本身不做任何的操作。每个阶段的具体工作都是由绑定的插件的插件目标完成。

Maven的三套生命周期是相互独立的。

**-clean 清理项目**

pre-clean

clean 清理上一次构建生成的文件，默认是删除target 文件夹

post-clean

**-default项目构建**

compile 编译项目主代码。对src/main/java 下的代码编译后，复制到项目输出的主classpath目录

test 使用单元测试框架执行测试单元，测试代码不会被打包或者部署

package 打包 包放在target目录下

deploy 将包部署到远程仓库中

**-site 建立项目站点**

pre-site

site 生成项目站点文档，默认目录是在target/site下

post-site

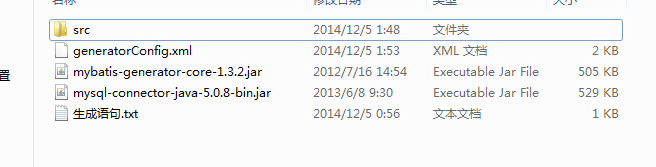
site-deploy 将生成的项目站点发布到服务器上

## 1.相关文件

下载Mybatis-Generator: <https://github.com/mybatis/generator/releases>

链接数据库jar包：

配置文件：pom.xml



## 2.详细配置

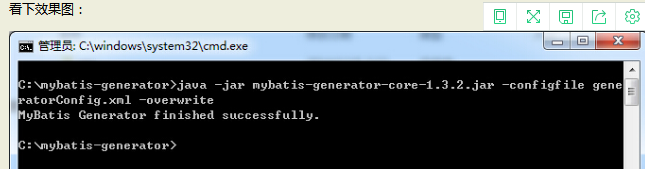
C:\Users\ADMINI~1\AppData\Local\Temp\1533265308(1).png

C:\Users\ADMINI~1\AppData\Local\Temp\1533265709(1).png

tableName和domainObjectName为必选项，分别代表数据库表名和生成的实力类名。

生成语句文件：

java -jar mybatis-generator-core-1.3.2.jar -configfile generatorConfig.xml –overwrite



# 3.Maven (mybatis自动生成实体代码)

#### [1].在pom文件根目录中增加如下代码

<build>

<plugins>

<plugin>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-maven-plugin</artifactId>

<version>1.3.2</version>

<configuration>

<verbose>true</verbose>

<overwrite>true</overwrite>

</configuration>

</plugin>

</plugins>

</build>

如果mapper文件中 import org.apache.ibatis.annotations.Param;报错。加入如下依赖

<dependency>

<groupId>org.apache.ibatis</groupId>

<artifactId>ibatis-core</artifactId>

<version>3.0</version>

</dependency>

#### [2].加入generatorConfig.xml 内容如下

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

<!DOCTYPE generatorConfiguration

PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"

"http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">

<generatorConfiguration> <!--数据库驱动路径 -->

<classPathEntry

location=*"E:\work1\.m2\n22\repository\mysql\mysql-connector-java\5.1.38\mysql-connector-java-5.1.38.jar"* />

<context id=*"DB2Tables"* targetRuntime=*"MyBatis3"*>

<commentGenerator>

<property name=*"suppressAllComments"* value=*"true"* />

</commentGenerator>

<jdbcConnection driverClass=*"com.mysql.jdbc.Driver"*

connectionURL=*"jdbc:mysql://localhost:3306/orderfood"* userId=*"root"* password=*"123456"*>

</jdbcConnection>

<javaTypeResolver>

<property name=*"forceBigDecimals"* value=*"false"* />

</javaTypeResolver>

<!--域模型层,生成的目标包,项目目标源文件 -->

<javaModelGenerator targetPackage=*"com.liting.model"*

targetProject=*"src/main/java"*>

<property name=*"enableSubPackages"* value=*"true"* />

<property name=*"trimStrings"* value=*"true"* />

</javaModelGenerator>

<!--XML映射文件,生成的位置（目标包）,源代码文件夹 -->

<sqlMapGenerator targetPackage=*"com.liting.mapperxml"*

targetProject=*"src/main/resources"*>

<property name=*"enableSubPackages"* value=*"true"* />

</sqlMapGenerator>

<!--XML对应的Mapper类 -->

<javaClientGenerator type=*"XMLMAPPER"*

targetPackage=*"com.liting.mapper"* targetProject=*"src/main/java"*>

<property name=*"enableSubPackages"* value=*"true"* />

</javaClientGenerator>

<!--下面是数据库表名和项目中需要生成类的名称，建议和数据库保持一致，如果有多个表，添加多个节点即可 -->

<table tableName=*"dishes"* domainObjectName=*"DishesMst"*

enableCountByExample=*"true"* enableUpdateByExample=*"true"*

enableDeleteByExample=*"true"* enableSelectByExample=*"true"*

selectByExampleQueryId=*"true"*></table>

</context>

</generatorConfiguration>

#### [3].生成

项目 右键--》run as --》 maven bulid --》弹出对话框 --》在goals中输入mybatis-generator:generate

# 4.注解 （本质 接口）

#### **[1]**[**.@Autowired**](mailto:.@Autowired) **@Qualifier @Resource**

详解（<https://blog.csdn.net/bestone0213/article/details/46561063>）

@Autowired 可以对成员变量、方法以及构造函数进行注释，而 @Qualifier 的标注对象是成员变量、方法入参、构造函数入参。

@Qualifier ：有多个Bean时，指定加载哪个bean

只能和 @Autowired 结合使用

1. @Autowired
2. public void setOffice(@Qualifier("office")Office office) {
3. this.office = office;
4. }

@Qualifier("office") 中的 office 是 Bean 的名称，所以 @Autowired 和 @Qualifier 结合使用时，自动注入的策略就从 byType 转变成 byName 了。

@Autowired ：自动注入

它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作

当不能确定 Spring 容器中一定拥有某个类的 Bean 时，可以在需要自动注入该类 Bean 的地方可以使用 @Autowired(required = false)

2. package com.baobaotao;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.beans.factory.annotation.Required;
7. public class Boss {
9. private Car car;
10. private Office office;
12. @Autowired
13. public void setCar(Car car) {
14. this.car = car;
15. }
16. @Autowired(required = false)
17. public void setOffice(Office office) {
18. this.office = office;
19. }
20. …
21. }

@Resource：有两个属性，分别是 name 和 type，Spring 将 @Resource 注释的 name 属性解析为 Bean 的名字，而 type 属性则解析为 Bean 的类型。如果既不指定 name 也不指定 type 属性，这时将通过反射机制使用 byName 自动注入策略。

1. public class Boss {
2. // 自动注入类型为 Car 的 Bean
3. @Resource
4. private Car car;
5. // 自动注入 bean 名称为 office 的 Bean
6. @Resource(name = "office")
7. private Office office;

@PostConstruct 和 @PreDestroy：初始化之后/销毁之前, 只能应用于方法上.

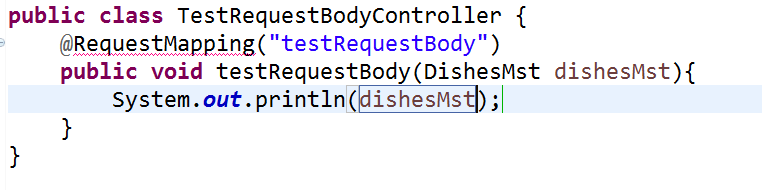
1. public class Boss {
2. @Resource
3. private Car car;
5. @Resource(name = "office")
6. private Office office;
8. @PostConstruct
9. public void postConstruct1(){
10. System.out.println("postConstruct1");
11. }
13. @PreDestroy
14. public void preDestroy1(){
15. System.out.println("preDestroy1");
16. }

#### [2][.@requestBody](mailto:.@requestBody)/@Responsebody

在业务方法如何获取ajax发送的请求参数，@requestBody（ajax的json字符串转换成目标类型）

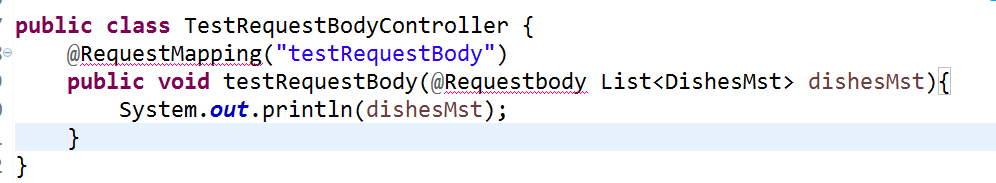
I:如果是一个简单的json,可以通过以往的方法，直接在方法入参加入指定类型的数据



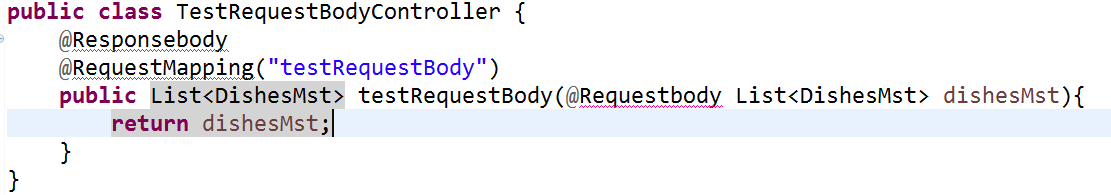


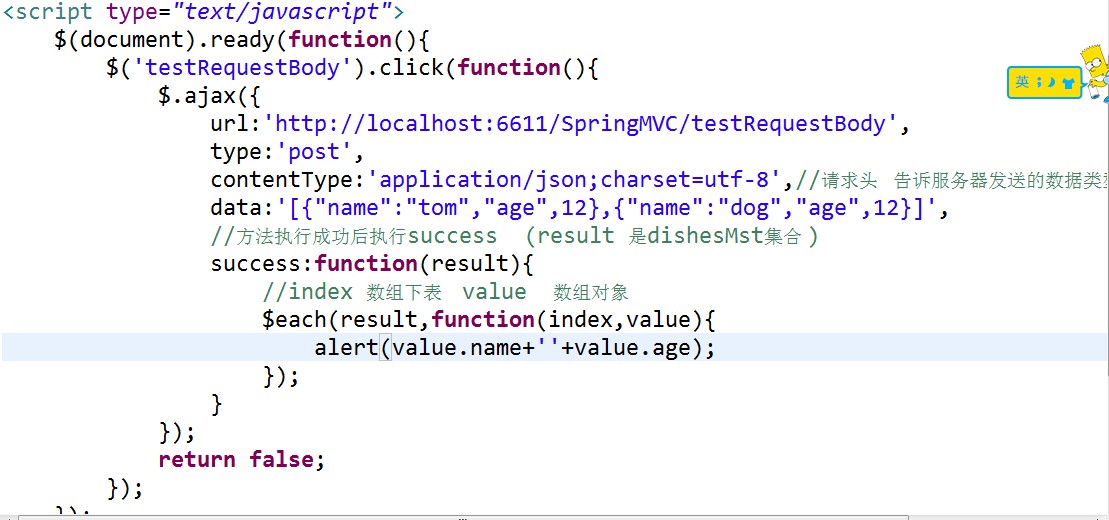
II:如果是一个数组，Ajax发送一个数组，业务方法入参是list，修饰方法的入参。





Ajax当中如何获取服务器响应的结果@Responsebody（将方法的返回值转换成相应的json类型）





@ModelAttribute注解：绑定数据到model中

# 5.关键字transient

Java的serialization提供了一种持久化对象实例的机制。当持久化对象时，可能有一个特殊的对象数据成员，我们不想用serialization机制来保存它。为了在一个特定对象的一个域上关闭serialization，可以在这个域前加上关键字transient。

打个比方，如果一个用户有一些敏感信息（如密码，银行卡号等），为了安全起见，不希望在网络操作（主要涉及到序列化操作，本地序列化缓存也适用）中被传输，这些信息对应的变量就可以加上transient关键字。换句话说，这个字段的生命周期仅存于调用者的内存中而不会写到磁盘里持久化。

　所以，transient的用途在于：阻止实例中那些用此关键字声明的变量持久化；当对象被反序列化时（从源文件读取字节序列进行重构），这样的实例变量值不会被持久化和恢复。

　1，一旦变量被transient修饰，变量将不再是对象持久化的一部分，该变量内容在序列化后无法获得访问。

　　2，transient关键字只能修饰变量，而不能修饰方法和类。注意，本地变量是不能被transient关键字修饰的。变量如果是用户自定义类变量，则该类需要实现Serializable接口。

　　3，被transient关键字修饰的变量不再能被序列化，一个静态变量不管是否被transient修饰，均不能被序列化。

# 6.深入分析Java的序列化与反序列化

a. <http://www.importnew.com/18024.html>

b. https://blog.csdn.net/wangloveall/article/details/7992448/

1、在Java中，只要一个类实现了java.io.Serializable接口，那么它就可以被序列化。

2、通过ObjectOutputStream和ObjectInputStream对对象进行序列化及反序列化

3、虚拟机是否允许反序列化，不仅取决于类路径和功能代码是否一致，一个非常重要的一点是两个类的序列化 ID 是否一致（就是 private static final long serialVersionUID）

4、序列化并不保存静态变量。

5、要想将父类对象也序列化，就需要让父类也实现Serializable 接口。

6、Transient 关键字的作用是控制变量的序列化，在变量声明前加上该关键字，可以阻止该变量被序列化到文件中，在被反序列化后，transient 变量的值被设为初始值，如 int 型的是 0，对象型的是 null。

7、服务器端给客户端发送序列化对象数据，对象中有一些数据是敏感的，比如密码字符串等，希望对该密码字段在序列化时，进行加密，而客户端如果拥有解密的密钥，只有在客户端进行反序列化时，才可以对密码进行读取，这样可以一定程度保证序列化对象的数据安全。

