Spring Boot自动配置是如何实现的

作者：lconline
原文链接：https://ld246.com/article/1470577459907
来源网站：链滴

Spring Boot习惯优于配置的理念可以让项目快速的运行起来，使用它可以不用或者只需要很少Spring配置，因为自动配置AutoConfiguration已经为我们做了很多工作。那么自动配置是如何实现呢？
<h3>1 Spring中的Profiles特性</h3>
在开发Spring应用的时候，可能会根据不同的情况注册不同的bean。例如，在开发环境和生产境会使用不同的数据源。常见的思路则是，把数据源配置写在配置文件中，但这样的话，不同的环境需要更改相应的配置，并重新打包生成应用。
为解决这样的问题，从Spring3.1开始引入了Profiles的概念。我们可以注册多个相同类型的bea，并将它们关联到一个或者多个profiles。应用运行的时候，则可以激活期望的profiles，同时跟这些profiles相关联的beans就可以自动被注册。
对于刚才所提到的开发环境和生产环境使用不同的数据源的情况，则可以这样来做：
<pre class="brush: java">@Configuration
public class AppConfig{
    @Bean
    @Profile("DEV")
    public DataSource devDataSource() {
        ...
    }

@Bean
@Profile("PROD")
public DataSource prodDataSource() {
    ...
}

}</pre>

然后通过启动选项-Dspring.profiles.active=DEV便可注册不同的数据源了。
上面通过指定不同profiles来注册不同beans的方法，往往只适用于一些很简单的业务场景，如注册beans的逻辑复杂起来，profiles的方式则显得力不从心了。
为了提供更多灵活的通过不同条件判断来注册beans的方式，Spring 4引入了@Conditional条注解，可在多种条件判断下选择性的注册beans。
<h3>2 @Conditional条件注解</h3>
我们可能在如下某种情况下才需要注册某个bean，而通过Spring的@Conditional都能够实现这既定条件下的bean注册：

classpath中存在某个class的时候
某个指定类型的bean在ApplicationContext中不存在的时候
某个文件在指定路径中存在的时候
某个配置项在配置文件中存在的时候

下面就来看看Spring的@Conditional是如何工作的，以及通过它怎么实现不同条件下bean的注。
假设有一个UserDAO接口，通过接口中的方法可从数据库中得到数据。这个接口有两个实现：JdcUserDAO和MongoUserDAO，分别用来操作MySQL数据库和MongoDB数据库。
<pre class="brush: java">public interface UserDAO{
    List<String> getAllUserNames();
}
 public class JdbcUserDAO implements UserDAO{
    @Override
    public List<String> getAllUserNames()
    {
        System.out.println("**** Getting usernames from RDBMS *****");

```
        return Arrays.asList("Siva","Prasad","Reddy");
    }
}
 public class MongoUserDAO implements UserDAO{
    @Override
    public List&lt;String&gt; getAllUserNames()
    {
        System.out.println("**** Getting usernames from MongoDB *****");
        return Arrays.asList("Bond","James","Bond");
    }
}</pre>
<p>接下来将通过使用@Conditional条件注解，展示几种在不同条件下注册不同bean的例子。</p>
<h3>2.1 通过启动参数注册不同的bean</h3>
<p>我们希望通过dbType这样一个启动参数来选择性地使用其中一个实现：</p>
<p>（1）java -jar myapp.jar -DdbType=MySQL，使用JdbcUserDAO</p>
<p>（2）java -jar myapp.jar -DdbType=MONGO，使用MongoUserDAO</p>
<p>下面，分别实现Condition接口，用于判断dbType系统属性：</p>
<pre class="brush: java">public class MySQLDatabaseTypeCondition implements Condition{
    @Override
    public boolean matches(ConditionContext conditionContext, AnnotatedTypeMetadata met
data)
    {
        String enabledDBType = System.getProperty("dbType");
        return (enabledDBType != null &amp;amp;&amp;amp; enabledDBType.equalsIgnoreCase("MYSQL"
);
    }
}
public class MongoDBDatabaseTypeCondition implements Condition{
    @Override
    public boolean matches(ConditionContext conditionContext, AnnotatedTypeMetadata met
data)
    {
        String enabledDBType = System.getProperty("dbType");
        return (enabledDBType != null &amp;amp;&amp;amp; enabledDBType.equalsIgnoreCase("MONG
DB"));
    }
}</pre>
<p>好了，接下来就可以使用@Conditional注解来对beans进行配置了：</p>
<pre class="brush: java">@Configuration
public class AppConfig
{
    @Bean
    @Conditional(MySQLDatabaseTypeCondition.class)
    public UserDAO jdbcUserDAO(){
        return new JdbcUserDAO();
    }


@Bean
@Conditional(MongoDBDatabaseTypeCondition.class)
public UserDAO mongoUserDAO(){
    return new MongoUserDAO();
}
```

