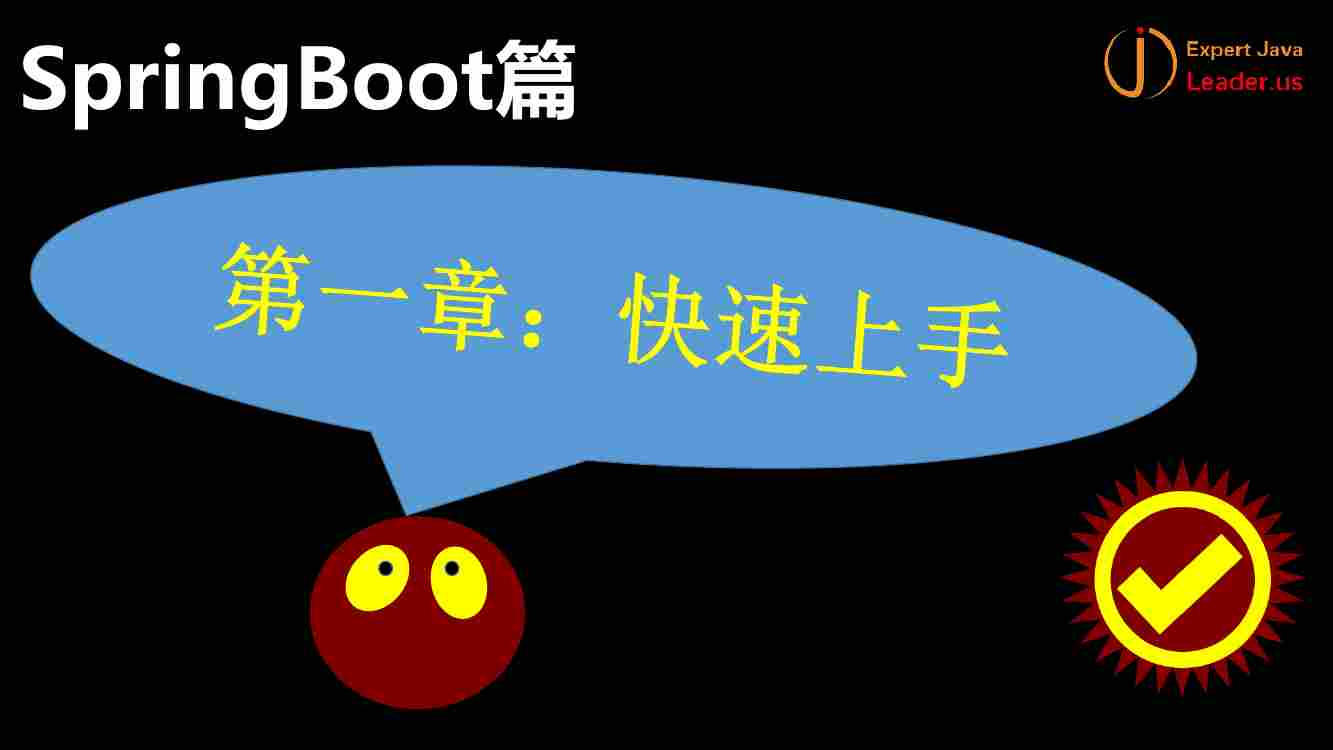
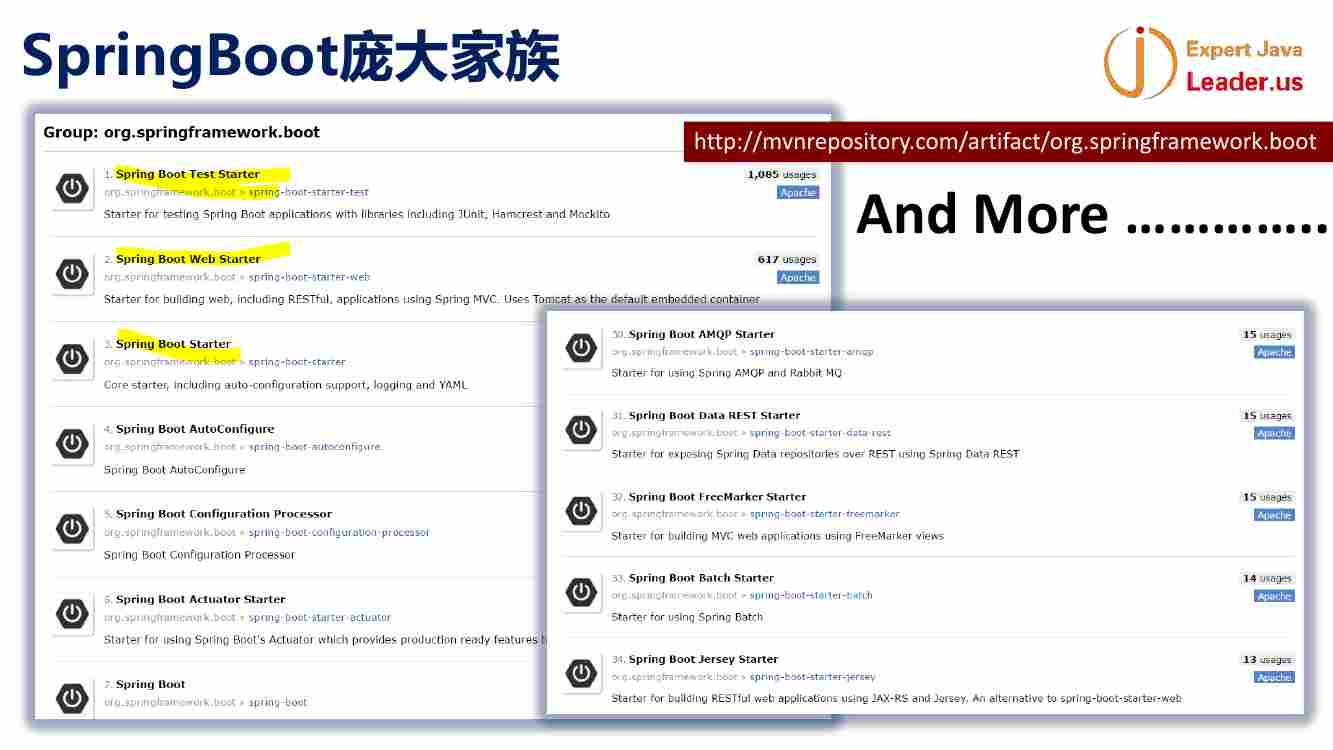
【需要自己先看看Mybatis基础那一节课 就是8.3】

