### @PropertySource&@ImportResource

**@PropertySource：**加载指定配置文件

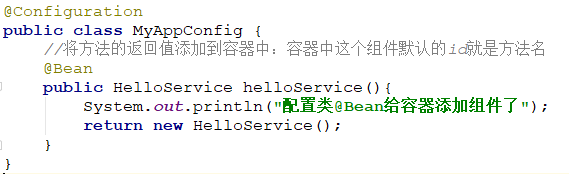
*——@ PropertySource(value = {“classpath:xxx.properties”})*

**@ImportResource：**导入自己编写的Spring的配置文件，让配置文件里面的内容生效，将@ImportResource标注在一个类上（主类上）

*——@ ImportResource(locations = {classpath:xxx.xml})*

springBoot推荐给容器中添加组件的方式：推荐使用全注解的方式。

1. 配置类======Spring配置文件
2. 使用@Bean给容器中添加组件



### Profile

1. 多Profile

我们在主配置文件编写的时候，文件名可以是 application-{profile}.properties or yml

默认使用的是application.properties

1. 以yml支持的多文档快方式

**server**:  
 **port**: 8081  
**spring**:  
 **profiles**:  
 **active**: dev  
---  
**server**:  
 **port**: 8082  
**spring**:  
 **profiles**: dev  
---  
**server**:  
 **port**: 8083  
**spring**:  
 **profiles**: prod

1. 激活指定profile
   1. 在默认配置文件中指定spring-profiles.active={profile}（properties的格式）
   2. 命令行：--spring.profile.active=dev
   3. 虚拟机参数：-Dspring.profiles.active=dev

### 配置文件加载位置:

Spring boot 启动会扫描一下位置的application.properties或者application.yml文件作为Spring boot的默认配置文件

* file:./config/
* file:./
* classpath:/config/
* classpath:/

以上是按照优先级从高到低的顺序，所有位置的文件都会被加载，优先级高的配置内容会覆盖低级优先级配置内容。配置文件会形成一个互补，高优先级没有配置的会使用低优先级中配置过的属性

我们也可以通过配置spring.config.location来改变默认配置

### 自动配置原理

配置文件能配置的属性参照:

自动配置原理：

1. springboot启动的时候加载主配置类，开启自动配置功能@EnableAutoConfiguration
2. @EnableAutoConfiguration作用：

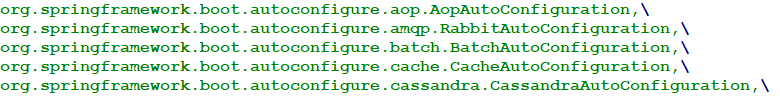
* [该注解里面有一个@Import(AutoConfigurationImportSelector.class)](mailto:该注解里面有一个@Import(AutoConfigurationImportSelector.class))，通过导入的方式，将AutoConfigurationImportSelector实例加载进IOC容器中，然后调用其selectImports方法。
* 利用AutoConfigurationImportSelector给容器中增加一些组件。
* 主要是调用了它的selectImports方法，其中有这样的一个方法：



来获取候选的配置。

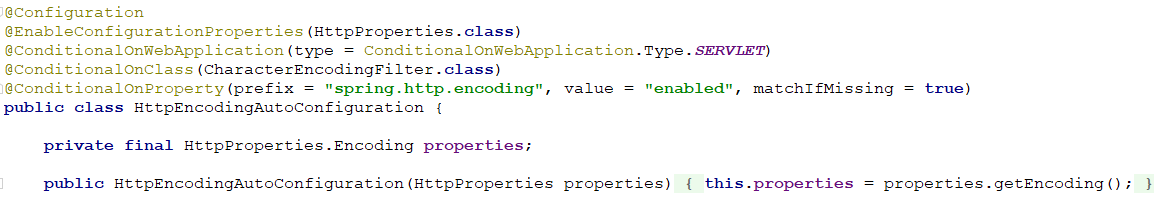
* + ，扫描所有jar包类路径下的META-INF/spring.factories，把扫描到的这些文件的内容包装成properties对象，从properties中获取到EnableAutoConfiguration.class类（类名）对应的值，然后把他们添加在容器中。

该方法的作用就是将所有META-INF/spring.factories中EnableAutoConfiguration类对应的组件加载进IOC容器中，例如：

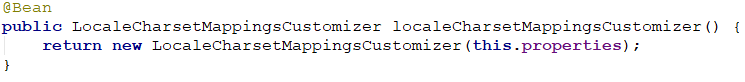


每一个这样的xxxAutoConfiguration类都是需要加载进容器的一个组件，而他们的作用是用来自动配置需要配置的组件。

1. 我们以HttpEncodingAutoConfiguration为例来进行解析。



* 1. @Configuration：表示是一个配置类，在IOC容器启动的时候会被加载到容器内，并且该类中标注了@Bean的方法返回的值会被加入容器中。



* 1. @EnableConfigurationProperties(HttpProperties.class)：HttpProperties类是一个映射类，用于将配置文件中的值（包括application.properties文件）与属性值相映射，得到一个属性对象。



但是并没有能够被IOC容器加载，所以EnableConfigurationProperties作用是让ConfigurationProperties注解生效并将配置对象加入IOC容器中。

* 1. @ConditionalOnWebApplication：用于判断是否是一个网站应用，如果是，则可以加载，否则该类的对象就不可以加载。
  2. @ConditionalOnClass：用于判断指定的类是否存在，不存在就不可以加载该类的对象。
  3. @ConditionalOnProperty：用于判断配置文件中是否有对应的属性，没有的话也没关系。