# 7.titles框架

<http://www.php.cn/js-tutorial-375382.html>

http://bootstrap-table.wenzhixin.net.cn/zh-cn/documentation/

<http://bootstrapvalidator.votintsev.ru/api/>

disabled : true,//设置是否可用

checkbox : true, // True以显示复选框。复选框列有固定的宽度

align : 'center',

valign : 'middle'

}, {

title : '方案ID', //列标题名称

field : 'planid', //列字段名称

align : 'center', //如何对齐列数据

valign : 'middle', //如何对齐单元格数据

visible : false //false 隐藏列项

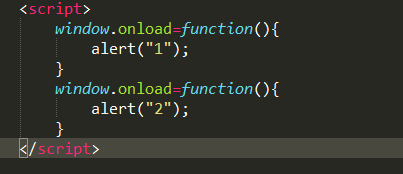
# 11.jQuery

#### 1.原生js和jQuery入口函数的加载模式不同

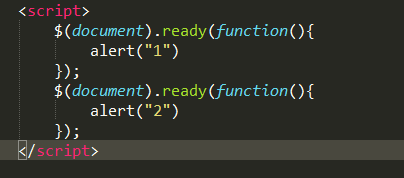
[1].原生js会等到Dom元素加载完毕，并且图片也加载完毕才会执行

[2].Jquery会等到Dom元素加载完毕，但不会等到图片加载完毕

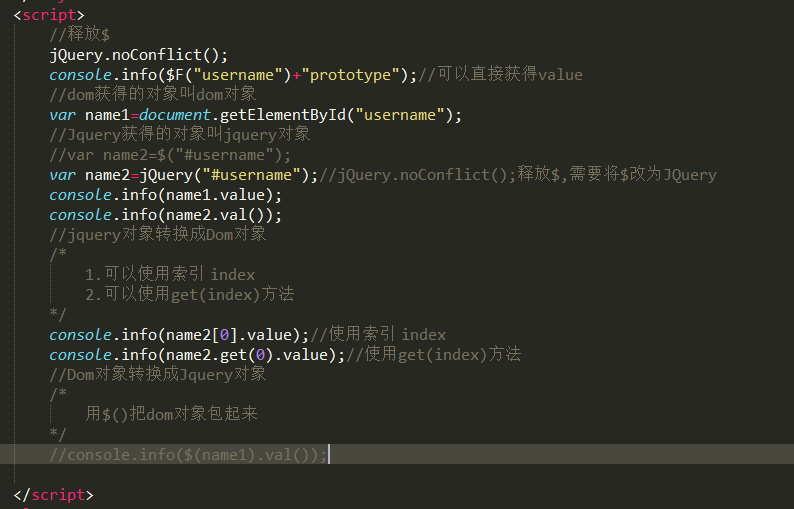
[3].原生的js如果编写了多个入口函数，后面的会覆盖前面的



[4].jQuery中编写多个入口函数，后面的不会被覆盖



[5]



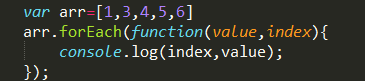
[6].jQuery对象是一个伪数组，（伪数组：有0-length-1的属性，并且有length属性）

#### 2.forEach()与$.each函数

1.第一个参数，遍历到的元素

2.第二个参数，遍历到的索引

注意：原生的forEach方法只能遍历数组，不能遍历伪数组



1. 第一个参数，遍历到的索引
2. 第二个参数，遍历到的元素

注意：$.each可以遍历伪数组

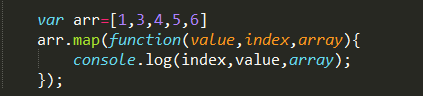
C:\Users\ADMINI~1\AppData\Local\Temp\1540277436(1).png

#### 2.map()与$.map函数

1.第一个参数，遍历到的元素

2.第二个参数，遍历到的索引

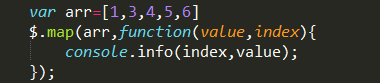
3.第三个参数，当前遍历的数组

****

1. 第一个参数要遍历的数组，
2. 第二个参数每遍历一个元素执行的回调函数

回调函数的第一个参数：遍历到的元素

第二个参数：遍历到的索引



总结：Each静态方法的默认返回值，遍历谁就返回谁。

Map静态方法的默认返回值是一个空数组

#### 3.$.trim() 去除字符串两端空格

C:\Users\ADMINI~1\AppData\Local\Temp\1540279156(1).png

#### 4.attr(name|pro|key,val|fn) 与removeAttr(name)

[1].Attr作用：获取或者设置属性节点的值

如果传递一个参数：代表获取属性节点的值

无论找到多少个元素，都只会返回第一个元素指定的属性节点的值。

如果传递两个参数：代表设置属性节点的值

找到多少个元素就会设置多少个元素

如果设置的属性节点不存在，那么系统会自动新增

[2].removeAttr作用：删除所有找到元素指定的节点（如果想要删除多个属性，用空格分开）

C:\Users\ADMINI~1\AppData\Local\Temp\1540282730(1).png

#### 5.prop与removeProp

作用与attr和removeAttr一样。

注意：在操作属性节点具有true和false时，（checked,selected,disabled..）推荐使用prop

#### 6操作类相关的方法

[1].addClass

添加一个类

如果要添加多个，多个类名之间用空格隔开即可

[2].removeClass

删除一个类

如果要删除多个，多个类名之间用空格隔开即可

[3].toggleClass

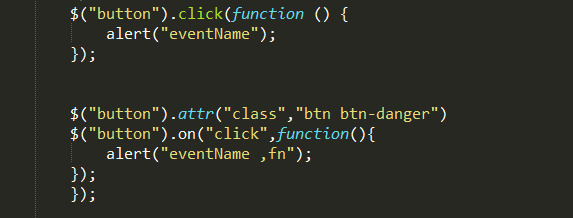
切换类

有就删除，没有添加

#### 7.jQuery中两种绑定事件方式

[1].eventName(fn) 编码效率高

[2].on(eventName,fn) 编码效率低



[3].事件移除 .off()



#### 8.事件冒泡和默认行为

[1].事件冒泡，事件会从低级向高级依次执行

[2].阻止事件冒泡



[4].默认行为

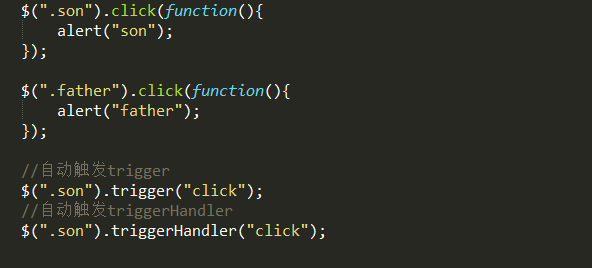
[5].阻止默认行为



#### 9.自动触发trigger

[1].利用trigger自动触发——会触发事件冒泡；会触发默认行为

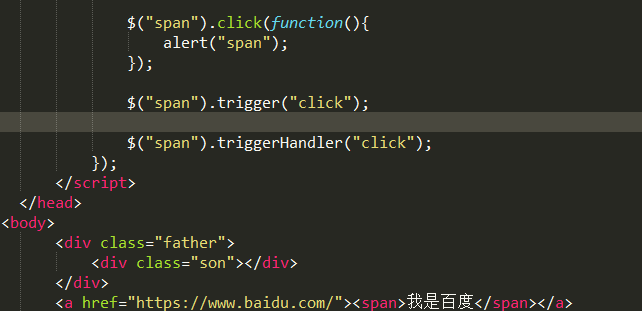
[2].利用triggerHandler自动触发——不会触发事件冒泡；不会触发默认行为



面试题：如果想要自动触发<a>，需要对<a>进行包装，监听<a>的子标签

如果想要触发a的事件并且触发a的行为使用trigger

如果想要触发a的事件但是不触发a的行为使用triggerHandler

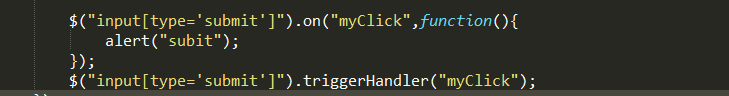


#### 10.自定义事件

自定义事件必须满足两个条件：

[1].事件必须是通过on来绑定

[2].事件必须是通过trigger或者是triggerHandler来触发

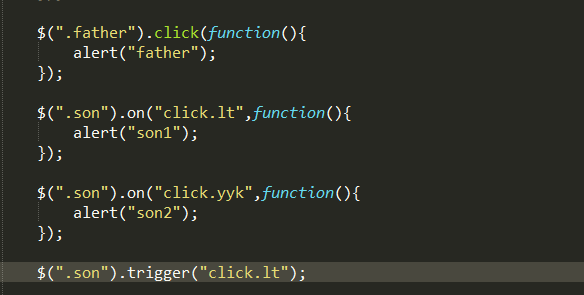


#### 11.事件命名空间

想要命名空间有效，必须满足两个条件

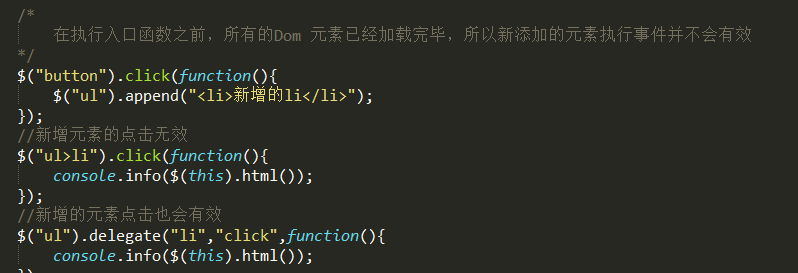
[1].事件必须是通过on来绑定

[2].事件必须是通过trigger或者是triggerHandler来触发



#### 12.事件委托 delegate

在jQuery中如果通过核心函数找到的元素不止一个，那么在添加事件的时候，jQuery会遍历所有找到的元素，给所有找到的元素添加事件

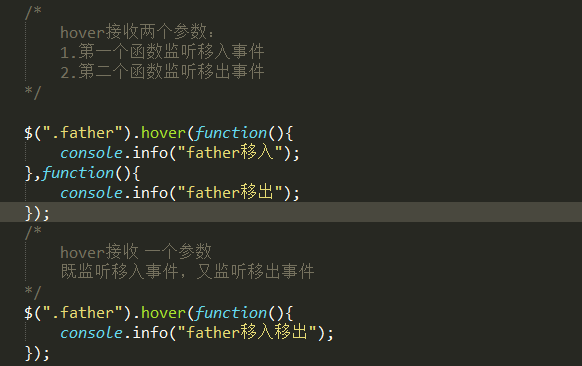


#### 13.移入移出事件

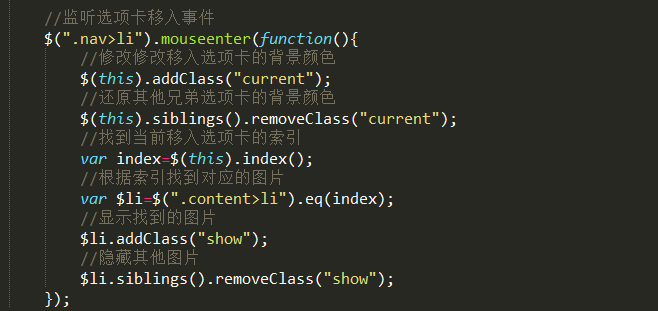
[1].mouseover\mouseout：子元素被移入移出也会触发父元素事件

[2].mouseenterr\mouseleave：子元素被移入移出不会触发父元素事件

[3].hover



#### 14．Tab选项卡 siblings



#### 15.显示、隐藏、切换动画



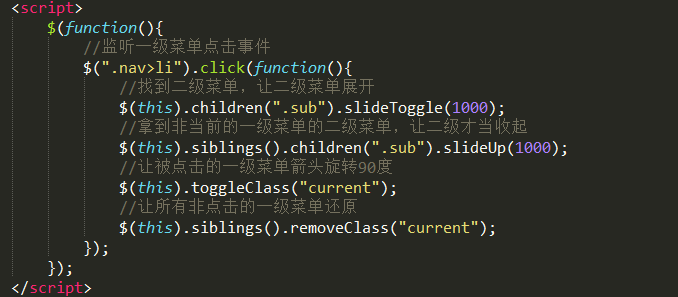
#### 16.展开、收起、切换**(slideDown,slideUp,slideToggle)**