SpringBoot



SpringBoot是一个Spring发展史上面**新的起点** 非常多的特性 ---- **自动化 少了很多配置**



非常多的产品

比较关键的 有三个 一个是Spring Boot Test Starter ---- Spring Boot Web Starter 和Spring Boot Starter 这三个

------ 如果开发一个web就从Spring Boot Web Starter开始

还有很多其他Spring Boot的子工程 批处理的 AMQP消息的 都是SpringBoot的子工程



Spring Boot模板maven配置也是**大量简化的** ------ 有一个spring.boot的版本配置

<properties>

<spring.boot.version>1.4.2.RELASE</spring.boot.version>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

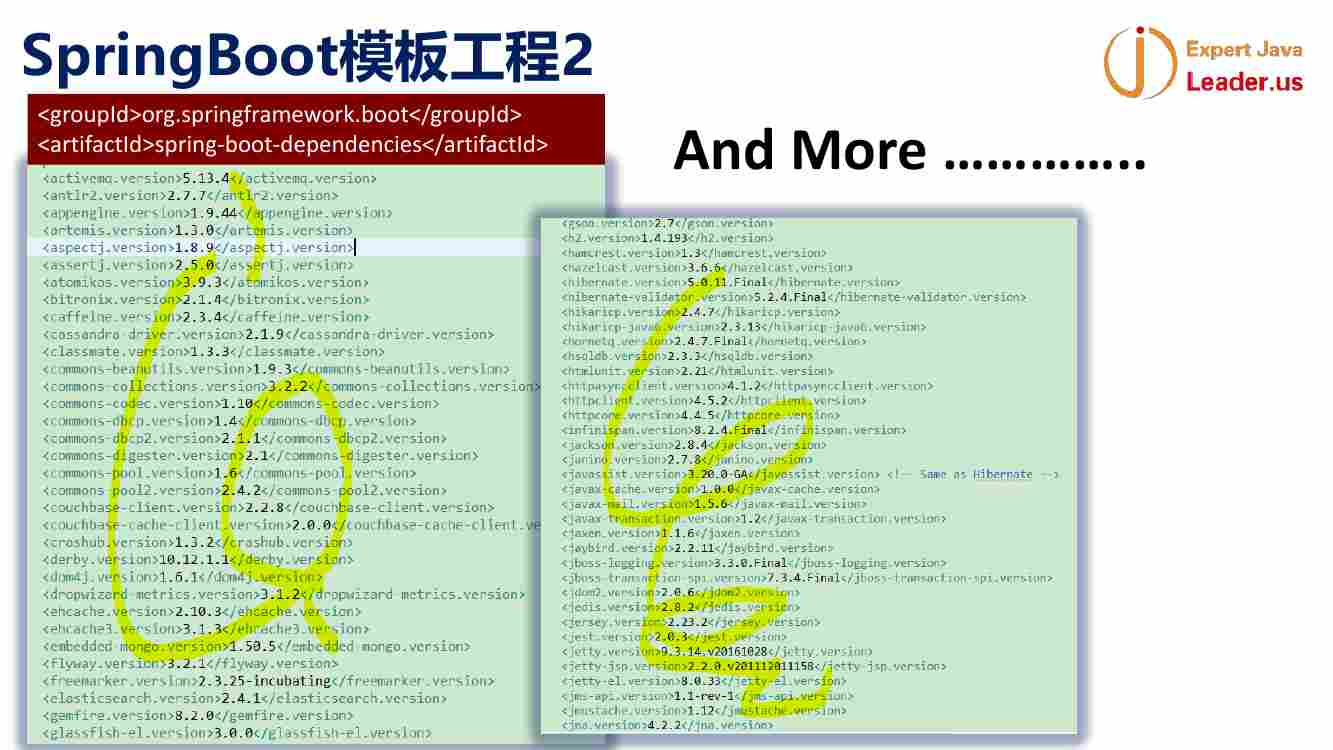
</properties>

配置完成之后 ----- 需要声明这个工程就是spring boot的 ----- 然后**依赖于springboot的子工程<parent>来进行声明** 就是 spring-boot-starter-test这个子工程（上面介绍过了）

------ **这样就能够说明这个工程是spring boot的了**

还有 ---- 依赖里面 ***如果是boot的子工程的依赖*** 或者***spring boot里面包含了*** ---- **就不用写版本号了** ---- 因为 前面已经是<parent>指定了 ----- 这个时候 你没有version的spring Boot的依赖 **你点击上去 就会出现上面那个 给出提示 告诉你版本号是多少**

这是因为spring boot本身好的去做了测试 保证对应的版本们是可用的 ---- 特殊情况 会用到某个依赖库额高版本 大多说不用修改版本



Spring boot 本身里面已经纳入了很多类库的版本 ------ 什么aspectj commons-beanutils codec javassist等等



**Spring boot模板里面 有hibernate的 有jpa的 有jdbc的 但是 没有mybatis的**

没有 为啥？ ---mybatis本身也做了一个spring Boot 前面讲了



Spring boot是以**@SpringBootApplication**注解开始的

先来分析这个注解的特性

\*\*来看 有一个**@SpringBootConfiguration** ----- 这个没什么特别的 因为这个本身就是一个Configuration 没有太大区别