总结：

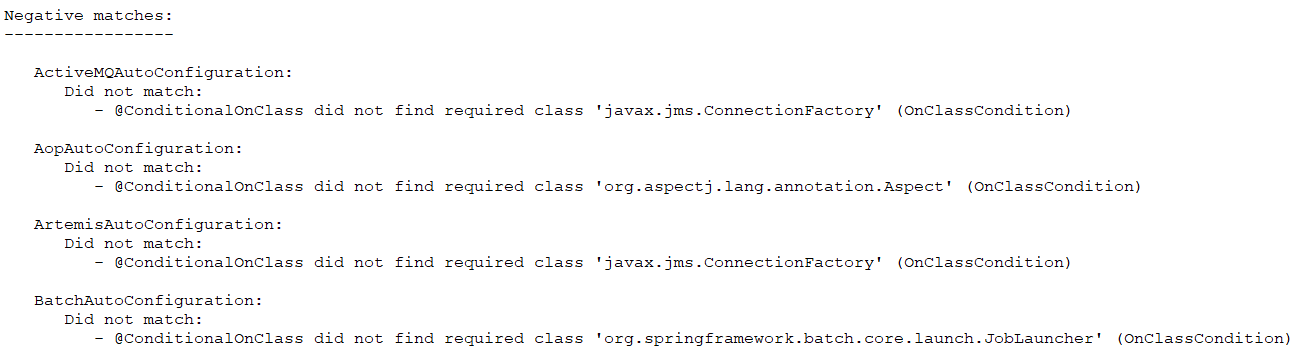
1. SpringBoot自动会加载大量的自动配置类
2. 我们看我们需要的功能有没有Springboot默认写好的自动配置类；
3. 我们再看这个自动配置类中到底配置了哪些组件；（只要我们要用的组件有，我们就不需要再来配置了）
4. 给容器中自动配置类添加组件的时候，会从properties类中获取某些属性，我们就可以在配置文件中指定这些属性的值。
5. xxxAutoConfiguration：自动配置类，给容器中添加类
6. xxxProperties：封装配置文件中相关的属性

如果想要查看哪些自动配置类装配上了，可以通过在配置文件中添加debug=true，就可以在启动容器的时候看到如下的提示：

已经配置了的：



条件不满足的无法配置的：



### 日志篇：



SpringBoot使用的是SLF4J，和Logback；SLF4J是一个接口，Logback是具体实现类

1. 如何在系统中使用SLF4J

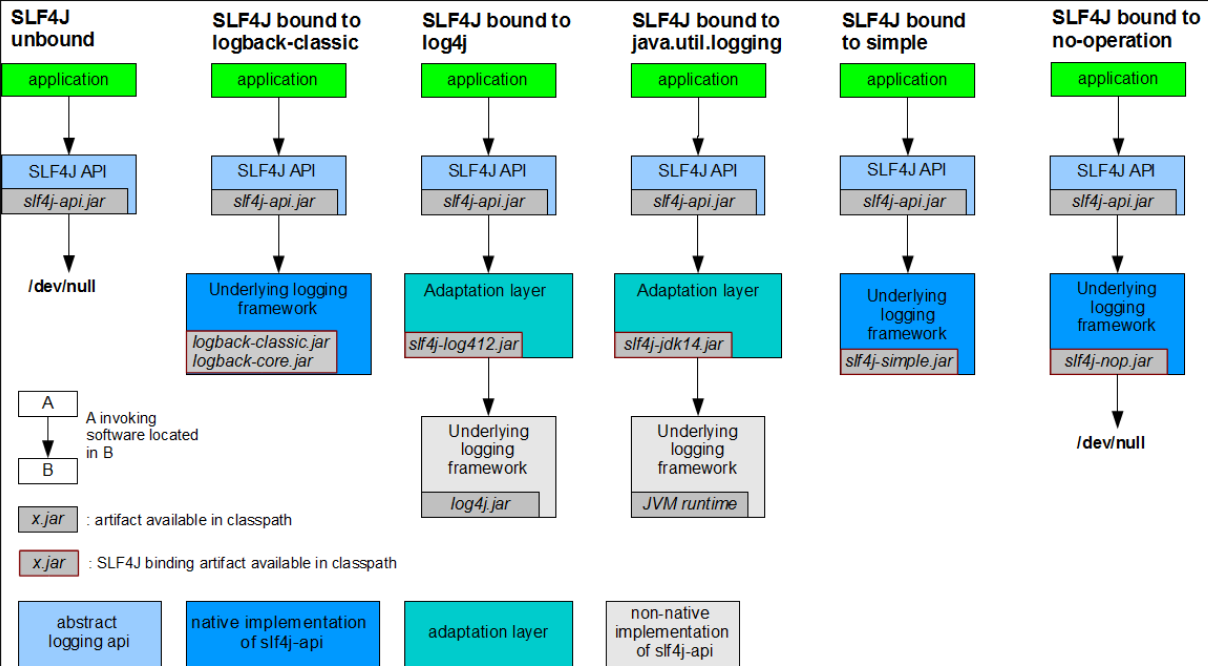
以后开发的时候，日志记录方法的调用，应该直接调用抽象层，而非具体实现类。

给系统中调用SLF4J的jar和logback的实现jar。

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
public class HelloWorld {  
  public static void main(String[] args) {  
    Logger logger = LoggerFactory.getLogger(HelloWorld.class);  
    logger.info("Hello World");  
  }  
}