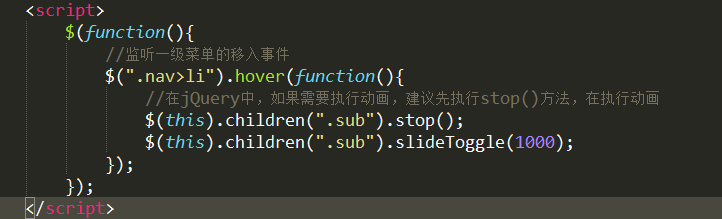
与显示、隐藏时间事件一样，传入一个参数设置时间，单位毫秒。

传入两个参数，第一个时间；第二个传入回调函数，动画执行完毕后调用。

**[折叠菜单]**



**[折叠菜单]**

****

#### 17.淡入、淡出



**[弹出广告]**

****

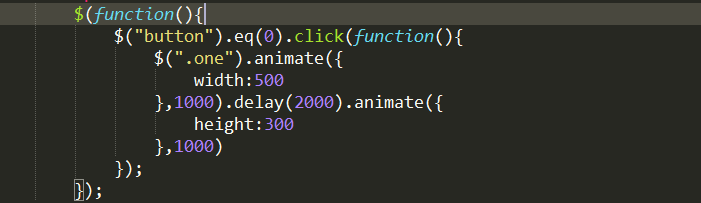
#### 18.自定义动画





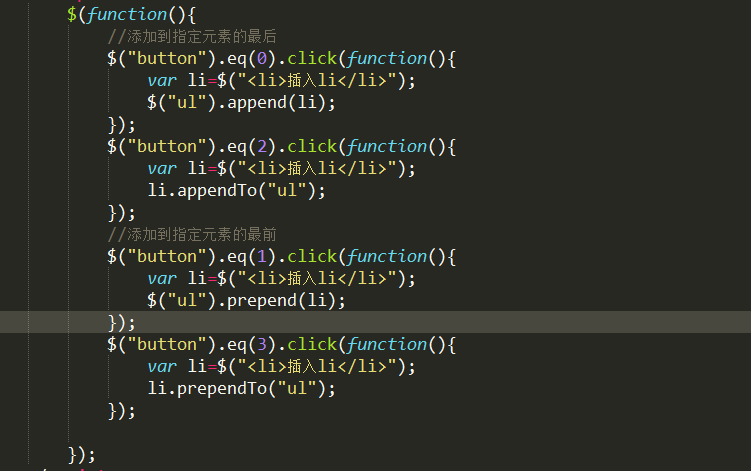
#### 19.stop和delay方法

[1].Delay方法的作用就是系统延迟时长



#### 20.添加节点

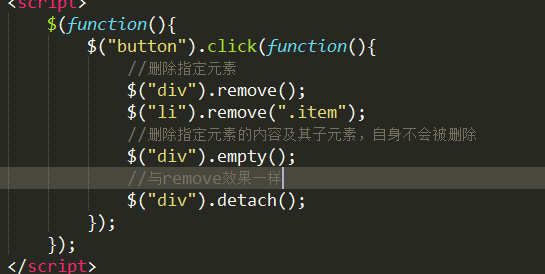
[1].内部插入



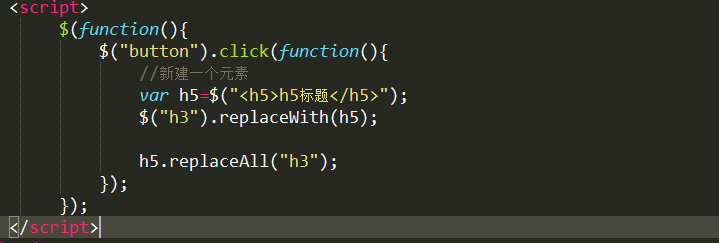
[2].外部插入



#### 21.删除节点



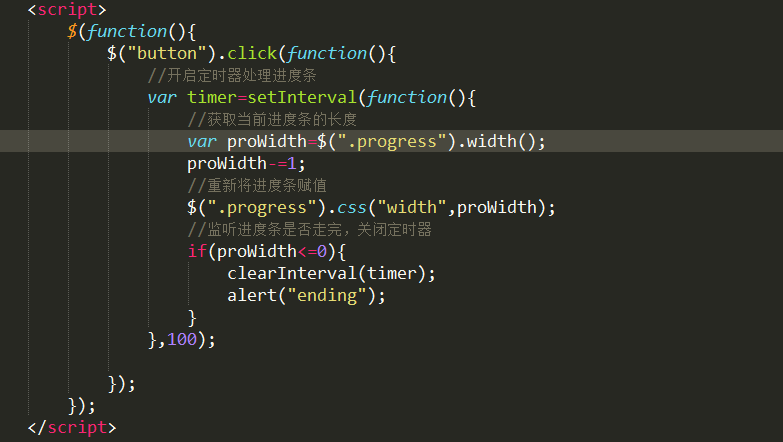
#### 22.替换节点



#### 23.复制节点



#### 24.处理进度条



#### 25.jQuery原理

[1].jQuery本质是一个闭包(可以自动执行)

[2].jQuery为什么要使用闭包来实现？为了避免多个框架的冲突。

[3].jQuery如何让外界访问内部定义的局部变量？window.xxx=xxx

[4].jQuery为什么要给自己传递一个windows参数？

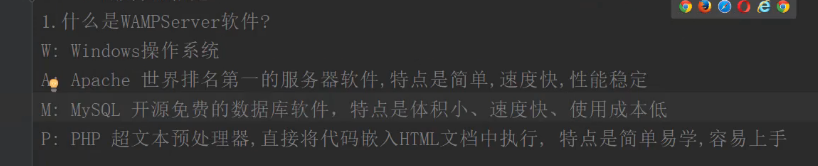
为了后期方便压缩代码

为了提升查找效率

[5].jQuery为什么要给自己接收一个windows参数？

IE9以下的浏览器undefined可以被修改，为了保证内部的undefined不被修改，所以需要接收一个正确的undefined

#### 26.AJAX

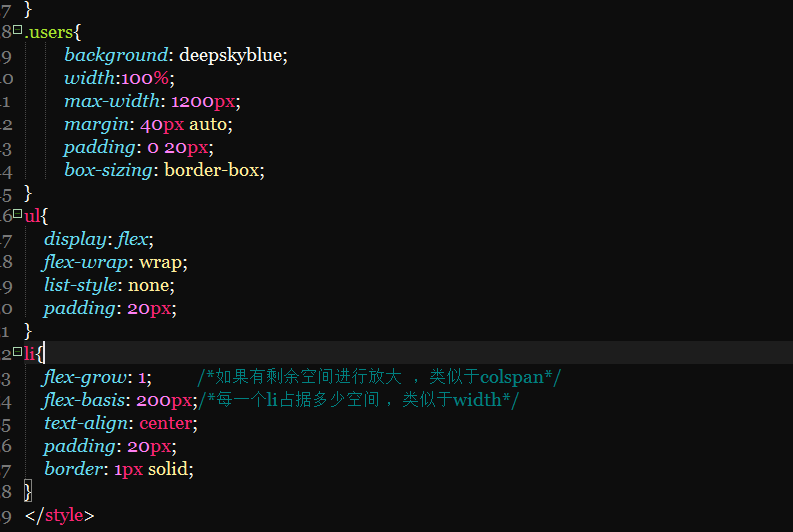


#### CSS样式积累

1. 列表去默认符号：list-style：none;
2. 行级元素变块级元素：display: inline-block;
3. 行前空字数：text-indent：（num）em;
4. 旋转，缩放：transform：(num)deg;
5. 元素监听事件：.sub>li:hover{ /\*监听hover事件 \*/

background: red;

}

1. 背景色透明：background:rgba(0,0,0,0.6);
2. ntn-child();
3. 应用文本阴影：style="text-shadow: black 5px 3px 3px;"
4. 

# 12.SpringBoot

#### 1.一个项目可以由 多个小型服务构成（微服务）

#### 2.SpringBoot可以快速开发 微服务模块

a.简化javaEE开发

b.整个Spring技术栈的整合（整合SpringMVC Spring）

c.整个JavaEE技术的而整合（MyBatis Redis）

#### 3.目录结构resource

Static:静态资源（js css 图片 音频 视频）

Templates:模板文件

Application.properties ：配置文件

#### 4．SpringBoot内置了Tomcat ，并且不需要打成war包在执行

#### 5.可以再application.properties对端口号等服务端信息进行设置

#### **6.Spring boot**

.Spring boot将各个应用三方框架设置成了有个场景 starter，以后用哪个，还需要引入哪个场景，选完后Springboot就会将该场景所需要的依赖自动注入

#### 7.@SpringBootApplication主配置类

主要包含的注解：

[1].@SpringBootConfiguration ：

包含@Configuration 表示“配置类” 1.该类是一个配置类

[2.加了@Configuration注解的类，会自动](mailto:2.加了@Configuration注解的类，会自动) 纳入到Spring容器中

@Configuration

Punlic class A{ //A是用于配置的类

}

[2]. @EnableAutoConfiguration 是SpringBoot可以自动配置，可以找到@SpringBootApplication主配置类所在的包。

作用：就会将该包及所有的子包全部纳入spring容器

SpringBoot在启动时，会根据META-INF/spring.factories找到相应的三方依赖，并且将这些依赖引入本项目

总结：

自己编写项目时，一般会对自己编写的的代码 以及 三方依赖 进行配置，但是SpringBoot会自动帮我们配置。

* 1. 自己写的代码：SpringBoot 通过@SpringBootConfiguration自动帮我们配置。
  2. 三方依赖：通过 spring-boot-test-autoconfigure-2.1.0.RELEASE.jar中的META-INF/spring.factories自动帮我们配置。spring-boot-test-autoconfigure-2.1.0.RELEASE.jar包中包含了J2EE整合体系中需要的依赖
  3. 如何自动装配：

研究org.springframework.boot.autoconfigure.web.servlet.HttpEncodingAutoConfiguration,\

通过观察该源码发现：

@Configuration:标识此类是一个配置类，将此类纳入SpringIOC容器中

@EnableConfigurationProperties(HttpProperties.class)：通过HttpProperties设置了编码（UTF-8,即自动装配编码为UTF-8.）

那么如何修改默认编码？

通过修改HttpProperties的prefix+属性名 进行修改。[在配置文件中：yml/properties ]

该注解给了默认编码UTF\_8,并且提供了 prefix+属性名 的方式供我们修改编码。

@ConditionalOnProperty(prefix = "spring.http.encoding", value = "enabled", matchIfMissing = true):

当属性（Property）满足条件时，条件成立。要求：如果没有配置spring.http.encoding. enabled=xxx;

总结： 1.每一个XXXAutoConfiguration都有很多条件@ConfigurationOnXX,当这些条件满足时，则此配置自动装配生效（UTF-8），但是我们可以手动修改自动装配，XXXproperties文件中的prefix.属性名=value.

2.全局配置文件中的key,来源于某properties中的prefix.属性名

d.如何找到Spring Boot中开启了哪些自动装配，禁止了哪些自动装配

在application.properties中 debug=true;

Positive matches:列表表示springboot自动开启的装配

Negative matches: 列表表示springboot此时没有开启的自动装配

#### 8.配置文件

作用：SpringBoot自动配置(约定：8080)，可以使用配置文件对默认配置进行修改

默认全局配置文件：

Application.properties key=value;

Application.yml 不是一个标记文档

注意点:1.key:空格value 或者行内样式

2.通过垂直对齐指定层次结构

3.默认不写引号 ：“”会将其中的转义字符进行转义，其他不会

Server：

Port： 6611

Path： a/b/c

Xml文件 是一个标记文档

<server>

<port>6611</port>

<path> a/b/c</path>

</server>

#### 9.通过yaml给对象注入值

[1].注入值

student:

name: 张三

age: 18

sex: **true**

birthday: 2008/2/2

[2].绑定

@Component //将此javaBean纳入Spring容器

@ConfigurationProperties(prefix="student")

**public** **class** Student {

**private** String name;

**private** **int** age;

@ConfigurationProperties（properties/yml）

@Value(“XXX”)

区别：

@ConfigurationProperties @Value

注值 批量注入 单个

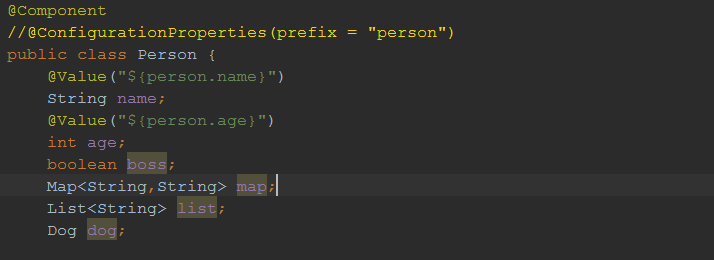
松散语法 支持 不支持

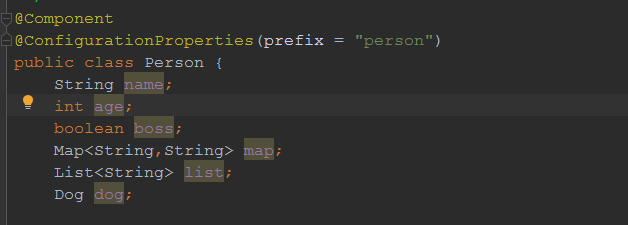
SpringEL 不支持 支持

JSR303数据校验 支持 不支持

注入复杂类型 支持 不支持

简单类型（8种基本类型String/Date）



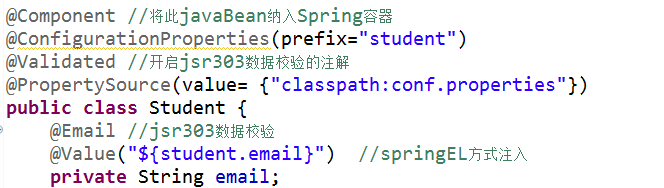


#### [10.@PropertySource](mailto:10.@PropertySource) :

默认加载application.properties/application.yml文件中的数据

例如：@PropertySource(value={"classpath:conf.properties"})加载 conf.properties文件中的数据。

但是，@PropertySource只能加载properties，不能加载yml



#### 11．@ImportResource

springBoot自动装配/自动配置

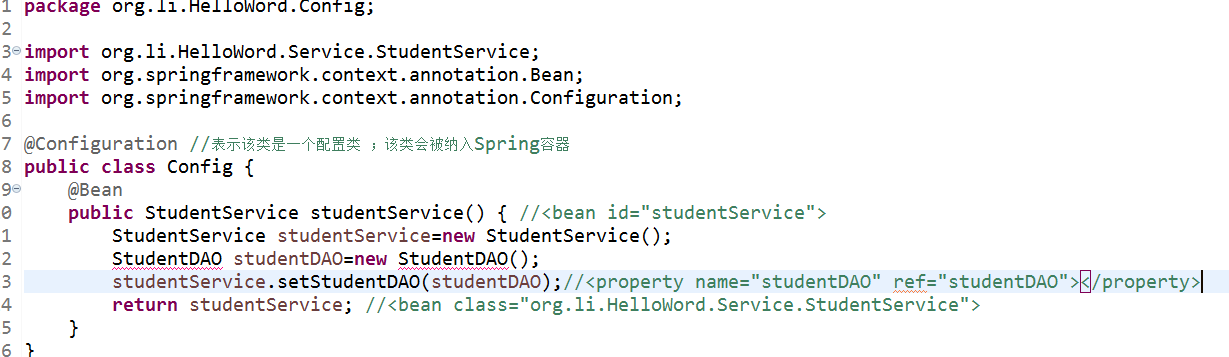
spring等配置文件 默认会被springboot自动给配置好

自己编写的spring等配置文件，springboot默认不识别。如果需要识别，在主配置类上通过@ImportResource（location={“classpath：spring.xml”}）指定文件的路径o.