------ ***所以在@SpringBootConfiguration这里面可以定义@Bean***

\*\***另外 使用@EnableAutoConfiguration开启了Spring的AutoConfiguration 这个非常重要 以后再讲**

\*\*还有 就是使用了**@ComponentScan** 默认情况 **只会加载这个注解包下面的bean**【ppt上说的 By default, @SpringBootApplication enables component scanning from the package where it’s declared ----- 就是 默认情况下 @SpringBootApplication能够对声明这个注解所在的包下面的类进行**组件扫描** --- 这个意思就是 只要使用@SpringBootApplication这个注解标记之后 这个类下面的包的bean就被自动进行组件扫描了】

**所以 要扫描其他的包 就需要添加scanBasePackage 为这个属性添加其他的包名**

---- [例子就是@SpringBootApplication(scanBasePackage=”edu.idcollege.\*\*”)](mailto:例子就是@SpringBootApplication(scanBasePackage=)

另外 ComponentScan的时候 可以使用@Filter来排除特定功能的bean 这个功能非常强大 ---- 一般用的不多 --- 但是可以扩展【可以把ppt上面的小文字读一遍-----!!!!!】

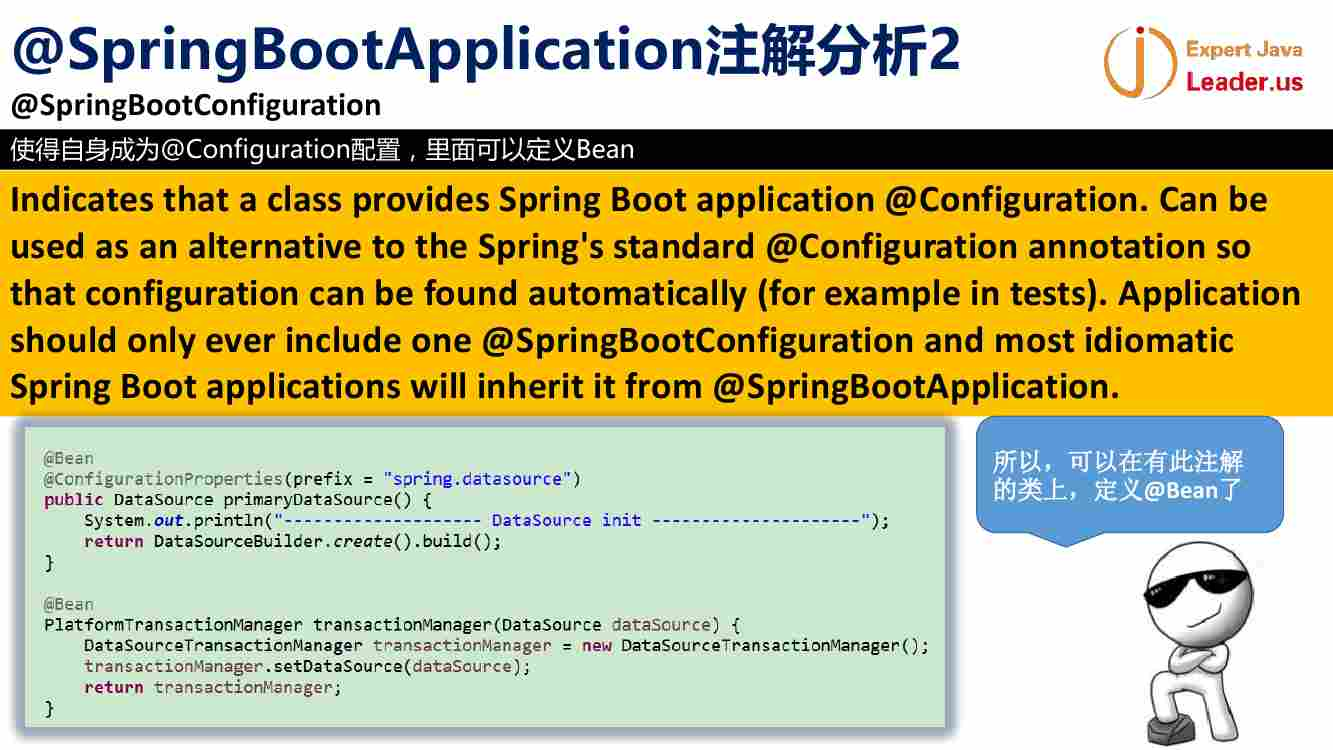
**@Filter ------ @ComponentScan的附属品 【@Filter进行扩展】**

---- 关于这个Filter 例子就是 @Filter(type=FilterType.CUSTOM, classes=TypeExcludeFilter.class) ----- 这样标记之后 配置到@ComponentScan中的excludeFilters=@Filter(type=FilterType.CUSTOM, classes=TypeExcludeFilter.class) ------ 这样 这个类型的实例对像就都不会被扫描出来 -----

【所以 重温一下 就是@SpringBootApplication这个注解就是三大注解构成：

**@SpringBootConfiguration + @ComponentScan(@Filter) + @EnableAutoConfiguration**】

【注解的内容要复习温习一下】



上一页PPT说了 @SpringBootApplication里面有@SpringBootConfiguration 说明**SpringBoot本身就是一个Configuration的注解** ---- 定义新的注解的目标是**标记了SpringBoot的Class本身就是Configuration**

可以定义@Bean 被引用 可以方便测试 【PPT英文】

*-----* ***需要注意*** *每一个SpringBoot的应用只有一个****@SpringBootConfiguration一个注解*** *==== 并且这个注解继承自@SpringBootApplication*

【这里面的是@SpringBootApplication这个注解定义中的注解 @SpringBootConfiguration ---- 是的自身成为@Configuration配置 里面可以额定@Bean ----- 这个@SpringBootConfiguration表明一个类提供了Spring Boot 应用 @Configuration ---- 可以用作对Spring标准@Configuration的一个替换 这样 就有配置就会被自动的找到（在测试中的例子可以看到） ----- 应用应该仅仅包含一个@SpringBootConfiguration 并且多数的常用的SpringBoot的应用都可以从@SpringBootApplication这个注解里面继承到@SpringBootConfiguration这个注解】