}</pre>

现在，如果使用java -jar myapp.jar -DdbType=MYSQL命令运行，那么只有JdbcUserDAO这bean会被注册；同理，如果参数-DdbType=MONGODB，那么则只有MongoUserDAO这个bean被注册。
<h3>2.2 通过判断classpath中的class存在与否注册不同的bean</h3>
假设如果MongoDB数据库驱动类“com.mongodb.Server”在classpath中存在则注册MongoUserDAO这个bean，否则则注册JdbcUserDAO bean。
要完成这一需求，我们可以通过如下方式来检查驱动是否存在，同样需要实现Condition接口：
<pre class="brush: java">public class MongoDriverPresentsCondition implements Condition{
    @Override
    public boolean matches(ConditionContext conditionContext,AnnotatedTypeMetadata metadata)
    {
        try {
            Class.forName("com.mongodb.Server");
            return true;
        } catch (ClassNotFoundException e) {
            return false;
        }
    }
}
 public class MongoDriverNotPresentsCondition implements Condition{
    @Override
    public boolean matches(ConditionContext conditionContext, AnnotatedTypeMetadata metadata)
    {
        try {
            Class.forName("com.mongodb.Server");
            return false;
        } catch (ClassNotFoundException e) {
            return true;
        }
    }
}</pre>
如果没有类型为UserDAO的bean，则注册MongoUserDAO bean：
<pre class="brush: java">public class UserDAOBeanNotPresentsCondition implements Condition
{
    @Override
    public boolean matches(ConditionContext conditionContext, AnnotatedTypeMetadata metadata)
    {
        UserDAO userDAO = conditionContext.getBeanFactory().getBean(UserDAO.class);
        return (userDAO == null);
    }
}</pre>
<h3>2.3 通过配置项注册不同的bean</h3>
如果配置文件中，存在配置app.dbType=MONGO，则注册MongoUserDAO bean：
<pre class="brush: java">public class MongoDbTypePropertyCondition implements Conditio
{
    @Override
    public boolean matches(ConditionContext conditionContext,

```
    AnnotatedTypeMetadata metadata)
    {
        String dbType = conditionContext.getEnvironment()
                        .getProperty("app.dbType");
        return "MONGO".equalsIgnoreCase(dbType);
    }
}</pre>
<h3>2.4 通过自定义注解注册不同的bean</h3>
<p>以上我们已经看到了多种通过不同条件注册不同bean的例子，接下来展示一种更加优雅的方法
那就是通过自定义注解来完成。</p>
<p>首先，定义一个DatabaseType注解：</p>
<pre class="brush: java">@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@Conditional(DatabaseTypeCondition.class)
public @interface DatabaseType
{
    String value();
}</pre>
<p>接下来，实现Condition接口，通过比较DatabaseType注解的值，和dbType系统参数的值，来
断使用哪个bean：</p>
<pre class="brush: java">public class DatabaseTypeCondition implements Condition{
    @Override
    public boolean matches(ConditionContext conditionContext,
    AnnotatedTypeMetadata metadata)
    {
        Map&amp;lt;String, Object&amp;gt; attributes = metadata.getAnnotationAttributes(DatabaseType
class.getName());
        String type = (String) attributes.get("value");
        String enabledDBType = System.getProperty("dbType","MYSQL");
        return (enabledDBType != null &amp;amp;&amp;amp; type != null &amp;amp;&amp;amp; enabledDBType.
qualsIgnoreCase(type));
    }
}</pre>
<p>然后，则可以在获得bean的方法上使用@DatabaseType注解了：</p>
<pre class="brush: java">@Configuration
@ComponentScan
public class AppConfig{
    @DatabaseType("MYSQL")
    public UserDAO jdbcUserDAO(){
        return new JdbcUserDAO();
    }


@Bean
@DatabaseType("MONGO")
public UserDAO mongoUserDAO(){
    return new MongoUserDAO();
}

}</pre>

<p>在DatabaseTypeCondition的matches方法中，通过比较DatabaseType注解的元数据与系统
性dbType，最终确定哪个bean被注册。</p>
<p>以上则是几个使用@Conditional条件注解按照不同的业务逻辑注册bean的例子。在看了这么多
```