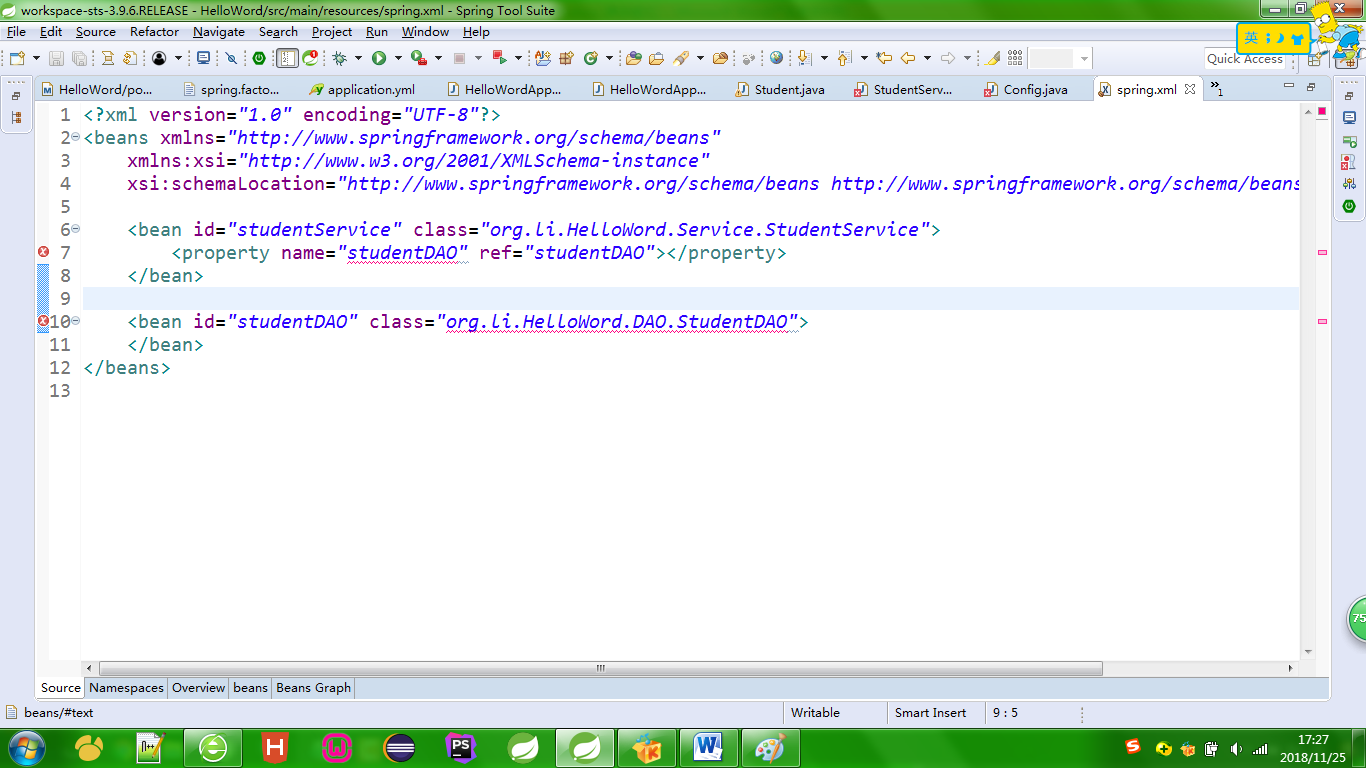
并不推荐手写。

SpringBoot推荐使用注解形式进行配置：@Configuration @Bean

配置类 等价于 spring.xml



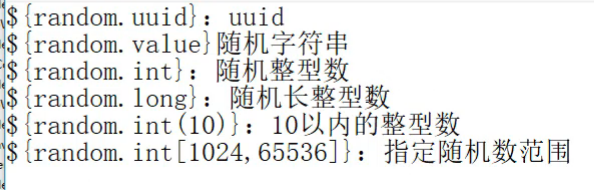
将方法的返回值添加到容器中；容器中这个组件默认的ID就是方法名



#### 12.占位符

Springboot全局配置文件中的占位符表达式

[1].随机数



[2].引用变量值

yml文件：

student:

name: ${student.user.name2:无效}

#age: 18

sex: **true**

birthday: ${student.user.date}

实际引用的是properties中的student.user.date=2003/2/4

#### 13.多环境切换profile

以端口号测试为例

a.properties

默认boot会读取application.properties环境8881

application-环境名.properties

application-dev.properties （开发环境8882）

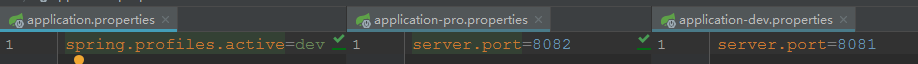
application-test.properties(测试环境8883)

如果要选择某一具体的环境：在application.properties中指定：spring.profiles.active=环境

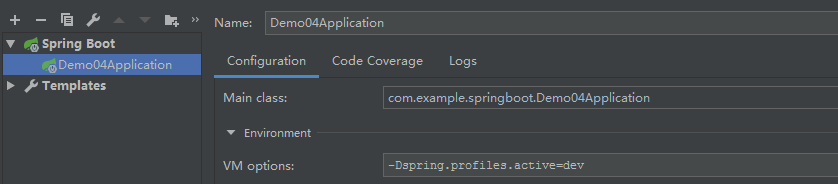
如果将application.properties注释掉，Spring boot仍然会读取其他application-环境名.properties文件中的数据

切换指定环境的方式

I.



II.



b.application.yml

server:

port: 8081 #第一个环境（主环境）

spring:

profiles:

active: dev #指定本次采用的环境

--- #采用---作为分割线

server:

port: 8083

spring:

profiles: dev #第二个环境

---

server:

port: 8082

spring:

profiles: test #第三个环境

7.要想在项目中改变自己想使用的版本号，找到最顶级的parent,在pom.xml文件中的<properties>

<java.version>1.8</java.version>

< /properties>

改变version.