每一个日志的实现都有自己的配置文件，使用slf4j以后，配置文件还是做成日志实现框架的配置文件。

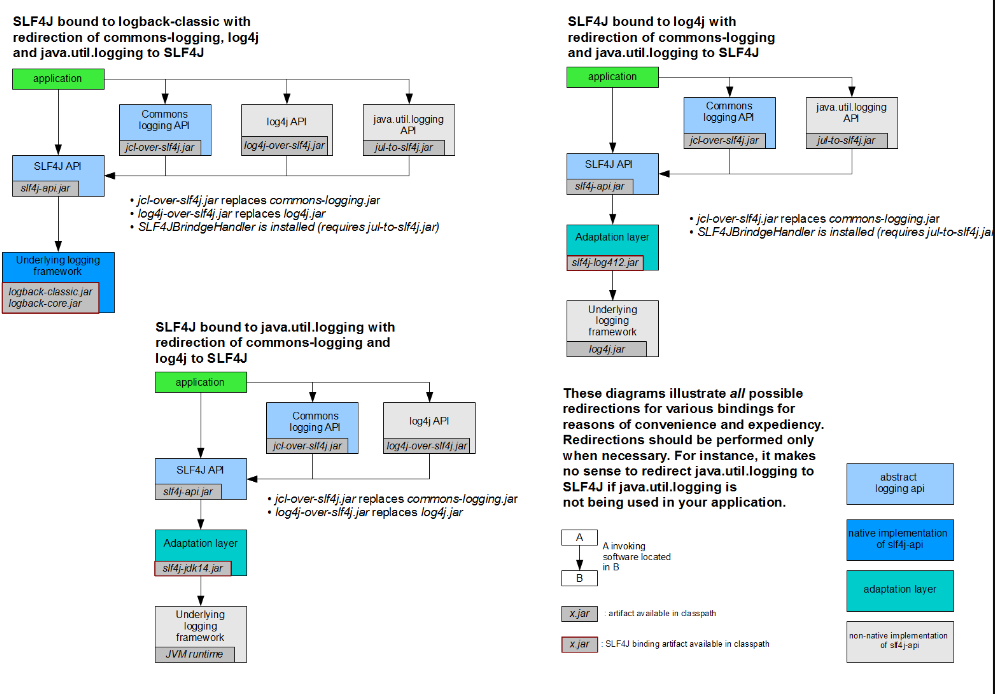
**如果底层不是使用的slf4j对应的那个实现怎么办？这个时候需要使用适配器模式，实现slf4j的所有的接口，但是调用的是原来的实现，如下图所示。**



1. 遗留问题

A(slf4j+logback):spring(commons-logging),Hibernate(jboss-logging),Mybatis,xxx

**统一日志记录，即便是别的框架和我一起统一使用slf4j进行输出。如下图所示，将原来的抽象jar包剔除掉，然后加入一个类名一模一样的jar包，调用的时候会调用slf4j的方法，这样就可以让其他框架也统一使用slf4j抽象了。**



如何让系统所有的日志都统一到slf4j？

* + 1. 将系统中其他日志框架先排除出去；
    2. 用中间包来替换原来的日志框架；
    3. 我们导入slf4其他的实现。

1. spring boot的日志

spring boot已经默认帮我们配置好日志了；

使用方法：

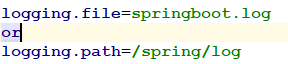
1. 创建一个logger对象：



1. Logger分为5个级别，trace，debug，info，warn，error。Springboot中默认使用的是info级别，可以在配置文件中配置logger的默认级别：



1. 可以指定日志文件：



1. 日志的profile功能
2. 日志框架的切换

### WEB

1. **使用springboot：**
2. 创建springboot应用，选中我们需要的模块
3. Springboot已经默认将这些场景配置好了，只需要在配置文件中进行少量的配置，例如Mysql地址，端口号，密码等。
4. 编写业务逻辑

自动配置原理

这个场景springboot帮我们配置了什么，能不能修改，能修改哪些配置，能不能修改等。

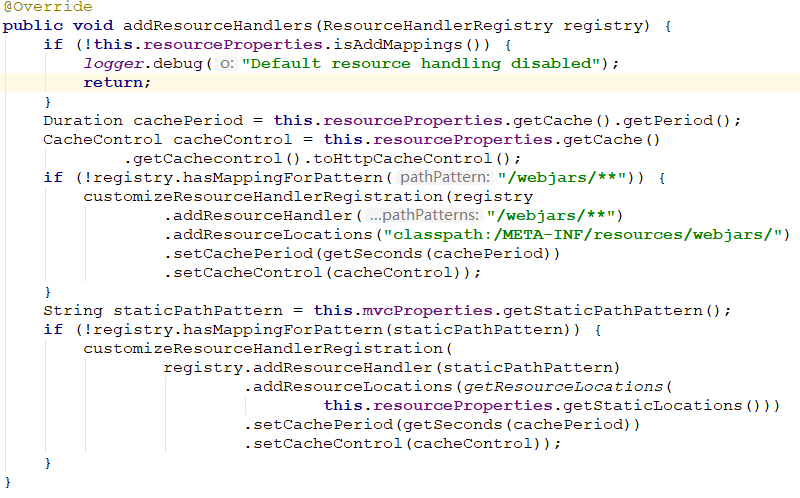
XXXAutoConfiguration：是帮我们给容器中自动配置组件

XXXProperties：配置类来封装配置文件中的类容。

1. **SpringBoot对静态资源的映射规则**

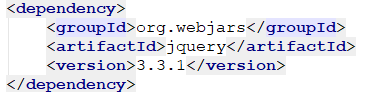


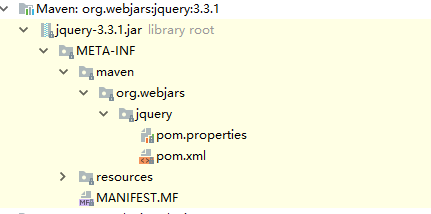
可以设置与静态资源有关的参数，比如缓存时间



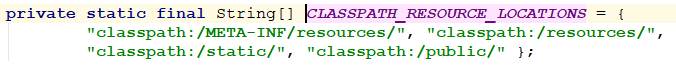
1. 所有的/webjars/\*\*，都去classpath:/META-INF/redources/webjars/下找资源