【这个注解要研究一下】



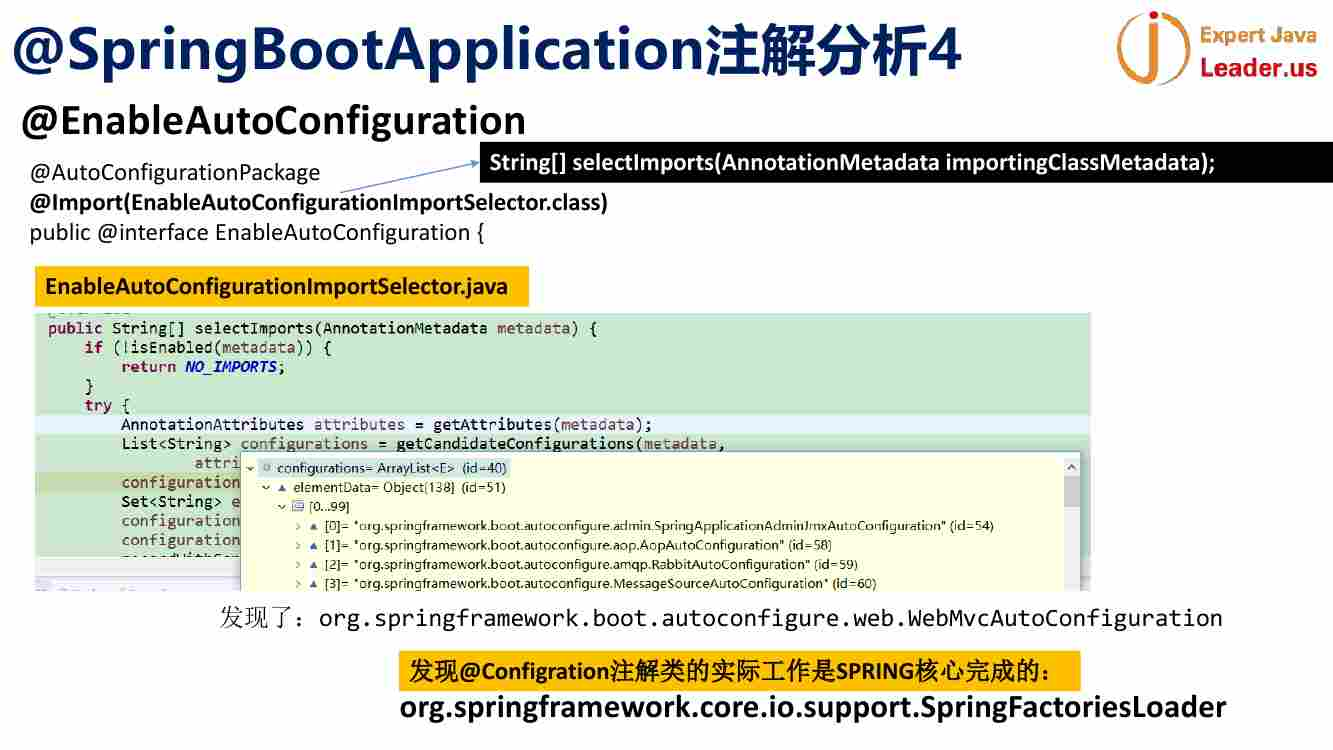
自动配置@EnableAutoConfiguration是SpringBoot的核心，按照这上面的解释 是最智能的一个注解 ---- 如果我的程序包中包含了某个class 也就是我把某个程序包包含进来了 ---- **那么Spring就会猜测你可能用到这个jar中的某些class ---- 对应的spring中就是用到这个jar中的某些bean** --- 于是这个注解就会定义一个Configuration 配置 把这个jar中所有的class都定义成bean ---定义好 纳进来 --- 也就是不用定义这些Bean 也不用写这些配置文件 那么SpringBoot会把这些jar中的bean自动加进来 ------ 就是这个意思  **好处就是做了很多现成的模块可以使用**

@EnableAutoConfiguration的机制：上面的类图 【理解Spring的扩展机制】

**ConfigurationClass有一个PostProcessor --- 实现的是BeanFactoryPostProcessor ----- 可以处理注册表** --- 也就是在Bean的注册表里面增加一些Bean ----- 那么BeanDefinitionRegistryPostProcessor就扩展了其中的postProcessBeanDefinitionRegistry这个方法 ----- 注册表传进来 可以对里面的注册表本身进行操作 ----- @Configuration可以import其他的@Configuration ---- 因此 通过 这个BeanDefinitionRegistryPostProcessor就可以把import的那些Configuration引进来 ----- 就是定义 ***【那么这个@EnableAutoConfiguration是怎么和ConfigurationClassPostProcessor进行关联的呢？ 老师的PPT的源码调用栈也没看出来 所以 这一块看看@EnableAutoConfiguration和这个类怎么关联的 然后 看看源码怎么处理的import Configuration的内容的】***

***【但是 还是看到了EnableAutoConfigurationImportSelector.selectImports(xxxxx)】***

【所以 可以看到 调用栈是ConfigurationClassPostProcessor.postProcessBeanDefinitionRegistry -> ConfigurationClassPostProcessor.processConfigBeanDefinitions -> … -> EnableAutoConfigurationImportSelector.selectImports(AnnotationMetadata) -------- 说明在修改BeanDefinitionRegistry的时候 就调用了EnableAutoConfigurationSelector的selectImports这个方法 方法就是下页PPT --- 可以分析源码 ---- 也就是要看下这个@EnableAutoConfiguration这个注解是怎么定义的 一看下一节PPT的注解的定义 就出现了这个EnableAutoConfigurationImportSelector的class】



@EnableAutoConfiguration这个注解定义的时候 继承了一个注解 是 **@Import(EnableAutoConfigurationImportSelector.class)**