8. 为了防止用Maven管理Spring项目时，不同的项目依赖了不同版本的Spring，可以使用Maven BOM来解决者一问题。

在依赖管理时，引入spring-framework-bom，如：

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-framework-bom</artifactId>

<version>4.3.1.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

在引入BOM之后，在引入其他Spring依赖时，都无需指定版本，如：

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

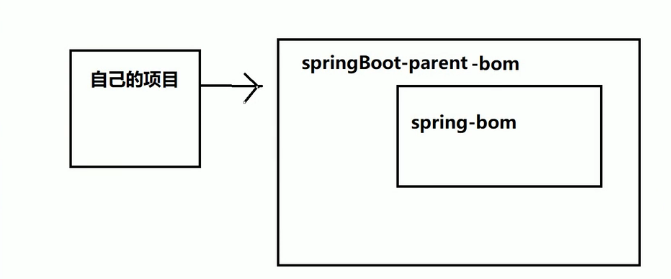
</dependency>

<dependency>

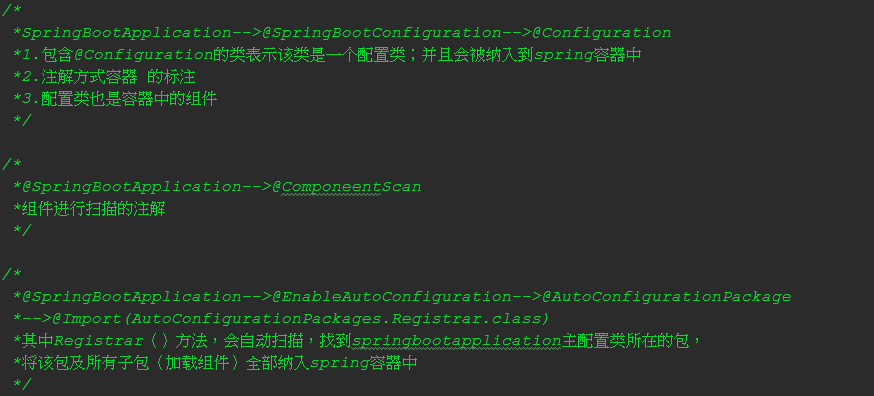
<groupId>org.springframework</groupId>

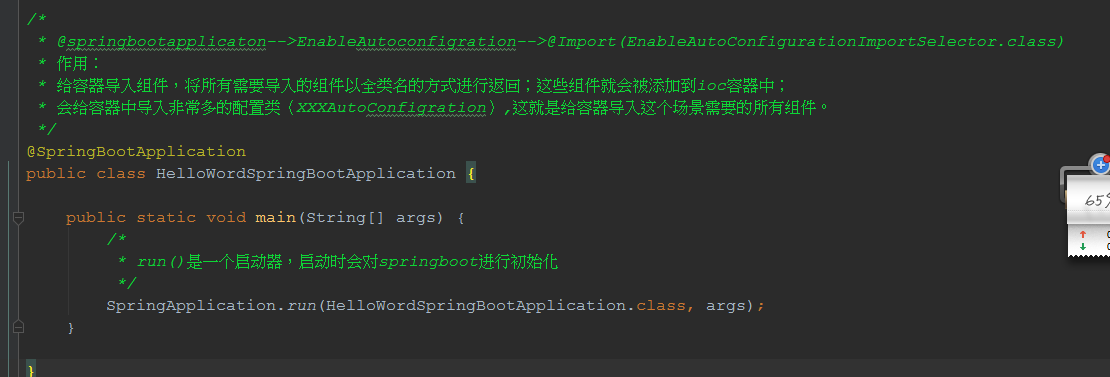
<artifactId>spring-web</artifactId>

</dependency>

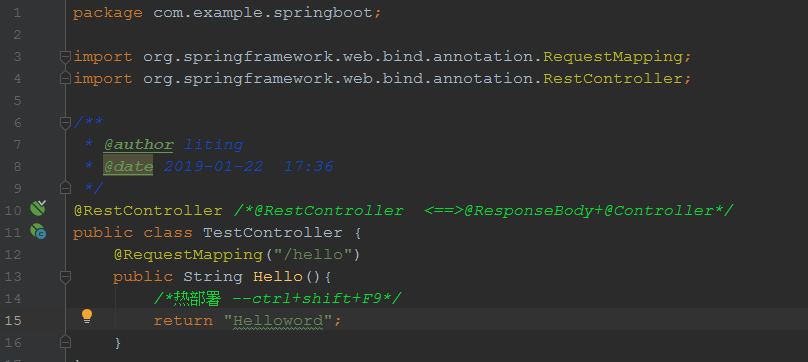
<dependencies>

## 1.第一个程序

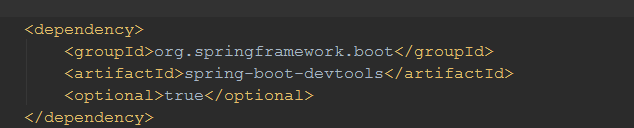




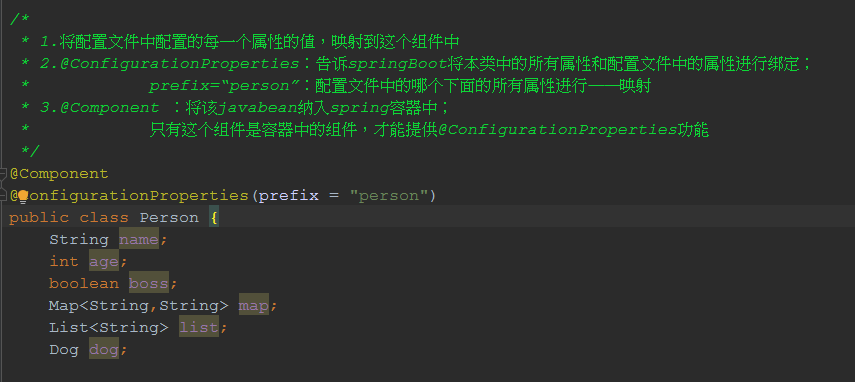
## 2．热部署

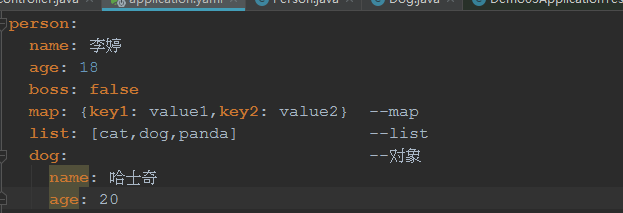


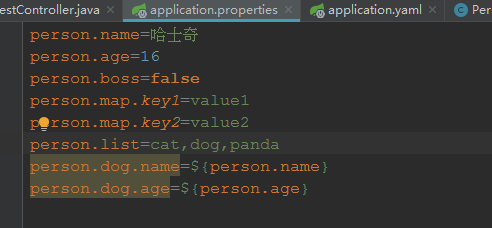
引入依赖

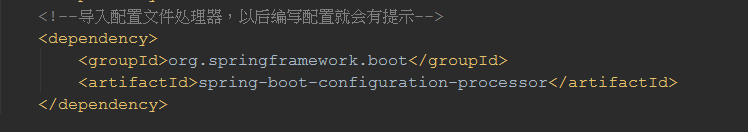


## 3.yaml配置文件

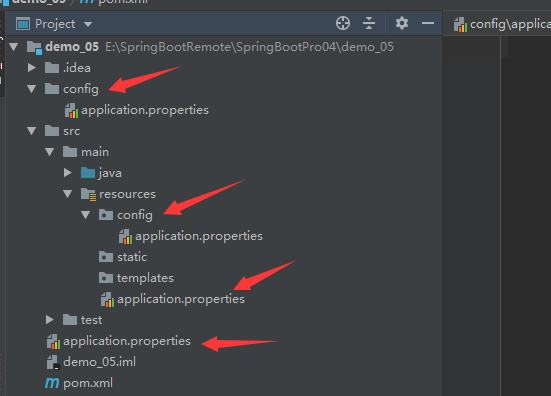








## 4.SpringBoot配置文件加载位置和优先级



file:/ 🡪2

classpath:/ 🡪4

classpath:./config 🡪3

classpath:./config 🡪3

File:./config 🡪1

--fille:./config

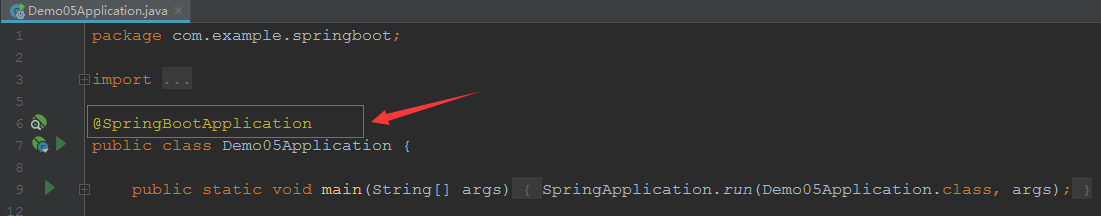
--file:./

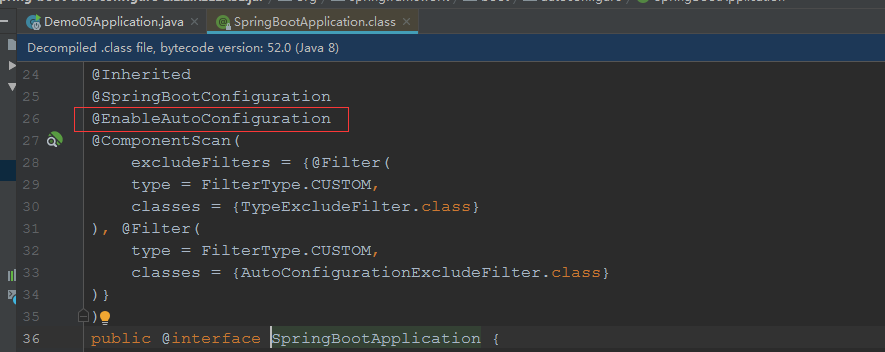
--classpath:/config/

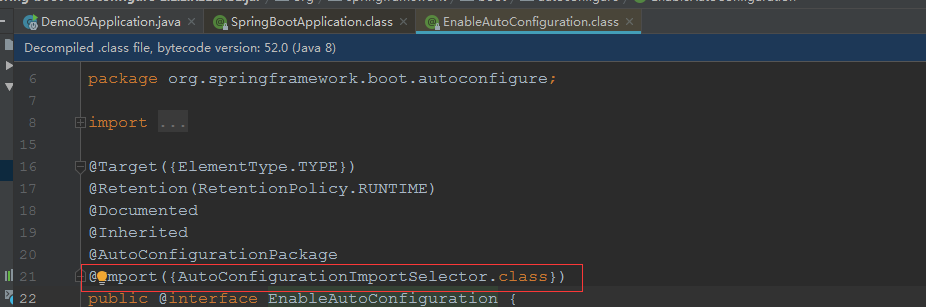
--classpath:/

## 5.SpringBoot自动配置原理

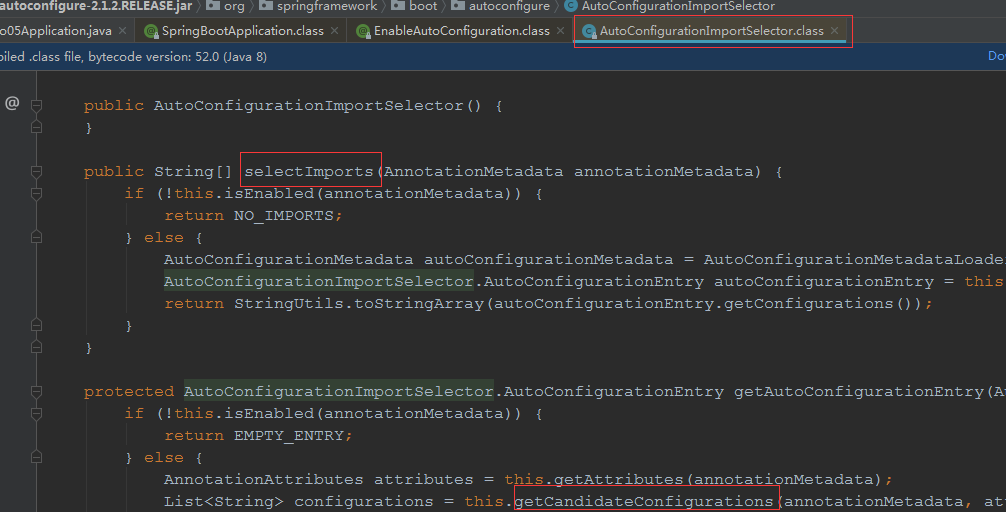
<http://docs.spring.io/spring-boot/docs/1.5.9.RELEASE/reference/htmlsingle/#common-application-properties>



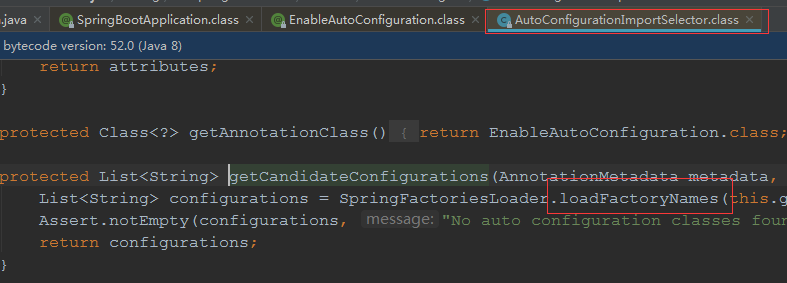


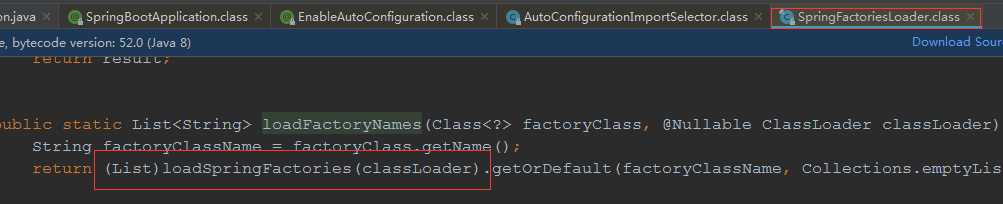


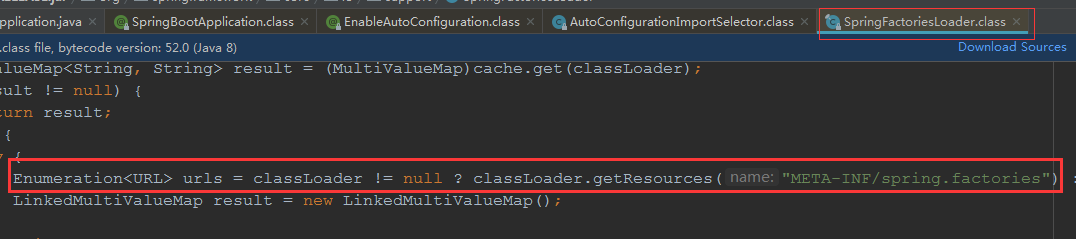
自动配置导入的选择器



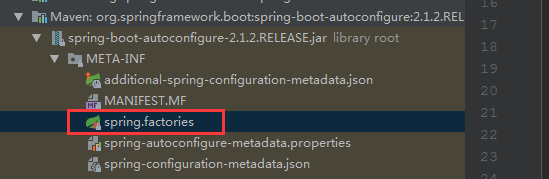
获取候选的配置



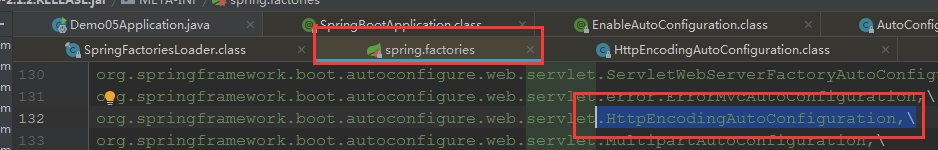


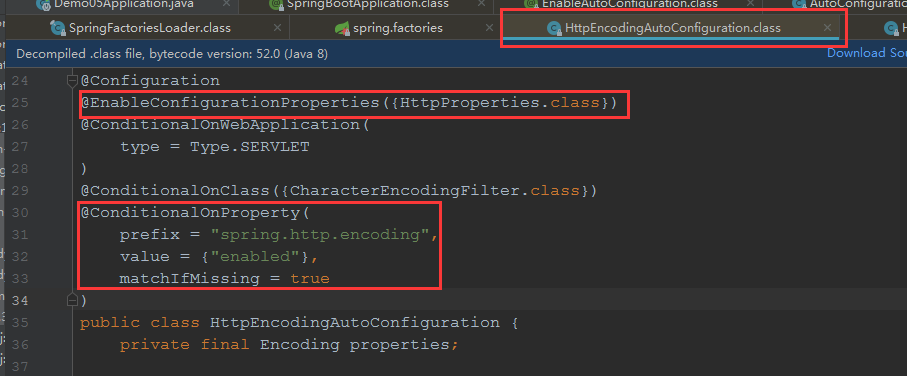


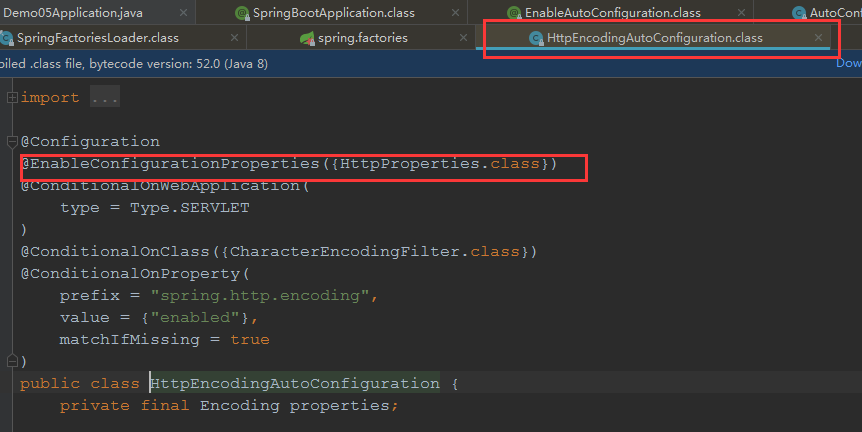
SpringBoot启动时会自动读取META-INF/spring.factories里边的自动配置类；这些配置类（@Bean）会被纳入ioc容器中。

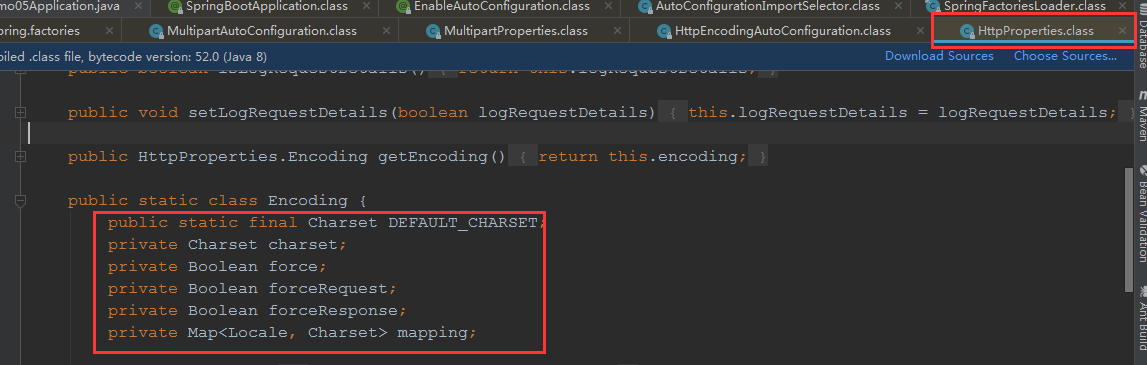


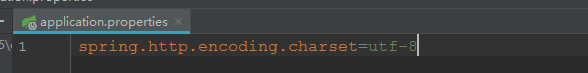
@EnableConfigurationProperties（启用配置）注解使@ConfigurationProperties注解生效











