链滴

Spring-Boot 加载 Bean 的几种方式

作者：pcstar

原文链接：https://ld246.com/article/1543806152685

来源网站：链滴

Spring 从 3.0 之后，就逐步倾向于使用 java code config 方式来进行 bean 的配置，在 spring-oot 中，这种风格就更为明显了。
在查看 spring-boot 工程的时候，总想着探究一下 spring-boot 如何简单的声明一个 starter、Enable**，就能额外增加一个强大的功能，spring 是如何找到这些具体的实现 bean 的呢。
目前，大概有这么几种：

直接在工程中使用 @Configuration 注解
这个就是基本的 java code 方式，在 @SpringBootApplication 里面就包含了 @ComponentScn，因而会把工程中的 @Configuration 注解找到，并加以解释。

通过 @Enable×× 注解里面的 @Import 注解
我们在 Enable 某个功能时，实际上是通过 @Import 注解加载了另外的配置属性类。
例如: 如果要给工程加上定时任务的功能，只需要在某个配置文件上加上 @EnableScheduling 实际上它是引入了 SchedulingConfiguration.class，代码如下：
<pre><code class="language-java highlight-chroma">@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RNTIME)
@Import(SchedulingConfiguration.class)
@Documented
public @interface EnableScheduling {
}
</code></pre>
而 SchedulingConfiguration 就是一个标准的配置文件了，里面定义了 ScheduledAnnotationeanPostProcessor 这个 bean。
<pre><code class="language-java highlight-chroma">@Configuration
@Role(BenDefinition.ROLE_INFRASTRUCTURE)
public class SchedulingConfiguration {
    @Bean(name = TaskManagmentConfigUtils.SHEDULED_ANNOTATION_PROCESSOR_BEAN_NAME)
    <span class="h</code></pre>

ghlight-nd">@Role(
eanDefinition.ROLE_
NFRASTRUCTURE)
    <span class="h
ghlight-kd">public ScheduledAnnotationBeanPostProces
or scheduledAnnotationProcessor<span class=
highlight-o">() {
    <span class="h
ghlight-k">return new <span class="highlight-n
>ScheduledAnnotationBeanPostProcessor();
    <span class="h
ghlight-o">}
<span class="high
ight-o">}
</code></pre>
有了 ScheduledAnnotationBeanPostProcessor 这 bean，就会在 context 初始化时候，查找
们代码中的 @Scheduled，并把它们转换为定时任务。

通过 @EnableAutoConfiguration 注解
添加了这个异常强大的注解，spring-boot 会利用 AutoConfigurationImportSelector 搜索所有
ar 包中 META-INF 文件夹中 spring.factories，找到其中 org.springframework.boot.autoconfigur
.EnableAutoConfiguration 的属性值，并把它作为需要解析的 @Configuration 文件。
例如：spring-cloud-commons 里面的 spring.factories
<pre><code class="highlight-chroma"><span class="highlight
cl"># AutoConfiguration
org.springframew
rk.boot.autoconfigure.EnableAutoConfiguration=\
org.springframew
rk.cloud.client.CommonsClientAutoConfiguration,\
org.springframew
rk.cloud.client.discovery.noop.NoopDiscoveryClientAutoConfiguration,\
org.springframew
rk.cloud.client.hypermedia.CloudHypermediaAutoConfiguration,\
org.springframew
rk.cloud.client.loadbalancer.AsyncLoadBalancerAutoConfiguration,\
org.springframew
rk.cloud.client.loadbalancer.LoadBalancerAutoConfiguration,\
org.springframew
rk.cloud.client.serviceregistry.ServiceRegistryAutoConfiguration,\
org.springframew
rk.cloud.commons.util.UtilAutoConfiguration,\
org.springframew
rk.cloud.client.discovery.simple.SimpleDiscoveryClientAutoConfiguration
</code></pre>

自己实现 ImportSelector
AutoConfigurationImportSelector 显然有时候还是不够用的，这时候就可以自己实现 ImportS
lector，实现更灵活的加载功能。
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">public i
terface ImportSelector <span class="highlight-
">{