Webjars:以jar包的方式引入静态资源





1. “/\*\*”访问当前项目的任何资源，如果没有人处理，那么会到

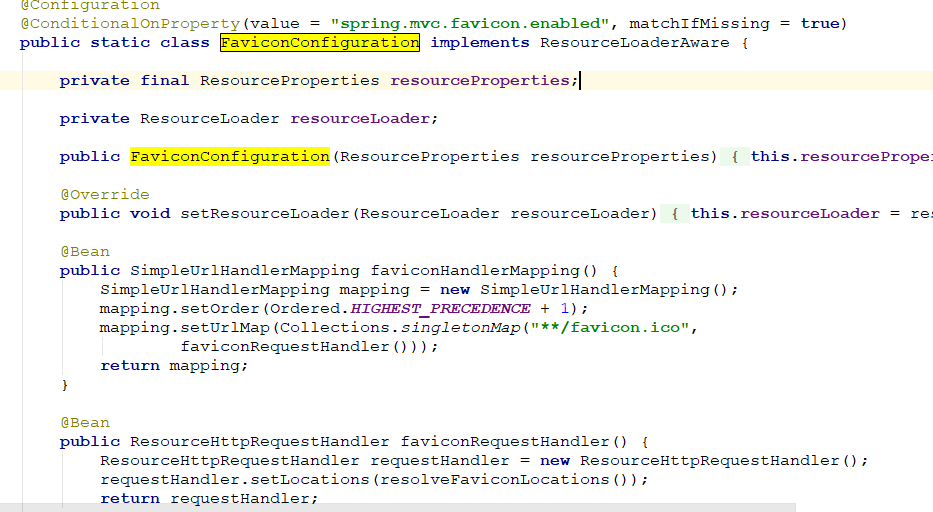


下面寻找。

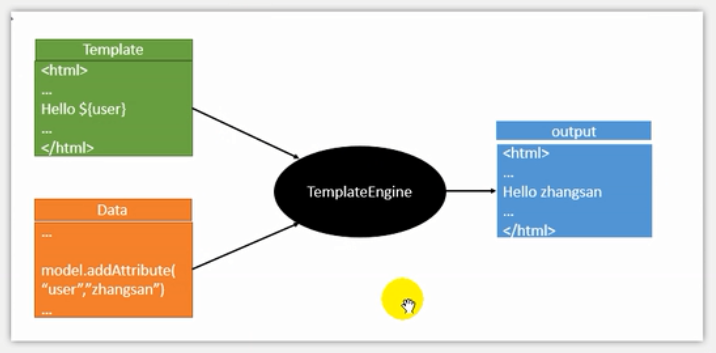
1. 欢迎页：静态资源文件下的所有的index.html页面；被“/\*\*”映射。