# 13三层架构

三层架构(3-tier application) 通常意义上的三层架构就是将整个业务应用划分为：表现层（UI）、业务逻辑层（BLL）、数据访问层（DAL）。区分层次的目的即为了“高内聚，低耦合”的思想。

　１、表现层（UI）：通俗讲就是展现给用户的界面，即用户在使用一个系统的时候他的所见所得。

２、业务逻辑层（BLL）：针对具体问题的操作，也可以说是对数据层的操作，对数据业务逻辑处理。

　３、数据访问层（DAL）：该层所做事务直接操作数据库，针对数据的增添、删除、修改、更新、查找等。

MVC是 Model-View-Controller，严格说这三个加起来以后才是三层架构中的WEB层，也就是说，MVC把三层架构中的WEB层再度进行了分化，分成了控制器、视图、实体三个部分，控制器完成页面逻辑，通过实体来与界面层完成通话；而C层直接与三层中的BLL进行对话。

# 14.控制反转

IOC—Inversion of Control，即“控制反转”，不是什么技术，而是一种设计思想。

传统的javaSE程序设计，是直接在对象内部通过new创建对象，是程序主动去创建这些对象，而IOC是有专门的一个容器去创建对象，即由IOC容器控制对象的创建，反转就是容器来帮忙创建及注入依赖对象；

# 15.Struts2

1.struts2是基于MVC设计模式的web应用程序框架。

2.目的：减少在运用MVC设计模式来开发web应用的时间。

3.MVC：是模型视图控制器，一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。

javaBean:

Model: DAO:数据库操作

Service:业务逻辑

View: jsp 显示用户看的页面

Controller：servlet 页面跳转



Struts2是webwork的升级，同时吸收了两者的优势。



<http://struts.apache.org/>

#### Struts2工作原理

客户端提出一个请求（HttpServletRequest），请求被提交到一系列过滤器（ActionContextCleanUp[\*用于和其他过滤器的集成]、otherFilter 、FilterDispatcher[\*struts2.0 和2.1.2之间版本的核心过滤器，2.1.3开始使用StrutsPrepareAndExecuteFilter]）,ActionMappe(web.xml)r会拦截.action的请求，找到ActionProxy通过configurationManager读struts.xml,找到具体的action类，通过ActionProxy代理创建反向实例Actioninvocation经过一系列拦截器执行action,返回result（jsp），在经过一系列拦截器，通过HttpServletResponse返回用户实例。

#### 2.SpringMVC工作原理：

jsp发送以.actin结尾的请求，dispatcherServlet就会拦截请求，跳到-servlet.xml找到相应的处理，跳到handlermapping,跳到controller,返回类型为modelandview的对象，跳到viewresolver ,返回数据结果及视图给用户。

#### 3区别

1.

Struts2 的核心是基于一个Filter即StrutsPreparedAndExcuteFilter

SpringMvc的核心是基于一个Servlet即DispatcherServlet(前端控制器)

2.

Struts2是基于类开发的，传递的参数是通过类的属性传递(属性驱动和模型驱动),所以只能设计成多例prototype

SpringMvc是基于类中的方法开发的，也就是一个url对应一个方法，传递参数是传到方法的形参上面，所以既可以是单例模式也可以是多例模式singiton

3.

Struts虽然方法之间也是独立的，但时action变量是可以共享的

springmvc方法之间基本独立，独享requset和response之间的数据。请求数据通过参数获取，处理结果通过model Map交回给框架。（方法之间不共享变量）

4.

SpringMVC处理ajax的请求很方便，只需一个注解2responseBody ,直接返回响应文本。

5.

intercepter的实现机制。Struts 有自己的拦截器机制.interceptor,Springmvc用的是独立的AOP方式,导致Springmvc的配置文件比struts2的配置文件多

#### 4核心文件

Web.xml

任何mvc框架都需要与web应用整合，这就不得不借助于web.xml文件，只有配置在web.xml中servlet才会被加载（model2=jsp+javaBean+servlet）

通常mvc框架需要借助web.xml文件加载核心控制器，对于struts2而言，需要加载StrutsPrepareAndExecuteFilter，只要web.xml负责加载StrutsPrepareAndExecuteFilter，StrutsPrepareAndExecuteFilter将会加载struts2 框架。

Struts.Xml

Struts2核心文件，该文件主要负责管理应用中的action映射，以及action包含的result定义等。

包含：

1.全局属性。

2.用户请求和响应action之间的对应关系。

3.action可能用到的参数和返回结果。

4.各种拦截器的配置

Struts.properties

Struts2框架的全局属性文件，自动加载。包含很多key-value对。

该文件可以完全配置在struts.xml文件中，使用constant元素

模板文件：

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

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<!--include节点是struts中 组件化 的方式，可以将每个功能模块独立到一个xml配置文件中，然后用include节点引用 -->

<include file=*"struts-default.xml"*></include>

<!--package提供了将多个action组织为一个模块的方式

package的名字必须是唯一的，package可以扩展，

当一个package扩展自另一个package时候package会在本身配置的基础上加上扩展的package的配置

父package必须在子package前配置

name:package名称

extends:继承的父package名称

acstract:设置package 属性为抽象的，抽象的package 不能定义action。值：true\false

namespace:packege命名空间 该命名空间影响到url地址，例如命名空间为/test 那么访问的地址就是：http:localhost:8080/test/hello.action-->

<package name=*"default"* namespace=*"/"* extends=*"struts-default"*>

<interceptors>

<!--定义拦截器

name:拦截器名称

class:拦截器类路径 -->

<interceptor name=*"timer"* class=*"com.liting.timer"*></interceptor>

<interceptor name=*"logger"* class=*"com.liting.logger"*></interceptor>

<!--定义拦截器栈 -->

<interceptor-stack name=*"mystack"*>

<interceptor-ref name=*"timer"*></interceptor-ref>

<interceptor-ref name=*"logger"*></interceptor-ref>

</interceptor-stack>

</interceptors>

<!--定义默认的拦截器 每个action会自动引用

如果action中引用了其他的拦截器 默认的拦截器将无效 -->

<default-interceptor-ref name=*"mystack"*></default-interceptor-ref>

<!--全局result配置 -->

<global-results>

<result name=*"input"*>/error.jsp</result>

</global-results>

<!--action配置一个action可以被多次映射（只要action中的name不同）

name:action名称

class: 对应的类的路径

method:调用action 中的 方法名 -->

<action name=*"helloworld"* class=*"com.liting.action.HelloWorldAction"*>

<!-- 引用拦截器

name:拦截器名称或者拦截器栈名称 -->

<interceptor-ref name=*"timer"*></interceptor-ref>

<!--节点配置：

name:result名称和Action中返回的值相同

type:result类型不写则选用superpackage的type struts-defaul.xml中默认的dispatcher -->

<result name=*"success"* type=*"dispatcher"*>/result.jsp</result>

<!--参数设置

name:对应action的 get/set方法 -->

<param name=*"url"*>http://www.sina.com;</param>

</action>

</package>

</struts>

MySql分页：

select \* from t\_traexa\_questions\_plan WHERE PLAN\_NAME LIKE '%人%' LIMIT 2,2;

Oracle分页：

SELECT \*FROM

(SELECT ROWNUM rn ,\*FROM student WHERE Rownum <= pageNo \*pageSize)

WHERE rn > (pageNo - 1) \* pageSize

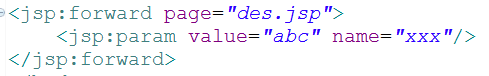
# 16

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | include指令 | |  | include标准动作 | |
|  |  |  |  |  |  |
| 语法 | <%@ include> |  |  | <jsp:include> |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 属性 | file |  |  | page |  |
|  |  |  |  |  |  |
| 包含 | 静态包含 |  |  | 动态包含 |  |
|  |  |  |  |  |  |









 Sleep()和wait()的区别？

1、这两个方法来自不同的类分别是Thread和Object

2、最主要是sleep方法没有释放锁，而wait方法释放了锁，使得其他线程可以使用同步控制块或者方法。

3、wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在。

XML的解析方式分为四种：1、DOM解析；2、SAX解析；3、JDOM解析；4、DOM4J解析。

（一）

1.String str=new String("abc") 在内存中是怎么分配的？

答：在运行时才在内存创建一个内存空间。

2.String str1="abc" 和 String str1=new String("abc"); 这两个的区别是什么？

答：String str1="abc"在编译时已经在一个常量池中生成一个拷贝，其中str1中的"abc"是字符串常量。而对于String str1=new String("abc")的话，它是在运行时才创建。这两句在内在中所产生的对象并不是同一个对象。

3.GC是什么？有几种算法实现？(这题到现在也不大明白..如果GC是垃圾回收的话.后面为什么要问有几种算法实现呢？)

答：垃圾回收.可实现的算法有引用计数,标记回收,复制回收等.

4.Java Server Page 和　servlet 的联系与区别

5.collection和collections的区别？

答：Collection是集合类的上级接口，继承与他的接口主要有Set 和List.

　　Collections是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜　　　索、排序、线程安全化等操作。

=====================================================

6.ArrayList和Vector的区别，HashMap和HashTable的区别？

答： Vector是同步的。这个类中的一些方法保证了Vector中的对象是线程安全的。而ArrayList则是异步的，因此ArrayList中的对象并 不是线程安全的。因为同步的要求会影响执行的效率，所以如果你不需要线程安全的集合那么使用ArrayList是一个很好的选择，这样可以避免由于同步带 来的不必要的性能开销。

Hashtable和HashMap的区别：

1.Hashtable是Dictionary的子类，HashMap是Map接口的一个实现类；

2.Hashtable中的方法是同步 的，而HashMap中的方法在缺省情况下是非同步的。即是说，在多线程应用程序中，不用专门的操作就安全地可以使用Hashtable了；而对于 HashMap，则需要额外的同步机制。但HashMap的同步问题可通过Collections的一个静态方法得到解决：

Map Collections.synchronizedMap(Map m)

这个方法返回一个同步的Map，这个Map封装了底层的HashMap的所有方法，使得底层的HashMap即使是在多线程的环境中也是安全的。

3. 在HashMap中，null可以作为键，这样的键只有一个；可以有一个或多个键所对应的值为null。当get()方法返回null值时，即可以表示 HashMap中没有该键，也可以表示该键所对应的值为null。因此，在HashMap中不能由get()方法来判断HashMap中是否存在某个键， 而应该用containsKey()方法来判断。

HashMap是Hashtable的轻量级实现（非线程安全的实现），他们都完成了Map接口，主要区别在于HashMap允许空（null）键值（key）,由于非线程安全，效率上可能高于Hashtable。

HashMap允许将null作为一个entry的key或者value，而Hashtable不允许。

HashMap把Hashtable的contains方法去掉了，改成containsvalue和containsKey。因为contains方法容易让人引起误解。

Hashtable继承自Dictionary类，而HashMap是Java1.2引进的Map interface的一个实现。

最大的不同是，Hashtable的方法是Synchronize的，而HashMap不是，在多个线程访问Hashtable时，不需要自己为它的方法实现同步，而HashMap 就必须为之提供外同步。

Hashtable和HashMap采用的hash/rehash算法都大概一样，所以性能不会有很大的差异。

================================================

7.overload和override的区别？

override（重写）

1、方法名、参数、返回值相同。

2、子类方法不能缩小父类方法的访问权限。

3、子类方法不能抛出比父类方法更多的异常(但子类方法可以不抛出异常)。

4、存在于父类和子类之间。

5、方法被定义为final不能被重写。

overload（重载）

1、参数类型、个数、顺序至少有一个不相同。

2、不能重载只有返回值不同的方法名。

3、存在于父类和子类、同类中。

8.char能不能存储汉字？

答：java char型变量是能够定义成为一个中文的，因为java中以unicode编码，一个char占16个字节，所以放一个中文是没问题的

（二）

1.interface 和 abstract 的区别？

(1)关于抽象类和接口，我想还是看具体的要求，什么模式之类的东西

都是有了应用才出现的，需求才是第一位的。

它们之间的区别是：

1. 在抽象类中可以实现某些方法，而接口不行

2. 抽象类是用来extends的，而继承有一定的局限性，只能继承一个类

而接口就不同，可以同时实现多个接口

这两者也可以联合使用。看具体的要求了。如果有共同方法的那些类

它们的共同的方法的实现是一样的，那么可以考虑用抽象类，把方法

在抽象类中实现，如果还有一些方法定义相同而实现不同，那么可以

考虑定义一个接口，让抽象类来implements这个接口

然后各个子类只用继承抽象类即可。

(2)接口是“like a ”象一个什么东西、或是具备什么功能

抽象类是 “ is a ”是一个什么东西。

一个带锁的门，那么一定是 “somthing extends 门 impliment 锁 ”

决不能说 “带门的锁”

(3)经过自己写代码的测试结果：I代表interface,A代表abstract

I implemets I 错

I extends I 对

I extends A 错

(4)一个是接口，一个是虚函数 java中的接口相当C++中纯虚函数

也就是一个也不实现，具体是在子类中实现，这体现了多态性

同时java接口也是java实现多重继承的一个方法

1.interface的所有函数都是abstract的；

2.interface的数据成员默认都是public static final的；

3.Interface可以被多继承，abstract class单继承

interface是接口，abstract class是抽象类 主要区别：

1、接口可以多重implement(实现),抽象类只能单一extends(继承)

2、接口内只能是功能的定义，抽象类中可以包括功能的定义和功能的实现。

3、接口字段定义默认为public static final,而抽象类的则默认为friendly。

3.GC是什么？常用的两种方法是什么？

(1)GC是垃圾收集的意思（Garbage Collection）,内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java提供的GC功能可以 自动监测对象是否超过作用域从而达到自动回收内存的目的，Java语言没有提供释放已分配内存的显示操作方法。