那么这个EnableAutoConfigurationImportSelector决定了引入哪些Configuration ----- 注意 这个EnableAutoConfigurationImportSelector.class这个类里面有一个方法 String[] selectImports(AnnotationMetadata importingClassMetadata); 这个方法 方法就是PPT下面显示的方法

这里面有一个getAttributes(metadata)方法被调用 ---- 返回一个AnnotationAttributes（复数） ---- debug的时候 可以看到 后面这个**configurations**变量有100多个（默认情况） ------ 发现了spring mvc的Configuration也进来了 ---- 就是org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration这个类 ---- 默认自动加载SpringMVC的Configuration ----- 可以看到 这个configurations下面所有的类都是 org.springframework.boot.autoconfigure下面的XXXxConfiguration类 ---- 因此 就应该去这个包下面分析 对应的这些文件 ---- 可以发现是标准的@Configuration标记的类【源码分析】

这些对应的Configuration是怎么被发现的呢？ 是Spring的core 也就是Spring的核心 SpringFactoriesLoader发现的 -----发现细节是下页PPT



现在分析SpringFactoriesLoader这个类如何发现对应的Configurations ---- SpringFactoriesLoader这个类加载并实例化了来自META-INF/spring.factories这些文件的一个指定的类 ------ 这些文件可能在类路径中存在于多个jar ---- 也就是扫描每一个jar下面的这个文件 如果有这个文件 就把对应的内容扫描出来 ---- 这个spring.factories文件应该是一个properties格式的文件 ---- 这个properties文件中的key是接口或者抽象类的全类名 并且 value就是逗号分隔的实现类类名的列表list。 比如

example.MyService=example.MyServiceImpl1,example.MyServiceImpl2

等号左边是接口 右边是两个实现类

看图中的源码 FACTORY\_RESOURCE\_LOCATION就是字符串”META\_INF/spring.factories”

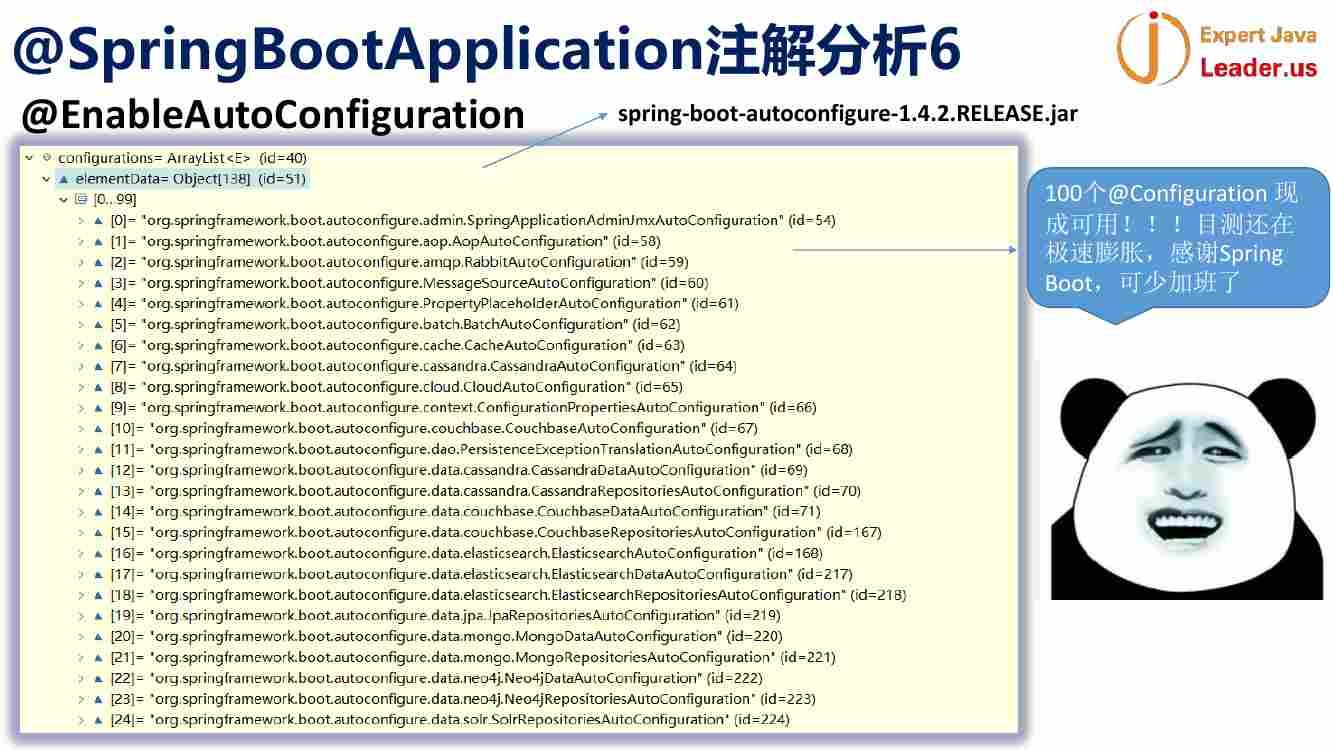
【Enumeration<URI> uris

= (classLoader != null? classLoader.getResources(FACTORY\_RESOURCE\_LOCATION):ClassLoader.getSystemResources(FACTORY\_RESOURCE\_LOCATION));

这个三目运算就获得了所有的jar下面的这个文件的uris ---- 每一个进行解析 解析完成 把factoryClassNames放到最终的reuslt list里面 ----- 因为这个类就叫SpringFactoriesLoader 所以 这样 这个Spring的工厂加载类就应该把这些工厂名都加载到一个list里面 并返回 List<String>】