Localhost:8080/

1. 配置喜欢的图标，所有的\*\*/favicon.ico在静态资源文件夹上找



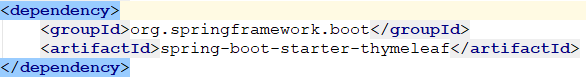
1. **模板引擎**



JSP、Velocity、Freemarker、Thymeleaf

SpringBoot推荐Thymeleaf：

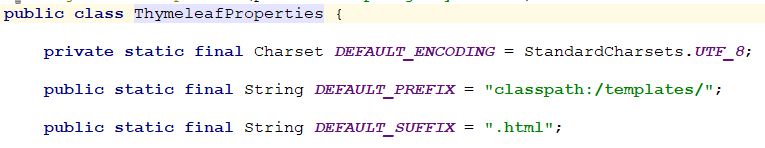
1. 引入Thymeleaf



1. 切换版本

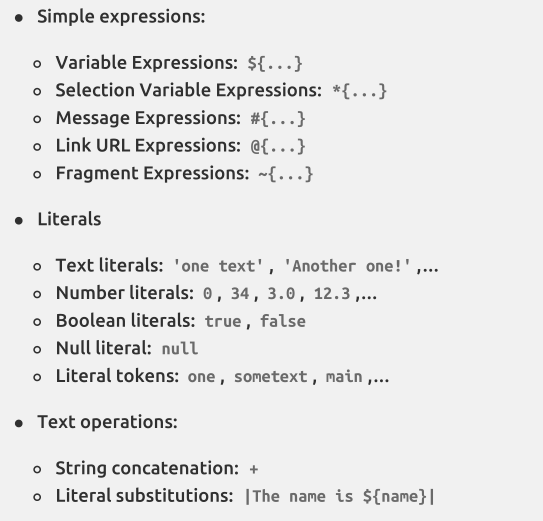


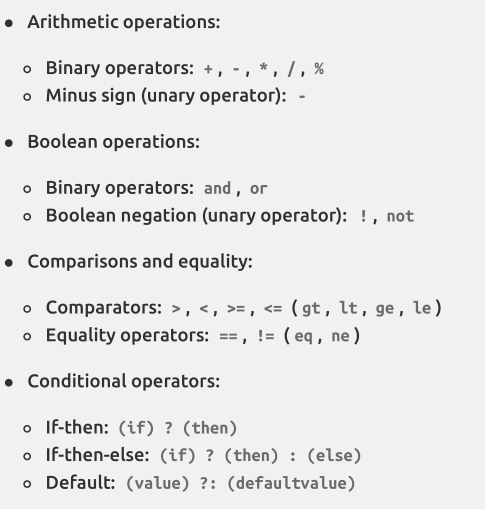
1. Thymeleaf的使用和语法



只要把html文件放在classpath:/templates/路径下，那么就thymeleaf会自动渲染

* 1. 任意的th属性可以替换原生属性
  2. 表达式语法

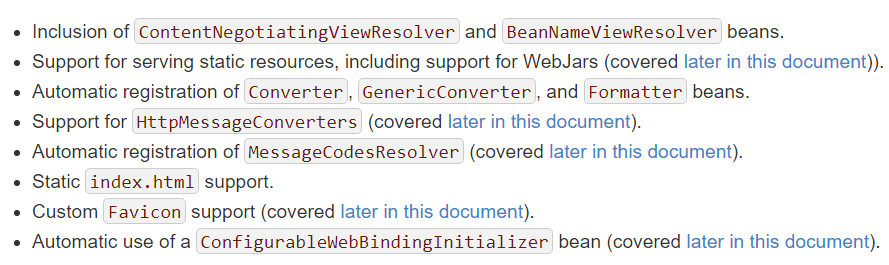


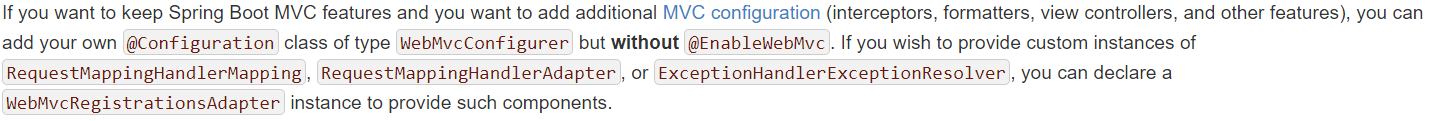




1. **SpringBoot对SpringMVC的自动配置**

以下是SpringBoot对springmvc的自动配置

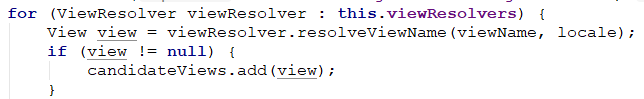




* 配置了ViewResolveer(视图解析器：根据方法的返回值得到视图对象，视图对象决定如何渲染（转发？重定向？）)
  + ContentNegotiatingViewResolver里面主要有下面两个方法：



getCandidataViews：该方法通过遍历所有的视图解析器，然后挨个解析生成视图，再将其添加到candidateViews里。



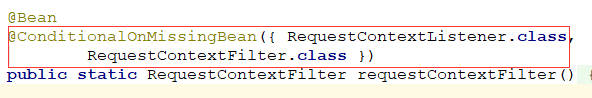
getBestView：根据要求来获取最适合的那个视图。

* + 如何定制自己的视图解析器？只需要将自定义的视图解析器放到容器中就行
* 静态资源的映射，包括Webjars的映射，首页访问的支持，自定义Favicon的支持
* 注册了Converter，GenericConverter，Formatter对象
  + Converter：转换器；public String hello(User user)：类型转换使用Converter
  + Formatter：格式化器：2017-12-17===Date;
* 支持HttpMessageConverters
  + HttpMessageConverter：SpringMVC用来转换Http请求和响应的：User——json
  + HttpMessageConverters是从容器中确定；获取所有的HttpMessageConverter，只需要将自己的组件注册到容器中。

1. **如何修改SpringBoot的默认配置**

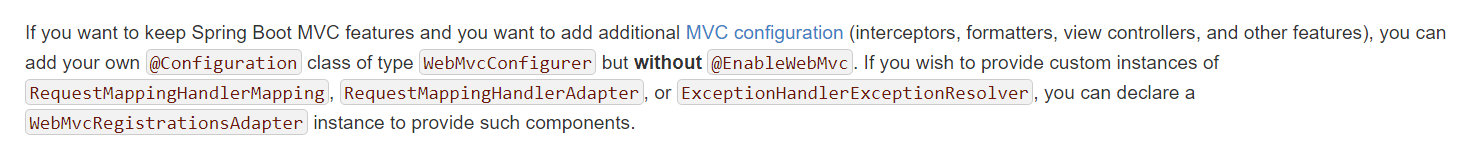
模式：

1. SpringBoot在自动配置很多组件的时候，会先看容器中有没有用户自己配置的（@Bean，@Component），如果有，就用用户配置的 如果没有才用自动配置的。



如果有些组件可以有多个（ViewResolver）将用户配置的和自己默认的组合起来；

1. 在springBoot中会有非常多的xxxConfigurer帮助我们进行扩展配置。
2. **扩展 SpringMVC**

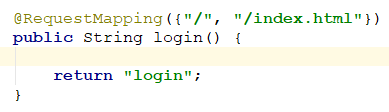


编写一个配置类（@Configuration），是WebMvcConfigurer类型；不能标注@EnableWebMvc

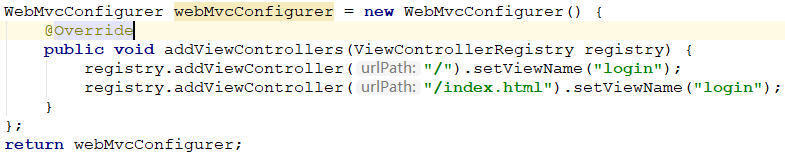


既保留了自动配置，也扩展了新的配置（SpringBoot会加载所有的WebMvcConfigurer，调用所有的WebMvcConfigurer）

1. **RestfulCRUD**
   1. **默认访问首页**
      1. 可以在controller中添加如下的方法：



* + 1. 可以返回一个WebMvcConfigurer对象，重写了addViewControllers方法，spring容器会遍历所有的WebMvcConfigurer，并且挨个调用方法。



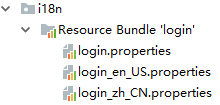
* 1. **国际化**

Springmvc步骤：

* 编写国际化配置文件：
* 使用ResourceBundleMessageSource管理股计划资源文件
* 在页面中使用fmt:message取出国际化内容

Springboot步骤

* 编写国际化配置文件，抽取页面需要显示的国际化消息



* 配置basename



其中i18n是文件夹，login是文件前缀

* 在页面中使用国际化配置文件中的值



使用点击链接的方法来切换中英文：

* 在链接后面添加参数：



* 创建一个LocaleResolver对象，重写resolveLocale方法



* 将其加入到Spring容器中

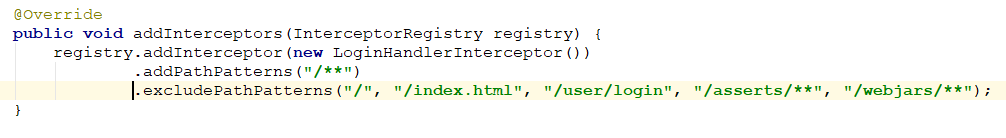


* 1. **用户页面登录**
* 提交表单，如果验证成功，那么就重定向到要到达的页面（防止表单的重复提交），否则就转发到登录也面重新登录，并显示用户或密码错误。
* 为了防止没有登录就可以访问登陆后的页面，需要使用拦截器。