    String[] selectImports(AnnotationMetadata importingClassMetadata);
}
</code></pre>
ImportSelector 接口定义了一个 selectImports()方法,根据程序里面定义的注解信息，动态返回被加载的类列表。这个实现就非常灵活了，简单的可以自己判断注解，直接返回类名；复杂的可以自定制类似 AutoConfigurationImportSelector 的功能。
例如：我们看看 spring-cloud 的 @EnableCircuitBreaker
<pre><code class="language-java highlight-chroma">@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RNTIME)
@Documented
@Inherited
@Import(nableCircuitBreakerImportSelector.class)
public @interface EnableCircuitBreaker {
}
</code></pre>
发现，它引入了 EnableCircuitBreakerImportSelector，它本身并没有实现 ImportSelector，是其父类 SpringFactoryImportSelector 实现的。
<pre><code class="language-java highlight-chroma">@Override
public String[] selectImports(AnnotationMetadata metadata) {
if (!isEnable()) {
    return new String[0];
}
AnnotationAttributes attributes <span c

```
ass="highlight-o">=</span> <span class="highlight-n">AnnotationAttributes</span><span
class="highlight-o">.</span><span class="highlight-na">fromMap</span><span class="hi
hlight-o">(</span>
</span></span><span class="highlight-line"><span class="highlight-cl">    <span class="h
ghlight-n">metadata</span><span class="highlight-o">.</span><span class="highlight-na
>getAnnotationAttributes</span><span class="highlight-o">(</span><span class="highlig
t-k">this</span><span class="highlight-o">.</span><span class="highlight-na">annotatio
Class</span><span class="highlight-o">.</span><span class="highlight-na">getName</sp
n><span class="highlight-o">(),</span> <span class="highlight-kc">true</span><span clas
="highlight-o">));</span>
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-n">Assert</span><span class="highlight-o">.</span><span class="highlight-na">not
ull</span><span class="highlight-o">(</span><span class="highlight-n">attributes</span
<span class="highlight-o">,</span> <span class="highlight-s">"No "</span> <span class=
highlight-o">+</span> <span class="highlight-n">getSimpleName</span><span class="hi
hlight-o">()</span> <span class="highlight-o">+</span> <span class="highlight-s">" attri
utes found. Is "</span>
</span></span><span class="highlight-line"><span class="highlight-cl">    <span class="h
ghlight-o">+</span> <span class="highlight-n">metadata</span><span class="highlight-o
>.</span><span class="highlight-na">getClassName</span><span class="highlight-o">()<
span> <span class="highlight-o">+</span> <span class="highlight-s">" annotated with @"
/span> <span class="highlight-o">+</span> <span class="highlight-n">getSimpleName</
pan><span class="highlight-o">()</span> <span class="highlight-o">+</span> <span clas
="highlight-s">"?"</span><span class="highlight-o">);</span>
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-c1">// Find all possible auto configuration classes, filtering duplicates
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-c1">// 调用SpringFactoriesLoader的loadFactoryNames去加载
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-c1"></span><span class="highlight-n">List</span><span class="highlight-o"
&amp;lt;</span><span class="highlight-n">String</span><span class="highlight-o">&amp;gt;</spa
> <span class="highlight-n">factories</span> <span class="highlight-o">=</span> <span
lass="highlight-k">new</span> <span class="highlight-n">ArrayList</span><span class="h
ghlight-o">&amp;lt;&amp;gt;(</span><span class="highlight-k">new</span> <span class="highlight
n">LinkedHashSet</span><span class="highlight-o">&amp;lt;&amp;gt;(</span><span class="highlig
t-n">SpringFactoriesLoader</span>
</span></span><span class="highlight-line"><span class="highlight-cl">    <span class="h
ghlight-o">.</span><span class="highlight-na">loadFactoryNames</span><span class="hi
hlight-o">(</span><span class="highlight-k">this</span><span class="highlight-o">.</sp
n><span class="highlight-na">annotationClass</span><span class="highlight-o">,</span>
<span class="highlight-k">this</span><span class="highlight-o">.</span><span class="hi
hlight-na">beanClassLoader</span><span class="highlight-o">)));</span>
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-c1">//省略了错误判断和多于一个的log
</span></span></span><span class="highlight-line"><span class="highlight-cl"><span cla
s="highlight-c1"></span><span class="highlight-k">return</span> <span class="highlight
n">factories</span><span class="highlight-o">.</span><span class="highlight-na">toArra
</span><span class="highlight-o">(</span><span class="highlight-k">new</span> <span
lass="highlight-n">String</span><span class="highlight-o">[</span><span class="highligh
-n">factories</span><span class="highlight-o">.</span><span class="highlight-na">size<
span><span class="highlight-o">()]);</span>
</span></span><span class="highlight-line"><span class="highlight-cl"><span class="high
ight-o">}</span>
</span></span></code></pre>
```

这里，我们看到，实际加载的代码是传入了 this.annotationClass，那么对于 EnableCircuitBreaerImportSelector 来说，就是在 spring.factories 找它的全类名：
org.springframework.cloud.client.circuitbreaker.EnableCircuitBreakerImportSelector 对应的值

最终在 spring-cloud-netflix-core-××.jar 的 pring.factories 中找到如下配置：
<pre><code class="highlight-chroma">org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker=\
org.springframework.cloud.netflix.hystrix.HystrixCircuitBreakerConfiguration
</code></pre>
这样就完成了通过 @EnableCircuitBreaker 的注解，最终加载到 Hystrix 的实现 ystrixCircuitBrakerConfiguration，实现了功能定义和具体实现的分离。