(2)答案：System.gc()用来强制立即回收垃圾，即释放内存。

java 对内存的释放采取的垃圾自动回收机制，在编程的时候不用考虑变量不用时释放内存，java虚拟机可以自动判断出并收集到垃圾，但一般不会立即释放它们的内 存空间，当然也可以在程序中使用System.gc()来强制垃圾回收，但是要注意的是，系统并不保证会立即进行释放内存.

gc首先要判断该对象是否是时候可以收集。两种常用的方法是引用计数和对象引用遍历。

==========================================================

4. 线程常用的方法是什么？

Thread类的常用方法：

static Thread currentThread() //静态方法，通过这个方法可以获得主线程的引用，从而达到操作主线程的目的。

static int activeCount() //静态方法，获得当前活动线程数量

long getId() //获得当前线程id

String getName() //获得当前线程名字

int getPriority() //获得优先级

boolean isAlive() //当前线程是否处于活动状态。

boolean isDaemon() //是否为守护线程

void run() //run()方法，我们用线程，主要就是对run()方法的重写

void start() // 使该线程开始执行；Java 虚拟机调用该线程的run() 方法。

void sleep() //使当前线程休眠，以执行其他线程，如Thread.sleep(1000) 休眠1秒

5. Sleep()和wait()的区别？

搞线程的最爱sleep()方法是使线程停止一段时间的方法。在sleep 时间间隔期满后，线程不一定立即恢复执行。这是因为在那个时刻，其它线程可能正在运行而且没有被调度为放弃执行，除非(a)"醒来"的线程具有更高的优先 级，(b)正在运行的线程因为其它原因而阻塞。wait()是线程交互时，如果线程对一个同步对象x 发出一个wait()调用，该线程会暂停执行，被调对象进入等待状态，直到被唤醒或等待时间到。

6. Servlet中forward()和redirct()的区别？

答:前者仅是容器中控制权的转向，在客户端浏览器地址栏中不会显示 出转向后的地址;后者则是完全的跳转，浏览器将会得到跳转的地址，并重新发送请求链接。这样，从浏览器的地址栏中可以看到跳转后的链接地址。所以，前者更 加高效，在前者可以满足需要时，尽量使用forward()方法，并且，这样也有助于隐藏实际的链接。在有些情况下，比如，需要跳转到一个其它服务器上的 资源，则必须使用sendRedirect()方法。

7. EJB和JavaBean都是什么？

答案：

javabean是客户端组件，EJB是服务器端组件。

其实ejb与javaBean的最根本区别是：分布性。这是关键！！！

Java Bean 是可复用的组件，对Java Bean并没有严格的规范，理论上讲，任何一个Java类都可以是一个Bean。但通常情况下，由于Java Bean是被容器所创建(如Tomcat)的，所以Java Bean应具有一个无参的构造器，另外，通常Java Bean还要实现Serializable接口用于实现Bean的持久性。Java Bean实际上相当于微软COM模型中的本地进程内COM组件，它是不能被跨进程访问的。Enterprise Java Bean 相当于DCOM，即分布式组件。它是基于Java的远程方法调用(RMI)技术的，所以EJB可以被远程访问(跨进程、跨计算机)。但EJB必须被布署在 诸如Webspere、WebLogic这样的容器中，EJB客户从不直接访问真正的EJB组件，而是通过其容器访问。EJB容器是EJB组件的代 理，EJB组件由容器所创建和管理。客户通过容器来访问真正的EJB组件。

8. EJB都有哪些构成？

答案：Session Bean, Entity Bean, Message Driven Bean.

9. Servlet的生命周期，比CGI有什么优点？

web容器加载servlet，生命周期开始。通过调用servlet的init()方法进行servlet的初始化。通过调用 service()方法实现，根据请求的不同调用不同的do\*\*\*()方法。结束服务，web容器调用servlet的destroy()方法。

Servlet是Java平台上的CGI技术。Servlet在服务器端运行，动态地生成Web页面。与传统的CGI和许多其它类似CGI的技术相 比，Java Servlet具有更高的效率并更容易使用。对于Servlet，重复的请求不会导致同一程序的多次转载，它是依靠线程的方式来支持并发访问的。

14. 存储过程和函数的区别？

本质上没区别。只是函数有如：只能返回一个变量的限制。而存储过程可以返回多个。而函数是可以嵌入在sql中使用的,可以在select中调用，而存储过 程不行。执行的本质都一样。 函数限制比较多，比如不能用临时表，只能用表变量．还有一些函数都不可用等等．而存储过程的限制相对就比较少。

Oracle三大范式：

第一范式：（1NF）无重复列；

第二范式（2NF）:属性完全依赖于主键

通俗理解：不能部分依赖。即，候选码是组合键时（联合主键），非主属性要完全依赖于组合键，而不能依赖于组合键的部分属性。

第三范式（3NF）属性不能传递依赖于主键

通俗理解:不能存在传递依赖。 除了主键外，其他字段必须依赖主键

# 17 Git

XHuangya

liting8225

配置ssh:先在本地配置，发送给远程（ssh是本地计算机和远程计算机关联）

先在本地生成ssh

ssh-keygen -t rsa -C 1483401993@qq.com 一直回车

发送给远程

gitHub --> settings -->ssh and GPG keys --> new sshkeys

title 任意

key 粘贴本地的key

测试连通性

ssh -T git@github.com

如果本地和远程成功通信 ，可以再.ssh找到known hosts

第一次发布项目

$ git add . //文件->暂存区

$ git commit -m "注释信息"

$ git remote add origin https://github.com/XHuangYa/mygitRemote.git

$ git push origin -u master(默认主分支) //本地->远程

第一次下载项目

$ git clone https://github.com/XHuangYa/mygitRemote.git

提交

$ git add .

$ git commit -m "注释信息"

$ git push origin master(默认主分支)

更新（远程到本地）

$git pull

比对修改前后的文件

$ git diff 文件名

查看文件内容

$ cat 文件名

Git撤销修改

$ git checkout --文件名

[1].文件自动修改后，还没有放到暂存区——使用 撤销修改回到个版本库一样的状态

[2].文件已经放入暂存区，接着做了修改——使用 撤销修改就回到添加暂存区后的状态

git查看项目日志

$ git log

简化查看项目日志

$git log --pretty=oneline

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*版本切换\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

查看版本变化

$ git reflog

切换版本————1

$ git reset --hard head //hard 代表强行 head代表当前最新的

切换版本————2

$ git reset --hard 版本号 //$ git reflog可以查看版本号

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*分支管理\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

查看分支

$ git branch

创建分支

$ git branch 分支名

切换分支

$ git checkout 分支名

删除分支(已经合并到主分支的分支才可以删除)

$ git branch -d 分支名

创建并且切换到该分支

$ git checkout -b 分支名

git branch --marged ：查看哪些分支已经合并到当前分支

git branch --no-marged ：查看所有包含未合并工作分支

【在本地分支进行add,commit操作；切换回master主分支，merge本地分支；push到远程仓库】

EGit:在eclipse中使用git

配置:

1.Team-Git-Configrarion 用户名 邮箱

2.General-NetWork Conditions -SSH2 选中.ssh目录

第一次发布

Team-shareProject

add To index //加到暂存区

commit //提交到本地分支

remote-push //推送到远程

提交

commit and push 和commit 的区别

commit：不能单独push某一个文件，只能push项目

commit and push：可以 单独push某一个文件

第一次下载

File-import-Git-https/ssh

更新

Team-Pull

冲突解决方式 ——1

发现冲突：Team-synchronize workspace

解决冲突：

add and index //添加到本地暂存区

commit //提交到本地分支

pull //更新服务端的分之内容到本地分支

修改冲突 ，直接修改或者merge tool

————此时的文件已经变成普通的工作空间中的文件

add and index //添加到本地暂存区

commit and push 或者 remote-push //提交到远程分支

冲突解决方式 ——2

发现冲突：Team-pull //更新服务端的分支内容到本地分支

解决冲突：Team-stashes-stash changes //添加到本地暂存区

Team-pull //更新服务端的分支内容到本地分支

Team-stashes-stash changes-apply stashed changes //从本地暂存区应用

修改冲突 ，直接修改或者merge tool

Team-commit-add and index

commit and push 或者 remote-push //提交到远程分支

设置姓名：（gloabl 全局的）

$ git config --gloabl user.name "李婷"

设置邮箱：

$ git config --global user.email LiTing188225@yeah.net

切换目录

$ cd E://Test

创建空的文件夹

$ mkdir Git

切换到当前目录下的文件夹

$ cd Git

查看当前文件夹所有内容

$ ls

将当前目录初始化为Git仓库

$ git init

用于显示工作目录和暂存区的状态(当新增文件 或者 文件进行了修改 使用该命令)

$ git status

将文件提交到暂存区

$ git add .

将文件提交到本地版仓库 （-m message）

$ git commit -m "备注信息"