拦截器先判断下该用户有没有登录，也就是查看下session中有没有用户，如果为空，那么就转发到登录页面，否则就放行。

* 将登录拦截器加入Spring容器中



在添加拦截器的时候需要指定拦截的路径，这边是拦截所有的路径，同时还需要指定不需要拦截的路径，比如访问首页的路径，登录表单提交的路径，还有静态资源的路径。这边需要特别注意的是静态资源的路径一定要配全，否则会出现死循环的情况。

* 1. **员工列表功能**

实验要求：

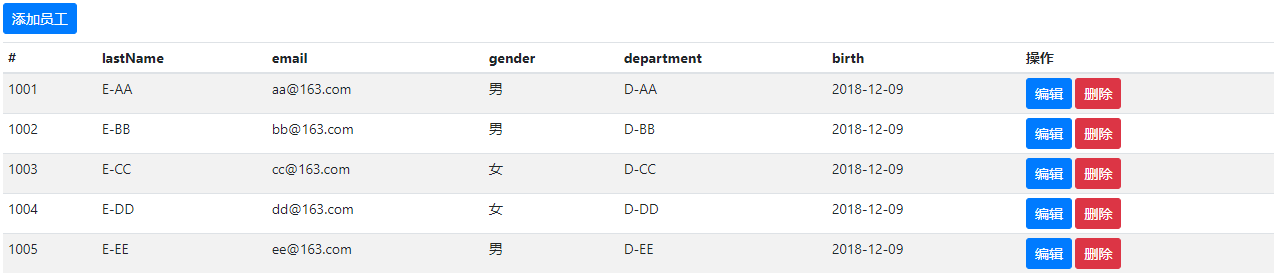




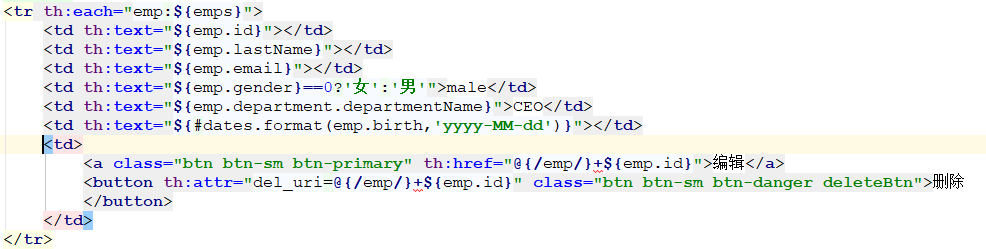
具体操作：

1. 显示员工信息：

登录后需要查看到员工的相关信息，如图所示：



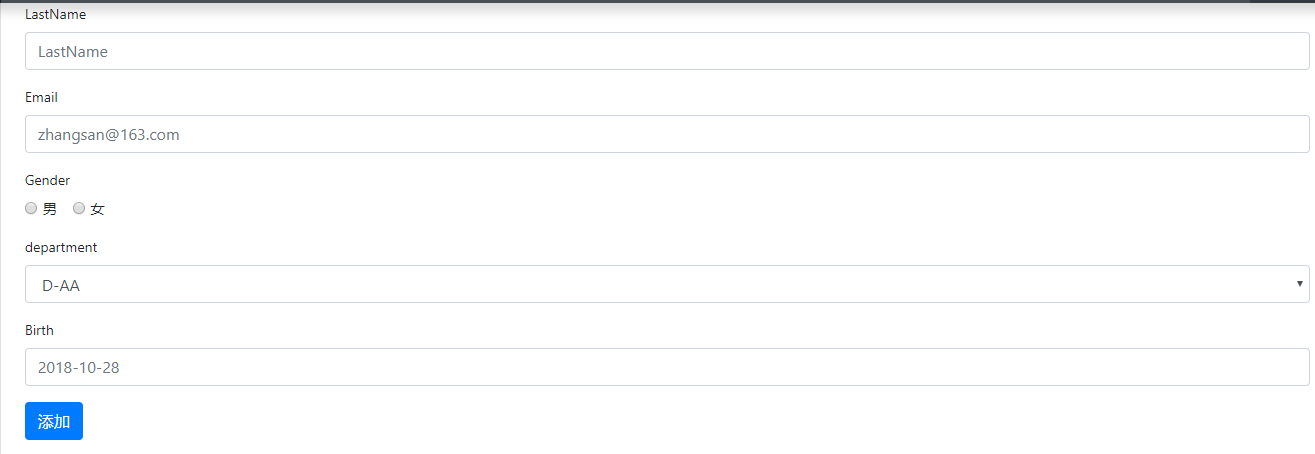
登录完成后，需要把用户的信息获取到并放入请求域中，以便转发的时候带到页面中。然后再列表页面需要使用Thymeleaf语法来进行值的显示，



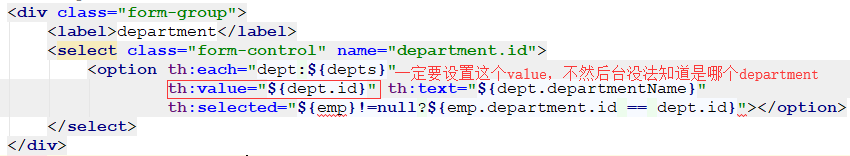
1. 添加员工：

添加员工分两步

1. 点击添加员工按钮，跳转到添加员工页面



在这一步中，由于部门不是固定的，不能写死，所以需要在controller中获取部门的相关信息，并将其放入到request请求域中，以便页面中可以使用到。例如：



1. 填写完员工信息后，点击提交，然后跳转到员工列表页面。
2. 修改员工信息

这边需要重用添加员工的页面，点击编辑按钮后需要跳转到添加员工的页面，在这之前controller中将需要回显的员工添加到请求域中（emp）。这一步很重要，emp是用来区分添加员工还是修改员工信息。