子之后，接下来该进入我们的主题，Spring Boot自动配置了。
<h3>3 Spring Boot自动配置</h3>
在Spring Boot中大量使用了@Conditional特性，在不同的情况下注册不同的bean。
在spring-boot-autoconfigure-{version}.jar这个文件的Org.springframework.boot.
autoconfigure包中，可以看到许多Condition接口的实现类。那么Spring Boot中的自动配置机是如何触发的呢？
下面的代码段，大家应该都不陌生：
<pre class="brush: java">@Configuration
@EnableAutoConfiguration
@ComponentScan
public class Application {

}</pre>

在Spring Boot应用程序入口类中，我们一般都会使用@Configuration，@EnableAutoConfiguration，@ComponentScan三个注解，或者使用@SpringBootApplication来代替。@EnableAutoCnfiguration注解则是Spring Boot中，自动配置的关键。该注解打开Spring ApplicationContext的动配置功能，扫描classpath下的components，并根据不同的条件注册beans。
在spring-boot-autoconfigure-{version}.jar中这个jar包中，Spring Boot提供了许多AutoConfguration类，他们负责注册不同的components。
典型的AutoConfiguration类有@Configuration和@EnableConfigurationProperties两个注。@Configuration说明这个来是一个Spring的配置类，@EnableConfigurationProperties用来绑自定义属性以及一个或多个bean的条件注册方法。
下面来看一下源码中的AutoDataSourceAuoConfiguration这个类，源码太长，不在这里贴出来了，完整源码请搭梯子去github看。传送门
在这个类中：
（1）@ConditionalOnClass({ DataSource.class,EmbeddedDatabaseType.class }) 注解说明只有DataSource.class,EmbeddedDatabaseType.class在classpath中存在的情况下，类中bean的动配置才会生效
（2）@EnableConfigurationProperties(DataSourceProperties.class) 注解说明，application properties配置文件中的属性会与DataSourceProperties中的相应字段进行绑定。
让我们在看看DataSourceProperties类：
<pre class="brush: java">@ConfigurationProperties(prefix = DataSourceProperties.PREFIX)
public class DataSourceProperties implements BeanClassLoaderAware, EnvironmentAware, InializingBean {

public static final String PREFIX = "spring.datasource";
...
...
private String driverClassName;
private String url;
private String username;
private String password;
...
//setters and getters

}</pre>

通过源码不难看到，配置文件中所有以spring.datasource.*开头的配置项，都会与DataSourceProperties中的相应字段想绑定。例如如下的配置：
<pre class="brush: plain">spring.datasource.driverClassName=org.postgresql.Driver

spring.datasource.url=jdbc:postgresql://192.168.249.130/jbase
spring.datasource.username=postgres
spring.datasource.password=postgres</pre>
在源码中，在许多内部类和bean定义方法上，均能看到Spring Boot的条件注解，例如：

@ConditionalOnMissingBean
@ConditionalOnClass
@ConditionalOnProperty

等等。只有满足条件，那些bean才会被注册。
<h3>4 小结</h3>
通过开发过程中切换不同数据源的例子，分别使用Spring中的Profiles特性以及Condition条件解来解决这一问题，并最终简单说明了Spring Boot的自动配置。
点下载PDF阅读版