# 18.VUE

## 1.属性绑定

HTML

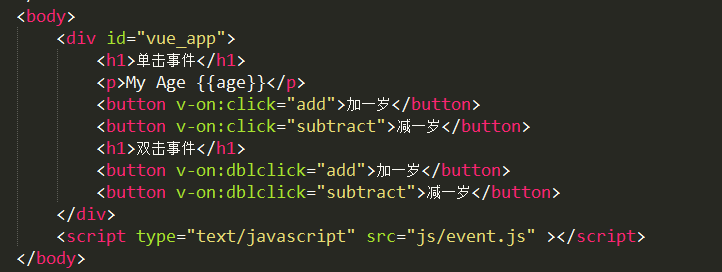


JS

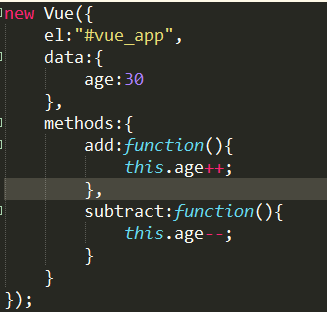


## 事件-点击、鼠标移动

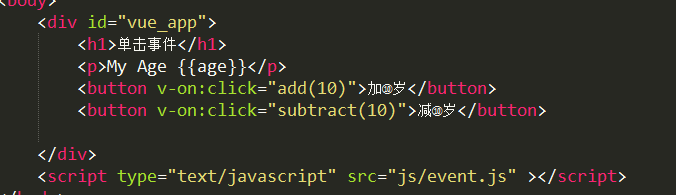
HTML-无参数

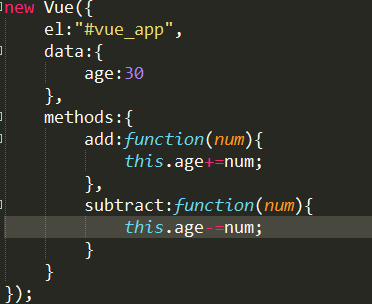


JS



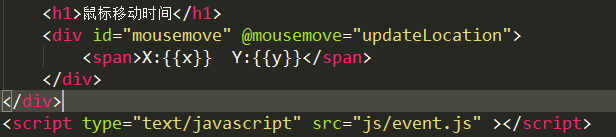
HTML-有参数



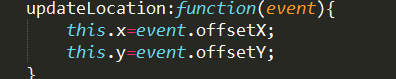


v-on=@

HTML-鼠标移动

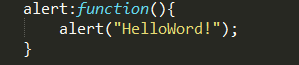


Event ：鼠标移动自带参数，可调用自带方法。

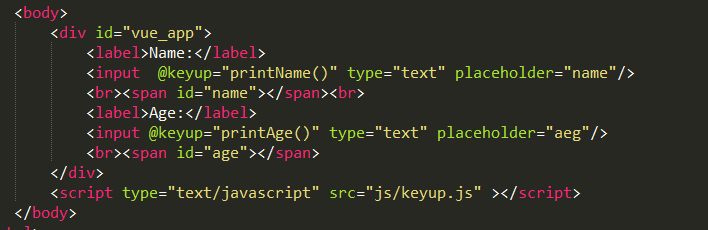


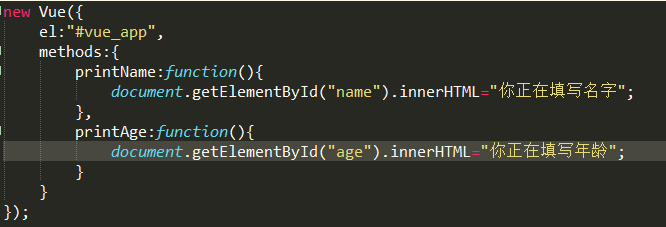
## 3.阻止事件冒泡、默认行为





## 4.键盘事件





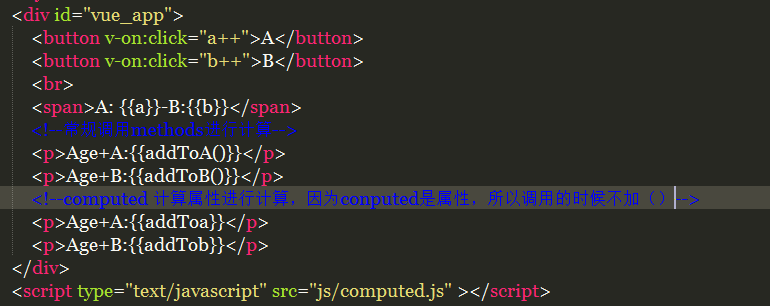
## 5.双向数据绑定input，select，textarea

推荐





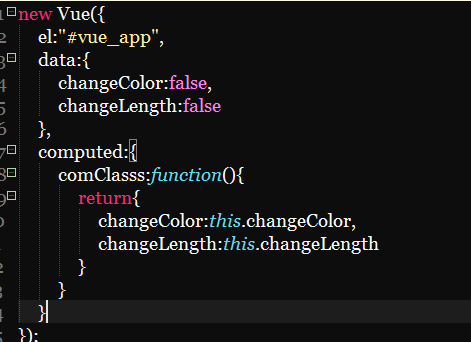
## 6.computed 计算属性





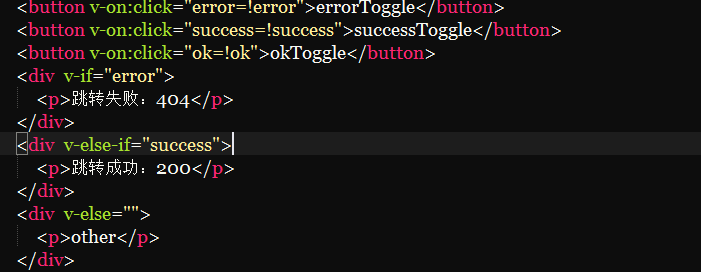
## 7.动态绑定Css属性



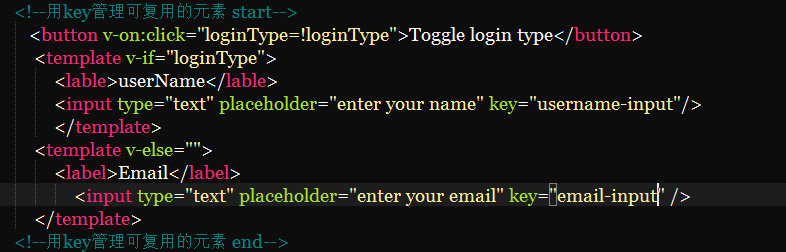


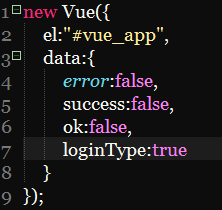
因为v-bind:class=”{}”是对象，所以需要返回对象

## 8.V\_IF指令



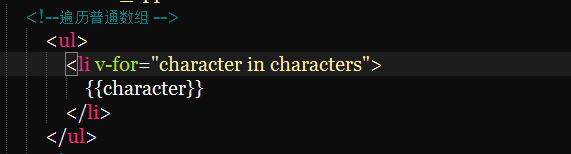




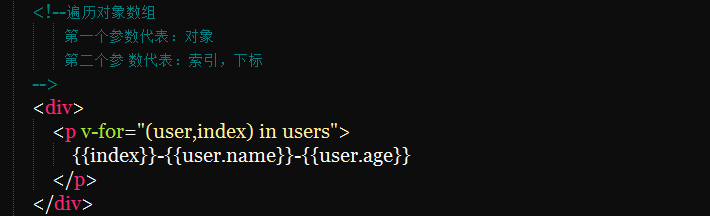


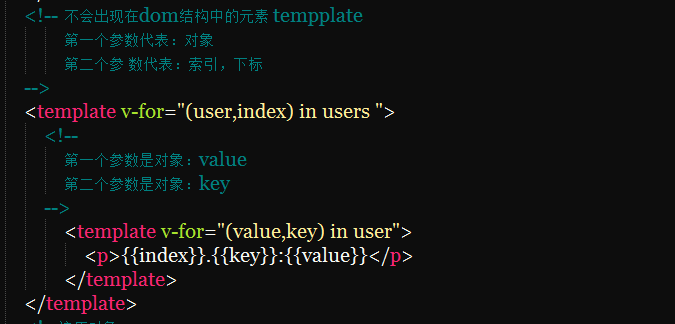
## 9.v-for指令

#### 1.遍历普通数组

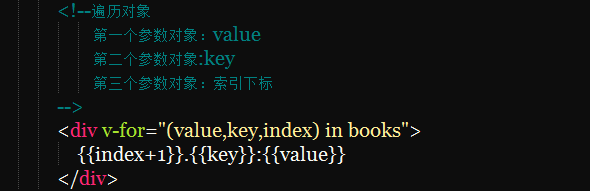


#### 2.遍历对象数组

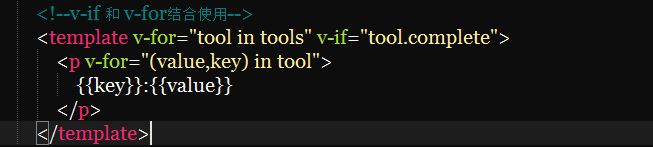


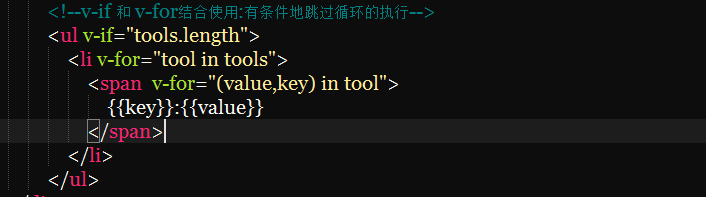


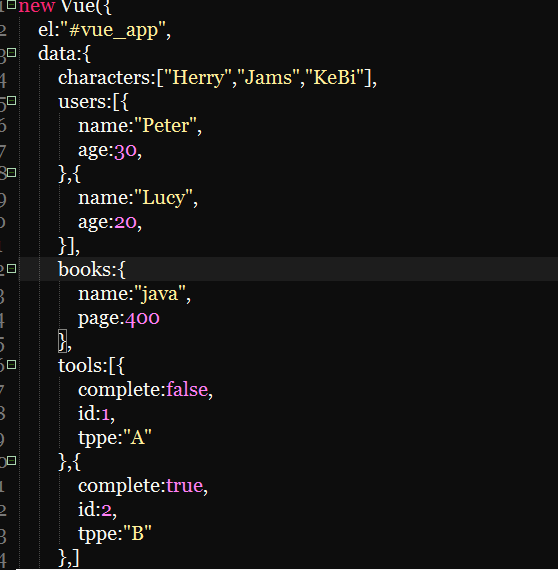
#### 3.遍历对象



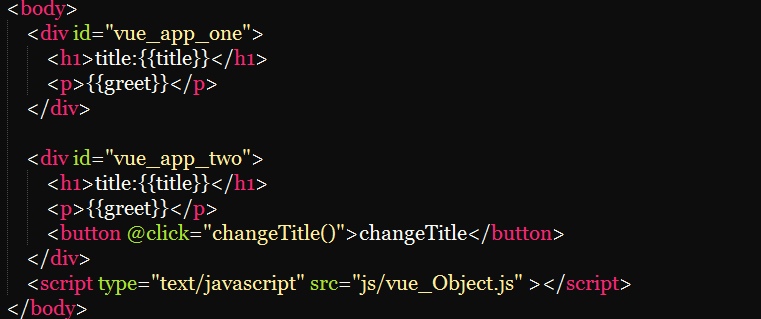
#### 4.v-if和v-for结合使用

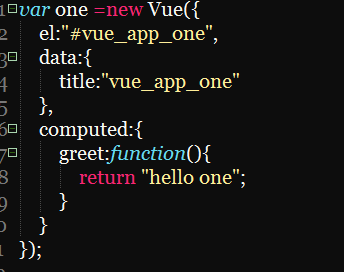




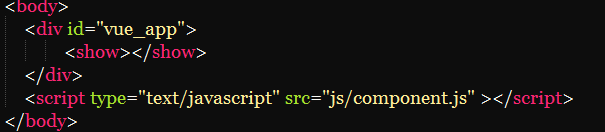


## 10.实例化多个vue对象





## 11.基础组件

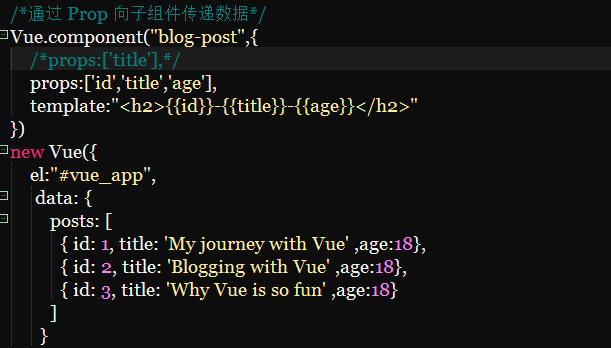




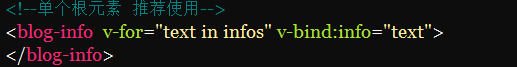
采用es6,支持换行

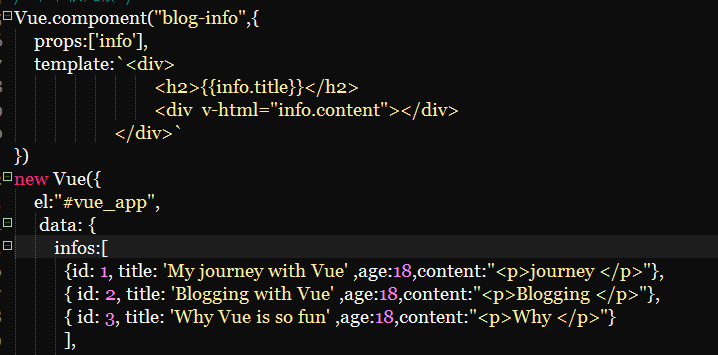
#### 通过 Prop 向子组件传递数据



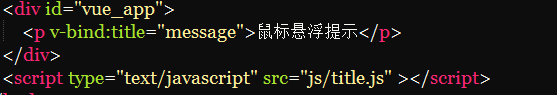


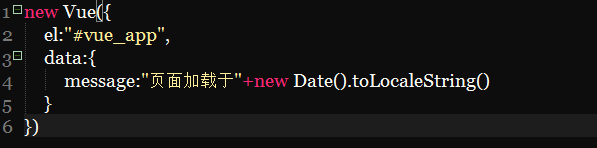
#### 单个根元素

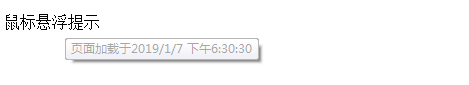




## 12.声明式渲染







## 13.splice应用

删除：

//删除起始下标为1，长度为1的一个值(len设置1，如果为0，则数组不变)

var arr = ['a','b','c','d'];

arr.splice(1,1);

console.log(arr);

//['a','c','d'];

//删除起始下标为1，长度为2的一个值(len设置2)

var arr2 = ['a','b','c','d']

arr2.splice(1,2);

console.log(arr2);

//['a','d']

替换：

//替换起始下标为1，长度为1的一个值为‘ttt’，len设置的1

var arr = ['a','b','c','d'];

arr.splice(1,1,'ttt');

console.log(arr);

//['a','ttt','c','d']

//替换起始下标为1，长度为2的两个值为‘ttt’，len设置的1

var arr2 = ['a','b','c','d'];

arr2.splice(1,2,'ttt');

console.log(arr2);

//['a','ttt','d']

添加

//在下标为1处添加一项'ttt'

var arr = ['a','b','c','d'];

arr.splice(1,0,'ttt');

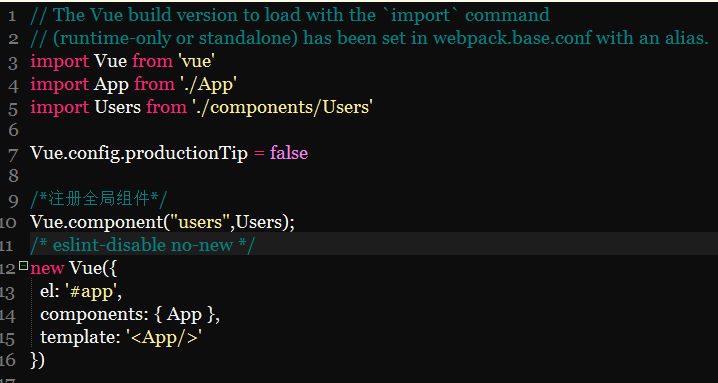
console.log(arr);

//['a','ttt','b','c','d']

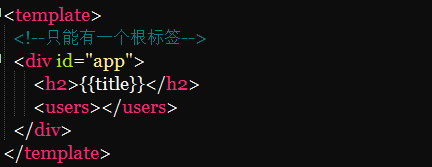
## 14.脚手架

#### 1.组件嵌套

[1].全局组件



Main.js



App.vue

使用全局组件<users>

App.vue

[2].局部组件

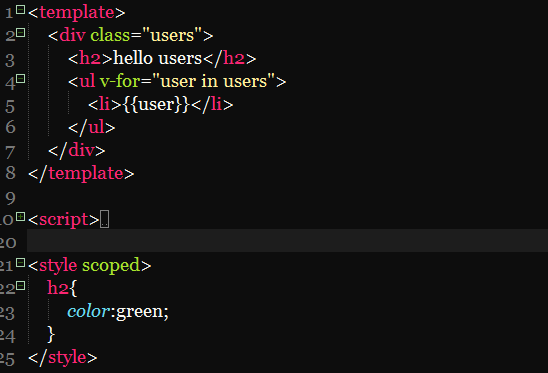


#### 2.CSS作用域



Scoped ： 样式只会在当前组件中起作用。（）

App.vue



# 19.tapestry

HTML模块（page）

页面（page）（page）

页面规范（page-specification）

规范（page specification）规范（page specification）（page）

页面（page）（page）

页面（page）

（interface）

IEngine

（interface）

IRequestCycle

Activate()

getPage()

getServiceParameters()

（interface）

IRequestCycle

指定应用程序中的一些特殊属性和方法

JAVA页面类（page class）（page）

#### Tapestry classes

getVisit() getLocal()

getGlobal() setLocal()

service()

begin() end()

beginEmpty() print()

attribute() printRaw()

（interface）

IMarkupWriter