首先，对于rest风格的url来说，修改员工对应的状态是put，但是form表单里面没有put方法，只可以是post或get，所以需要通知后台在到达controller前把method的值（post）设置成想要的值（put），这边需要使用到HiddenHttpMethodFilter，与此同时，需要在form表单里添加一个隐藏域：



name必须是\_method，value是想要变成的方法，这样表单提交的时候过滤器会获取到\_method对应的value，然后将原先的method值修改成value的值。该隐藏域只有在修改员工的时候会出现，因此需要通过emp空否来判断。同理，对于回显，也需要通过emp空否来判断是否进行回显。

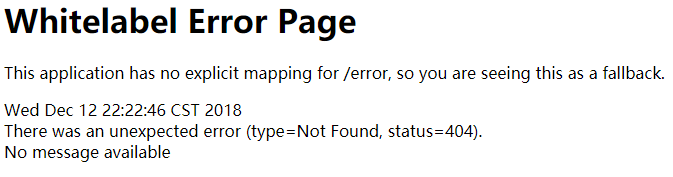
1. 删除员工

点击删除按钮，controller中接收到id后，将其删除，然后重定向到员工list的页面就可以了。这边也需要修改提交的方法，将post修改成delete，同理，添加一个隐藏域：

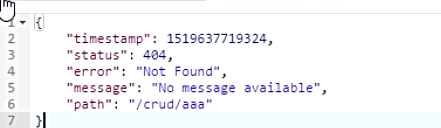


* 1. **错误处理机制**
     1. springboot默认的错误处理机制

默认效果：

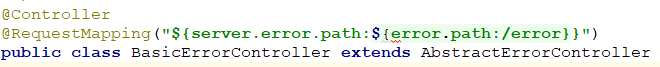


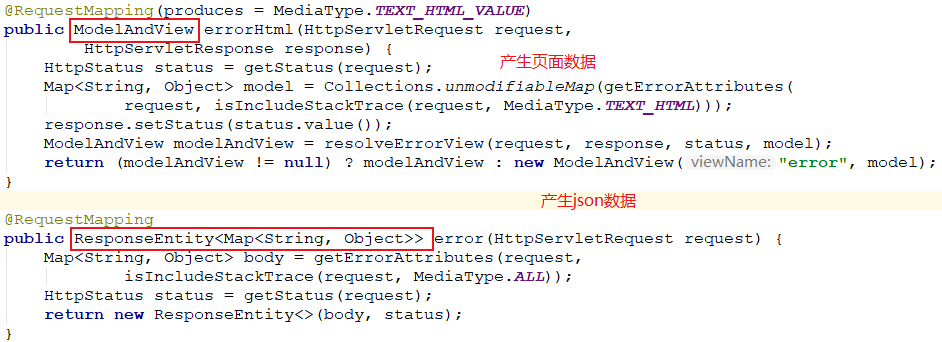
如果是其他客户端：，默认响应一个json数据



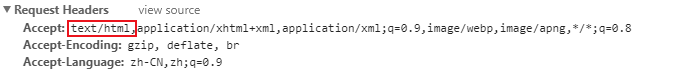
参照ErrorMvcAutoConfiguration

* + - 1. DefaultErrorAttributes:帮我们在页面共享信息，
      2. BasicErrorController:处理默认的/error请求





浏览器发送的请求头，text/html，可以根据该请求头来区别是浏览器还是其他的客户端



* + - 1. ErrorPageCustomizer:

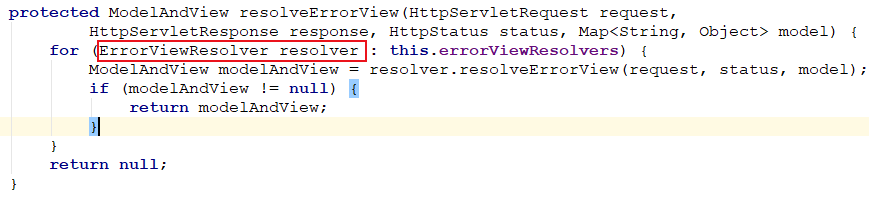
系统出现错误以后，来到error请求进行处理



* + - 1. DefaultErrorViewResolver

步骤：一旦系统出现4xx或者5xx之类的错误：ErrorPageCustomizer就会生效，（定制错误的响应规则）到/error请求；就会被BasicErrorController处理

响应页面



* + - 1. 去哪个页面是由DefaultErrorViewResolver解析
    1. 如何定制错误页面：

有模板引擎的情况下：error/404.html，将错误页面命名为错误状态码.html 放在模板引擎文件夹下的error文件夹下。

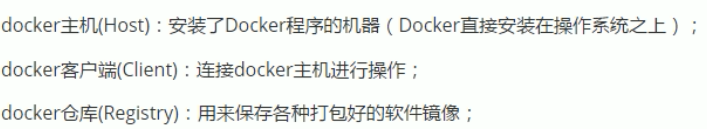
可以使用4xx和5xx作为错误页面的文件名，但是精确有限，优先寻找精确的状态码.html

