# 1.Springboot启动

1. 在启动springboot的时候首先加载SpringBootApplication 注解，该注解是一个复合注解，由enableAutoConfiguration 和springbootConfigure和ComponentScan 三个注解组成。
2. ComponentScan 是扫描包路径，其中最重要的注解是enableConfiguration注解 这也是一个组合注解：由Autoconfiguration注解和import注解组成，Autoconfiguration作用是扫面当前类下面的所有组件配置到容器中，import作用是将一些组件导入到容器中。其中的import注解里面主要用一个SelectImport（）方法将组件导入。该方法中生效的方法是SpringFactoryLoader.LoaderFactoryNames方法。将MATA-INFO/spring.factories 文件中的内容扫描到并且已集合形式返回，在导入到容器中。

## 自动配置分析：HttpEncodingAutoConfiguration分析

@Configuration 表示这个类是一个组件 可以被扫描进容器

@EnableConfigurationProperties({HttpProperties.class}) 指定类ConfigurationProperties 的功能，在HttpProperties类中有一个@ConfigurationProperties( prefix = "spring.http" )注解，指定配置文件读取属性绑定

@ConditionalOnWebApplication( type = Type.SERVLET ) spring底层的Conditional注解，如果满足指定的条件就配置类生效，这里判断应用是不是web应用来决定是否生效，是就生效。

@ConditionalOnClass({CharacterEncodingFilter.class}) 判断当前项目是否有这个乱码拦截器

@ConditionalOnProperty 判断配置文件中是否存在这个配置

一共五个注解

用注解根据不同条件来判断配置类是否生效

精髓：springboot启动会加载大量的自动配置类，我们需要确定我们需要的功能是否有自动配置类，有需要看看自动配置的我们是否满足要求，没有就需要自动配置，自动配置类需要在一定条件下面生效。

Conditional注解

ConditionalOnJava ：判断系统版本是否符合要求

ConditionalOnBean：判断容器存在指定的bean

ConditionalOnMissingBean：判断容器不存在指定的bean

ConditionOnExpression： 满足指定spEL表达式

ConditionalOnClass： 判断系统中有指定的类

ConditionalOnMissingClass： 判断系统中没有指定的类

ConditionalOnSingleCandidate： 容器中只有一个指定的bean

ConditionalOnProperty： 系统中属性是否有指定的值

ConditionalOnResource： 类路径下是否存在指定资源文件

ConditionalOnWebApplication： 当前应用是否为Web应用

ConditionalOnNotWebApplication：当前不是web环境

ConditionalOnJndi： JNDI指定项

# 2.Springboot日志

Springboot选择的日志框架是 SLF4j和logback；

## 1.日志使用

开发的时候调用抽象类里面的方法，就绪调用接口一样。在系统中导入这两个jar，调用方法： Logger logger=LoggerFactory.getLgger(字节码) ，写的配置文件需要按照实现的日志框架来。

## 2.统一日志框架（不同jar里面使用的日志不一样）

先去除其他日志包，导入中间包替换，然后使用指定的日志包

## 指定配置

在类路径下面放上日志的配置文件，springboot就会放弃默认配置

使用log4j.xml 或者lg4j-spring.xml ，前者不经过springboot，由日志框架自己识别，后者可以使用springboot的高级功能profile功能

# 3.Springboot实战

Springboot对静态资源映射规则

1. webjars 引入jquery jar
2. 直接访问页面会映射到静态资源文件夹路径下面（比如static包下面）
3. 访问当前项目会首先进入欢迎页，静态资源文件夹下面所有的index页面都会被映射

模板引擎（jsp，thymeleaf，freemarker）

已thymeleaf为例

我们将页面放到templates下面就能自动渲染

1.html里面打入名称空间

<html lang="en" xmlns:th="http:www.thymeleaf.org">

2.看thymeleaf语法规则使用

## Springmvc的自动配置

写一个自己的配置类继承WebMvcConfigurerAdapter 抽象类并将这个类标记为组件就可以扩展springmvc里面的功能，通过现实这个类里面的方法。