SpringBoot通过SpringFactoriesLoader的配合就得到了jar中这些文件的信息 ---- 那么这些文件到底是什么呢？

【需要源码跟进才行】



刚才说的100多个Configuration就是在spring-boot-autoconfigure-1.4.2.RELEASE.jar这个jar里面加载出来的

各种常见的配置都有了

打开这个jar中的META-INF/spring.factories文件 看下下页PPT



这样 在这个properties中 加载的是#Auto Configure下面的这些内容 ---- 各种各样的Configuration都有 【唯一不知道的就是SpringFactoriesLoader和Spring声明周期如何连接到一起 源码跟进】

\表示这一行没结束 ----= \所以行是接口 下一行是实现类 (,\截止) -----一整行

全部加载完了 建议打开这个文件的Configuration配置文件看看 ---基本我们常用的模块都涉及到了 比如redis 可以看到是使用spring.data.redis ---- Spring Security中涉及到redis ---redis也给你配置好了

还看到了autoconfigure.h2.H2ConsoleAutoConfiguration也看到了

**还有就是 SpringMVC ---WebMvcAutoConfiguration 和WebSocket都配置好了**



可以明白 Springboot使用了嵌入的tomcat ---- 举例就是 如果你在类路径中使用了tomcat-embedded.jar 那么你就很可能想使用TomcatEmbeddedServletContainerFactory这个类 ---- 除非你已经定义了自己的EmbeddedServletContainerFactory bean

所以 打开Spring boot的pom文件 可以看到 关于嵌入的tomcat相关的jar有 tomcat-embed-core, tomcat-embed-el, tomcat-embed-jasper, tomcat-embed-websocket 这么多和嵌入tomcat相关的jar ------ 然后来看tomcat相关的spring boot的配置文件

----- 看到 有一个ConditionalOnWebApplication 【学习这个注解】---- 表示 当前Spring环境是WebApplication的话 这个EmbeddedServletContainerAutoConfiguration就会被加载 看一下 这个内部静态类 public static class EmbeddedTomcat 这个类 下面有三个注解进行修饰：

@Configuration

@ConditionalOnClass({Servlet.class, Tomcat.class})

@ConditionalOnMissingClass(value=EmbeddedServletContainerFactory.class, search = SearchS…..)

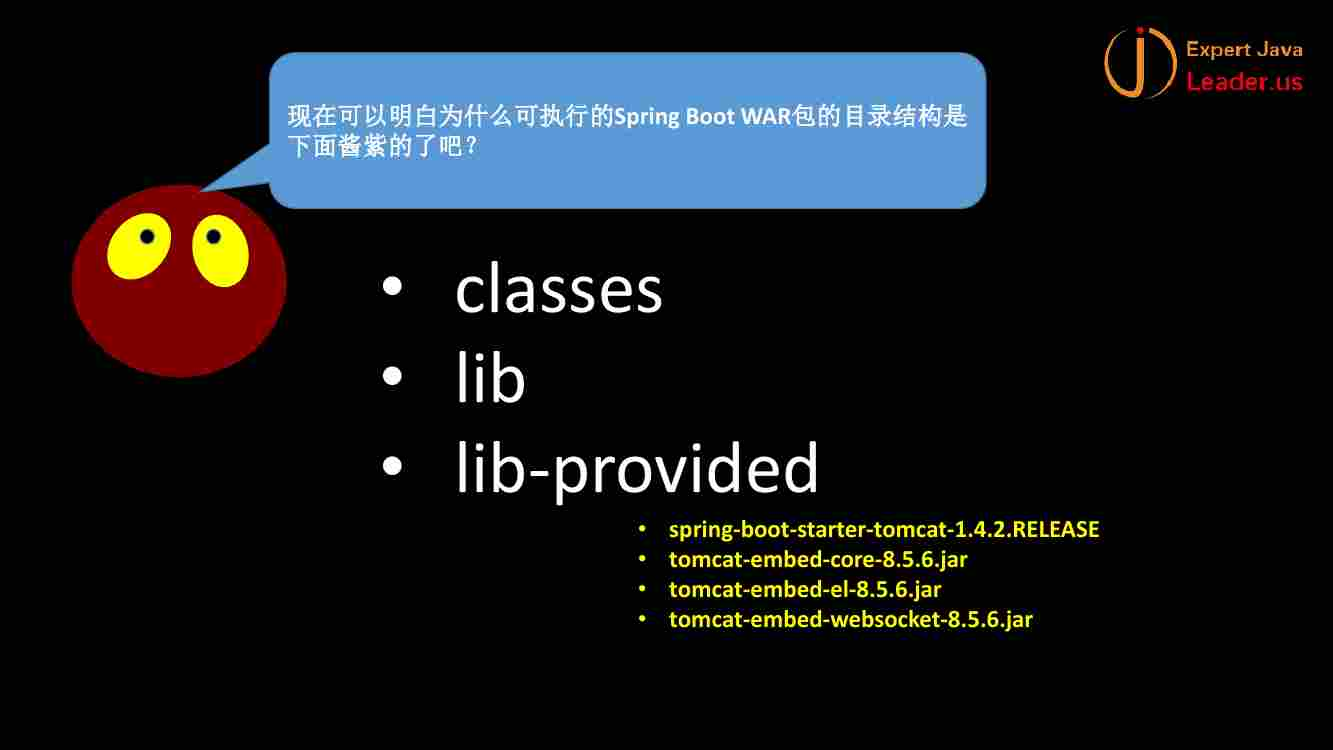
含义就是 如果你这里面已经有Servlet和Tomcat这两个class 并且还没有EmbeddedServletContainerFactory这个类的时候 那么下面的@Bean就会创建 EmbeddedServletContainerFactory这个类

【不同的注解值得好好学习】

SpringBoot为什么能完成这些 就是因为SpringBoot已经把这些Configuration做好了 就是根据不同的条件进行触发 ---- 只是工作没有减轻 Spring做了