错误页面可以使用的消息为：



### Docker

1. 核心概念



Docker镜像（image）:软件打包好的镜像，放在docker仓库中；

Docker容器（Container）：镜像启动后的实例我们称为一个容器；容器是一个或一组应用

使用Docker步骤：

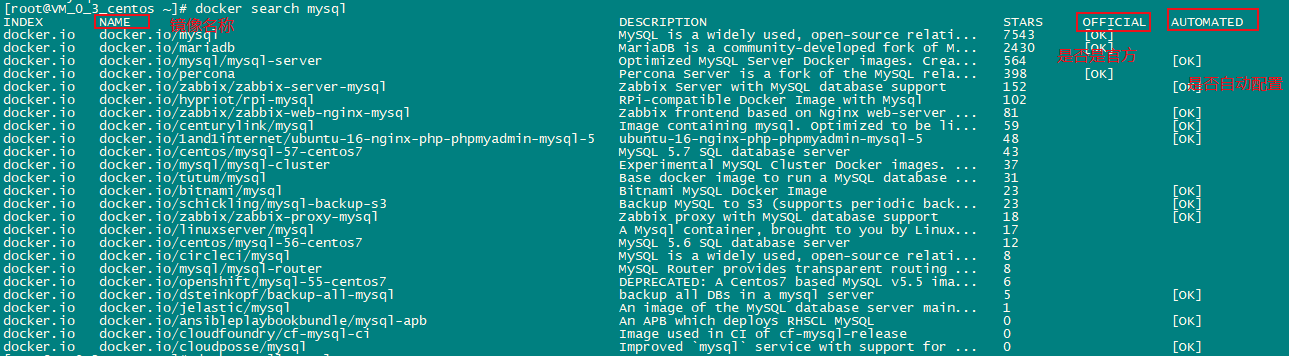
1. 安装docker
2. 去docker仓库中找到这个软件对应的镜像；
3. 使用docker运行这个镜像，这个镜像就会生成一个docker容器；
4. 对容器的启动停止就是对软件的启动停止
5. 安装Docker
6. 安装linux虚拟机
7. 在linux虚拟机上安装docker

步骤：

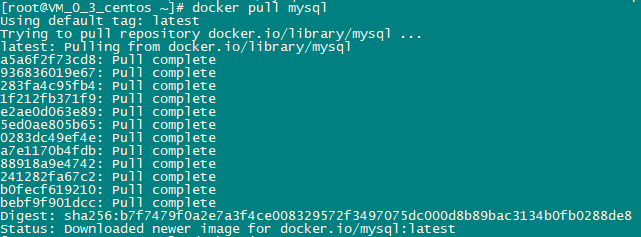
1. 检查内核版本，必须要3.10以上
2. 安装docker：yum install docker
3. 启动docker：systemctl start docker
4. 开机启动docker：systemctl enable docker



1. 镜像操作
2. 查找镜像：docker search



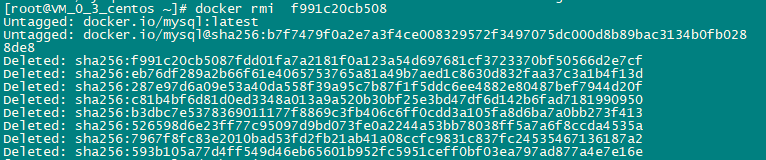
1. 下载镜像：docker pull 镜像name[:tag]，默认使用的是latest版本



1. 查看所有镜像：docker images



1. 删除镜像：docker rmi 镜像id



1. 容器操作

软件镜像（QQ.exe）——>运行镜像——>产生一个容器（正在运行的软件，运行qq）

步骤：

1. 根据镜像启动容器（第一次运行时将镜像放入容器中，相当于安装，-p port:port 可以进行端口映射，主机端口映射到容器内部端口）

docker run --name mytomcat -d tomcat:latest（--name用于重新起名字）

docker run -d -p 8888:8080 tomcat

1. 查看docker运行的进程

docker ps



docker ps -a



1. 停止运行中的容器

docker stop 容器id

1. 启动容器（非第一次）

docker start 容器id

1. 删除容器

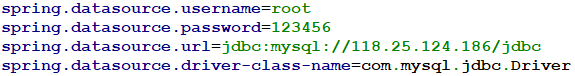
docker rm 容器id

1. 查看dockerlogs

docker logs 容器id

### 数据篇

1. 在springboot中使用jdbc，我们只需要在application.properties中做如下的配置:



就可以获取到数据源和jdbcTemplate了。

1. 自动配置解析：

**DataSourceConfiguration：**这个类的作用是用来创建Datasource对象，默认的使用是com.zaxxer.hikari.HikariDataSource，还提供了其他的一些配置，例如org.apache.tomcat.jdbc.pool.DataSource.class

org.apache.commons.dbcp2.BasicDataSource.class

除此之外，可以自动的配置，只需要在配置文件中指定spring.datasource.type，并且提供对应的类，那么springboot就可以通过反射创建datasource，并且和属性值绑定。

**DataSourceInitializer：**该类的作用是加载那些sql文件，例如创建数据库的sql或加载数据的sql，只需要将.sql文件命名为schema.sql和data.sql，这样在springboot启动的时候就会帮我们自动执行里面的sql。当然也可以通过在application.properties中配置相关的属性来设置加载的属性

**spring.datasource.username**=**root  
spring.datasource.password**=**123456  
spring.datasource.url**=**jdbc:mysql://118.25.124.186/jdbc  
spring.datasource.driver-class-name**=**com.mysql.jdbc.Driver  
spring.datasource.type**=**com.alibaba.druid.pool.DruidDataSource***# 初始化大小，最小，最大***spring.datasource.initialSize**=**5   
spring.datasource.minIdle**=**5   
spring.datasource.maxActive**=**20***# 配置获取连接等待超时的时间***spring.datasource.maxWait**=**60000***# 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒***spring.datasource.timeBetweenEvictionRunsMillis**=**60000***# 配置一个连接在池中最小生存的时间，单位是毫秒***spring.datasource.minEvictableIdleTimeMillis**=**300000***# 校验SQL，Oracle配置 spring.datasource.validationQuery=SELECT 1 FROM DUAL，如果不配validationQuery项，则下面三项配置无用***spring.datasource.validationQuery**=**SELECT 'x'   
spring.datasource.testWhileIdle**=**true   
spring.datasource.testOnBorrow**=**false   
spring.datasource.testOnReturn**=**false***# 打开PSCache，并且指定每个连接上PSCache的大小***spring.datasource.poolPreparedStatements**=**true   
spring.datasource.maxPoolPreparedStatementPerConnectionSize**=**20***# 配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙***spring.datasource.filters**=**stat***# 通过connectProperties属性来打开mergeSql功能；慢SQL记录***spring.datasource.connectionProperties**=**druid.stat.mergeSql=true;  
druid.stat.slowSqlMillis**=**5000***# 合并多个DruidDataSource的监控数据***spring.datasource.useGlobalDataSourceStat**=**true**