在我们的配置类中添加EnableWebMvc注解就去掉了springboot对的mvc的自动配置，需要我们自己来全部配置

## springboot错误处理机制

默认错误页面

如何定制错误页面和json数据

ErrorMvcAutoConfiguration 类 来自动配置错误处理 自动配置的bean

1.DefaultErrorAttributes：一些默认错误的属性 如时间错，状态码，错误提示，异常对象等等

2.BasicErrorController：处理默认的/error请求，浏览器发送请求返回modelandview，其他客户端返回json数据

3.ErrorPageCustomizer：系统出错来进行error请求，就回来到/error请求，然后就来到了BasicErrorController

1. DefaultErrorViewResolver：异常视图解析器 他会先看模板引擎是否存在404页面，没有就自动去寻找静态资源下的404页面

# 4.Springboot内置容器，嵌入式容器配置

1. 修改配置：修改server.xxx
2. 编写一个内置servlet容器定制器：EmbeddeServletContainerCustmizer作为组件
3. 在springboot中也会有很多xxx Custmizer的类来定制一些东西

# 5.Linux下安装docker

1. 首先linux内核版本需要高于3.10 查看内核命令 uname –r
2. 如果版本不够需要升级 命令 yum update
3. 安装docker 命令yum install docker
4. 启动docker systemctl start docker systemctl stop docker
5. 设置开机自启docker systemctl enable docker
6. Docker 命令 search + 资源名称 搜索资源 pull资源名称:tag(版本) 拉去资源 images 查看本地镜像列表 rmi +镜像id或者名称 删除本地该镜像

## Docker容器操作 菜鸟教程

1. 首先搜索（search）需要的镜像 然后拉取（pull）该镜像
2. 启动容器 docker run –name mytomcat –d –p 8080:8080 tomcat:8
3. 停止容器 docker stop+容器id或者容器名称
4. Docker rm 容器id
5. 查看防火墙状态命令 service firewalld status 关闭防火墙service firewalld stop
6. Docker log +容器id 查看容器日志
7. 更多命令<https://docs.docker.com/engine/reference/commandline/docker>
8. Mysql 启动 docker run --name ssqmysql –p 3306:3306 -e MYSQL\_ROOT\_PASSWORD:root -d mysql:tag
9. Mysql 1251错误
10. 启动进入docker里面的mysql容器（docker exec -it 容器id bash）
11. 如果要正常退出不关闭容器，请按Ctrl+P+Q进行退出容器
12. 登录mysql （mysql -uroot -p） 输入密码
13. 对远程连接进行授权：GRANT ALL ON \*.\* TO 'root'@'%';
14. 更改密码的加密规则：ALTER USER 'root'@'%' IDENTIFIED BY 'password' PASSWORD EXPIRE NEVER;
15. 更改root的密码：ALTER USER 'root'@'%' IDENTIFIED WITH mysql\_native\_password BY '123456';
16. 刷新权限：flush privileges;

# springboot原理解读

## 一．创建SpringApplication对象

1. 从执行mian方法开始，首先来new SpringApplication类，调用其中的initialize方法

2. 在initialize方法中先判断传进来的主配置类是否为空，然后将其保存sources属性中。

3.然判断当前环境是不是web应用

4.设置初始化器：从getSpringFactoriesInstances方法进去，主要在里面调用SpringFactoriesLoader.loadFactoryNames方法获取类路径下的所有META-INF/spring.factories配置文件的下面的ApplicationContextInitialize

5．初始化和设置监听器：同理去加载所有类路径下的META-INF/spring.factories配置文件的下面的Listener

6.从多个配置类中找到main方法为入口启动，到这里就创建好了SpringApplication对象

## 二．调用run方法

1. 首先需从根目录META-INF/spring.factories下面获取所有的SpringApplicationRunListeners对象，然后调用starting方法将各个监听器启动
2. 封装命令行参数
3. 准备环境：首先来获取或者创建环境，有就获取没有就创建，然后配置环境，配置完坏境就回调 监听器的环境environmentPrepared 方法表示环境准备好了。最后判断是不是web环境然后返回该坏境
4. 打印环境的图标
5. 创建ioc容器createApplicationContext 判断创建是否为webIOC容器，最后用反射创建该容器
6. prepareContext spring的准备上下文 将环境保存到上下文，这里面主要的是获取所有的ApplicationContextInitialize回调initialize方法初始化和回调所有的监听器的contextPrepared方法，最后回调监听器的contextLoaded方法配置监听器
7. refreshContext刷新容器：扫描，加载，配置IOC容器所有的组件，自动配置都是在这一步完成的。
8. afterRefresh方法；里面调用所有runner（CommandLineRunner和ApplicationRunner）的callrunner方法
9. 调用所有的监听器的finish方法
10. 这里springboot启动完成，返回IOC容器