所以SpringBoot之后 你的配置更少了 --- 代码都是标准化的 都是优化的

所需要的就是 对这些进行定制 并且这里面的定制可以用来进行参考学习



为什么Spring Boot WAR的目录 ---- Spring Boot WAR 可以放到tomcat中使用 也可以独立的执行 ---- 这种情况下 可以发现tomcat的这几个jar是在lib-provided下面 而不是在lib下面 ----

这样的话 如果单独执行的话 lib和lib-provided下面的jar都会加载 变成了嵌入的tomcat 独立运行

如果放到tomcat web环境中运行 那么智能加载lib下面的jar 剩下的lib-provided下面的tomcat相关的jar不能被加载进去 --- 这是因为tomcat相关的jar tomcat这个容器本身会提供

【这个就和maven中 为了不让tomcat相关的类报错 引入 servletx等jar 但是 scope是provided 这个就是Spring boot借鉴的地方 ---个人猜测】

【所以 有了这样的分层 就表示这个war既可以独立运行 又可以放到tomcat容器中进行部署】

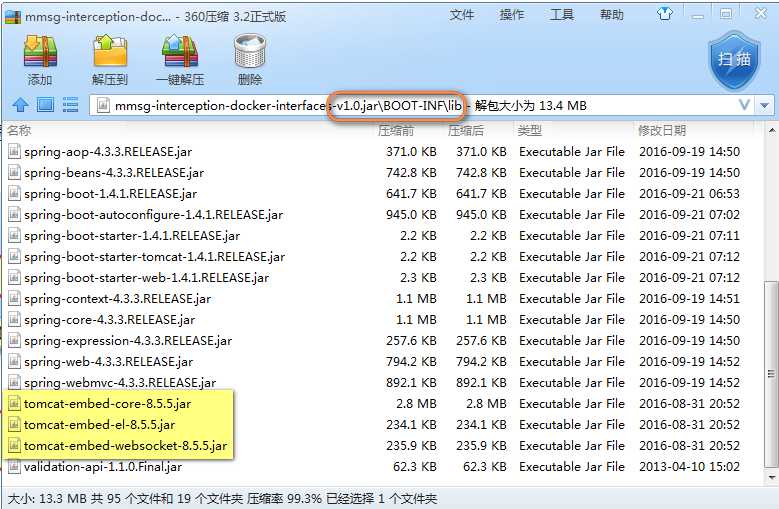
结构是 classes lib 和lib-provided

【我的mmsg和notification-admin这两个项目 就很好的展示了spring-boot打成jar和war的结构 ---- 打成jar的时候 只有class和lib tomcat相关的jar都放到了lib中 因为打成jar 就是确定是依赖 并且 只能是可执行单独运行

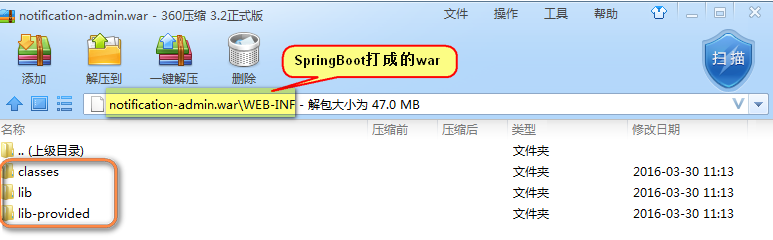
截图贴过来

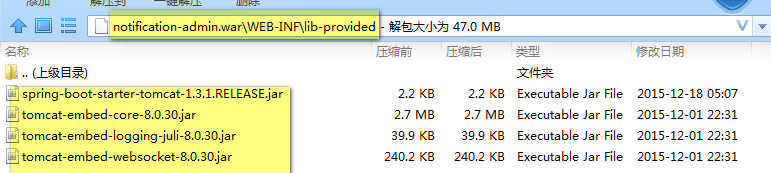


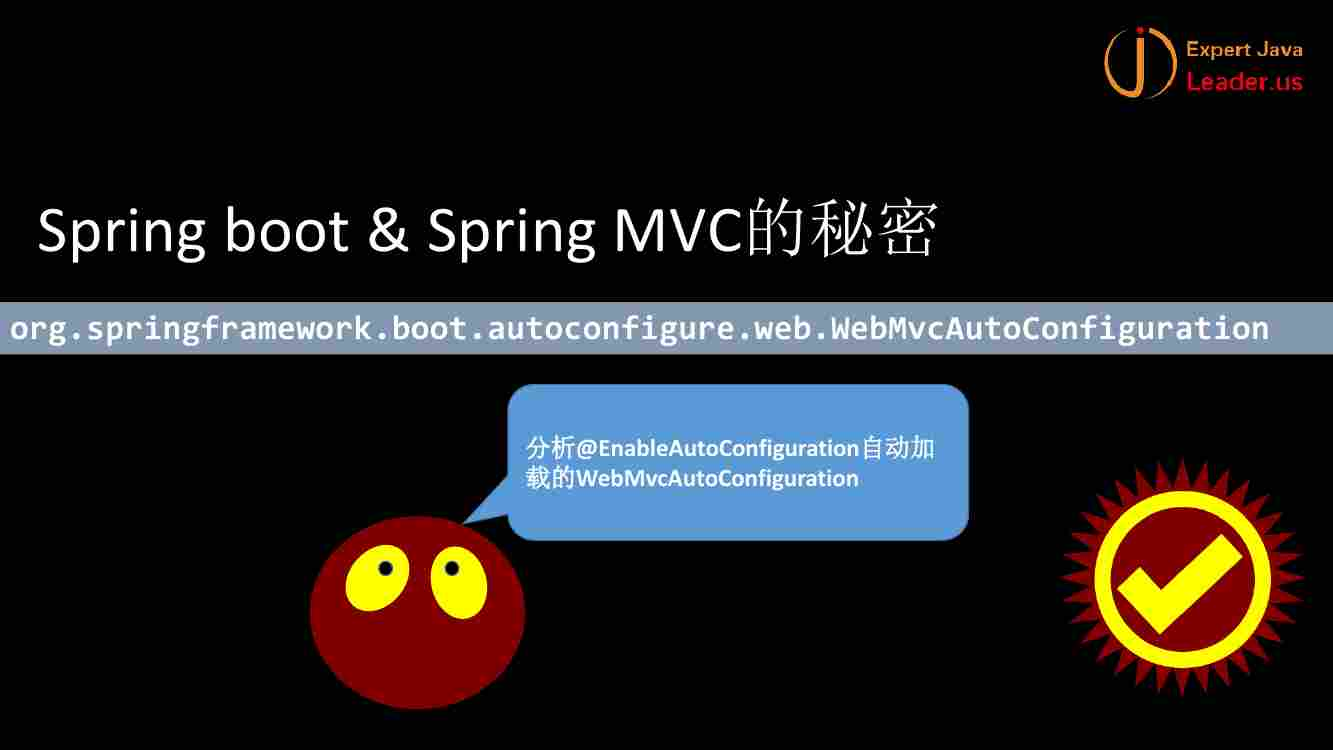
进入到这个lib里面看看

 tomcat相关的jar都在lib里面

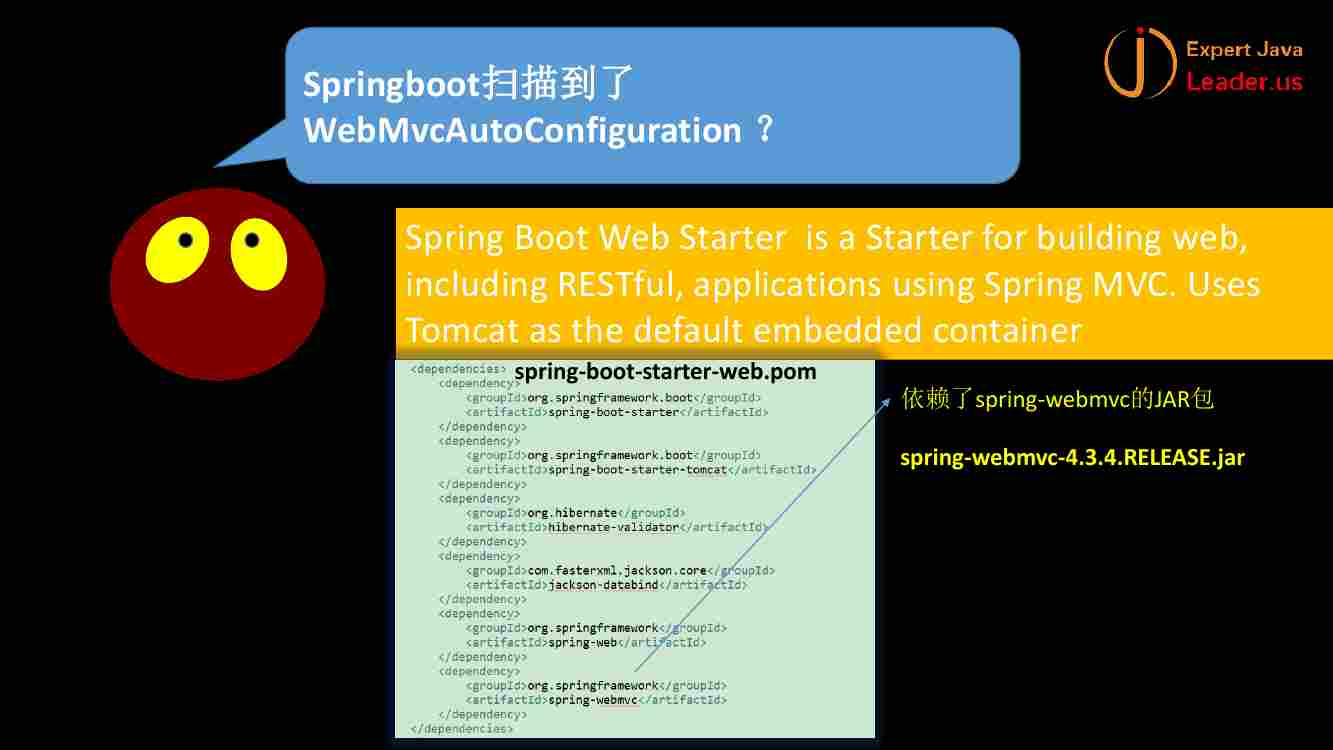
对于 war

 tomcat相关的jar都放在lib-provided这个目录下面

】



SpringBoot和Spring MVC的衔接点就是@EnableAutoConfiguration自动加载了WebMVCAutoConfiguration这个配置



因为Spring Boot Web Starter的pom中定义了这些jar ----- 因此Spring Boot的 WebMvcAutoConfiguration可以生效

【PPT中说 Spring Boot Web Starter是一个用来构建web 包括RESTful和使用SpringMVC的应用的Starter ---- 使用了Tomcat作为默认的嵌入式容器】



从上面分析 只要开启这个@EnableAutoConfiguration 并且引入了合适的jar 一切就变得简单了

所以 以前比较复杂的web程序 在spring boot下面就几句话了

添加三个jar ----- Spring Boot starter web + mybatis-spring-boot-starter + mysql-connector-java这三个依赖的jar 【老师说上一节课中 说了 Spring boot 里面没有mybatis 那么 mybatis自己做了一个spring boot的jar 所以 要单独引入】

在主方法类中 class App

@SpringBootApplication(scanBasePackages=”leader.\*”) --- 指定扫描路径

@MapperScan(“leader.mapping”) ---- 这个注解是Mybatis Spring boot加进来的注解 加载进来所有的mapping定义

然后就是properties的配置文件

全注解完成

【作业题】

如果想看SpringBoot自动注册了多少个Bean --- 可以使用这段代码试试 把Bean得到 ---- 作业题看看 加深理解



如果不愿意这样自动化 可以使用exclude这个属性 把某一个Configuration排除掉 ---然后那就你自己做Configuration

这种排除一般用不到