链滴

Java SPI

作者：noelcliu

原文链接：https://ld246.com/article/1669895964044

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20220801.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h2 id="是什么">是什么</h2>
Java SPI 的全称是 Java Service Provider Interface，是一种动态加载服务的机制。这些专有名
听起来有点难理解，比较抽象，其实从用法和最后的实现结果上来看，Java SPI 就是一个让开发者可
使用配置文件来动态指定某个接口或者抽象类的具体实现是哪一个类的机制。我们下面直接使用它，
可以更直观的感受到它有着什么功能。
<h2 id="怎么用">怎么用</h2>
<h3 id="理论基础">理论基础</h3>
在用之前我们需要一些理论基础，对于 Java SPI 而言，就是它让我们使用所制定的规则是什么
这就像玩具的说明书一样。首先再次重复一下 Java SPI 的作用 —— 让开发者可以使用配置文件来动
指定某个接口或者抽象类的具体实现是哪一个类的机制。可以看到文字中加粗的部分，要完成这个功
，加粗的部分是必不可少的，所以我们首先得提供这些加粗部分的内容，才可以使用 Java SPI。
<h3 id="Java-SPI-组件">Java SPI 组件</h3>
承接上文，介绍一下 Java SPI 的四个重要组件（注意，这里的组件是逻辑概念，具体对应的实
在解释中）：

Service Provider Interface：服务提供方接口，其实它对应的就是上文中的接口或抽象类；
Service Provider：服务提供方，对应上文中的具体实现（一个或多个）；
SPI Configuration File：SPI 配置文件，很明显对应上文中的配置文件，这个配置文件的具体位
就在
resources 目录下的 META-INF/services 目录下，具体使用方法我们在例子中再介绍；
ServiceLoader：这个在上文中没有提到，其实可以想象得到，配置文件、接口（或抽象类）、
现都有了，
还需要的就是从代码中获取该接口的具体实现是什么，而这个 ServiceLoader 就是用来获取接口的具
实现类结果的角色。

理论基础介绍完毕了，下面我们来实操看看。
<h3 id="代码编写">代码编写</h3>
承接上文，我们得先准备接口（或抽象类）、具体实现和配置文件。
<h4 id="编写接口">编写接口</h4>
我们定义一个 MusicInstrument（乐器）接口，就定义一个方法 play()（演奏）；
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">public i
terface MusicalInstrument <span class="highli
ht-o">{

  <span class="hi
hlight-kt">void play
);
<span class="high
ight-o">}
</code></pre>
<h4 id="编写具体实现">编写具体实现</h4>
具体实现类定义两个：Throat（歌喉）、Guitar（吉他）：
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">public c
ass Throat implem
nts MusicalInstrument <span class="highlight-o
>{
  <span class="hi
hlight-nd">@Override
  <span class="hi
hlight-kd">public void <span class="highlight-

f">play() {
  <span class=
highlight-n">System.<span class="highlight-na
>out.println</span
("我一路向北～ 离开有你的季
～");
  <span class="hi
hlight-o">}
<span class="high
ight-o">}
</code></pre>
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">public c
ass Guitar implem
nts MusicalInstrument <span class="highlight-o
>{
  <span class="hi
hlight-nd">@Override
  <span class="hi
hlight-kd">public void <span class="highlight-
f">play() {
  <span class=
highlight-n">System.<span class="highlight-na
>out.println</span
("Guitar Play～"<spa
class="highlight-o">);
  <span class="hi
hlight-o">}
<span class="high
ight-o">}
</code></pre>
<h4 id="编写配置文件">编写配置文件</h4>
Java SPI 的配置文件是存储在 classpath 下的 META-INF 文件夹的 services 目录下的，文件名
定义好的接口（或抽象类）的名字，而文件中的值则为接口（或抽象类）的 具体实现类的包名 + 类
：
配置文件路径及名称：
<img src="https://ld246.com/images/img-loading.svg" alt="640.png" data-src="https://
3logfile.com/file/2022/12/640-0e7boF0.png?imageView2/2/interlace/1/format/jpg">
配置文件内容：
<img src="https://ld246.com/images/img-loading.svg" alt="641.png" data-src="https://
3logfile.com/file/2022/12/641-HAhtsRY.png?imageView2/2/interlace/1/format/jpg">
<h4 id="编写-main-方法">编写 main 方法</h4>
接下来我们使用 ServiceLoader 来获取具体指定的实现类，并调用 play 方法：
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">public c
ass Main {

  <span class="hi
hlight-kd">public static <span class="highlight
kt">void main(</sp
n>String[] <span cla
s="highlight-n">args) <span class="highlight-o
>{

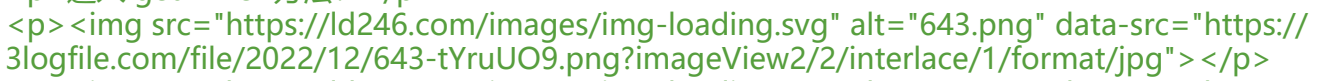
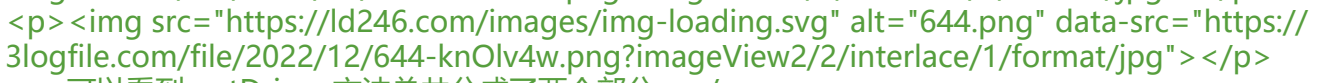
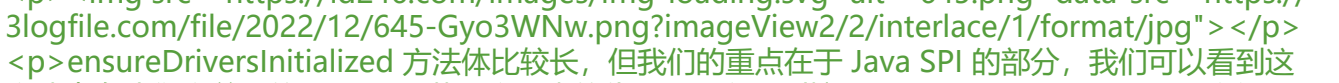
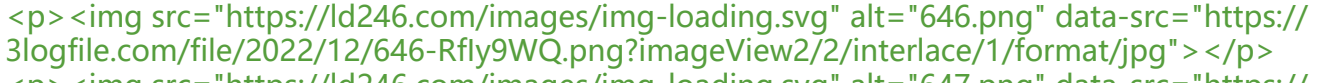
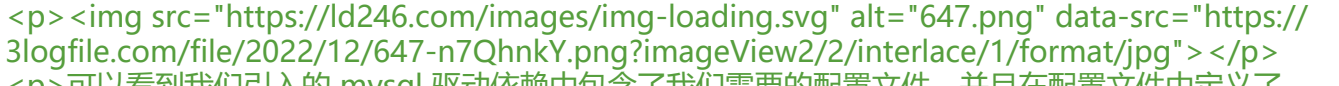
        <span class=
highlight-c1">// 获取具体实现类对象集合
<span cla
s="highlight-c1">   ServiceLoader<span class
"highlight-o"><MusicalInstrument<span clas
="highlight-o">> serviceLoader <span class=
highlight-o">= ServiceLoader<span class="highl
ght-o">.load(</sp
n>MusicalInstrument.</span
class);

        <span class=
highlight-c1">// 遍历调用play方法
<span cla
s="highlight-c1">   for <span class="highlight
o">(MusicalInstrument <span class="highlight-
">service : serviceLo
der) {
            <span clas
="highlight-n">service.<span class="highlight-
a">play();
        <span class="
ighlight-o">}
    <span class="hi
hlight-o">}
<span class="high
ight-o">}
</code></pre>
最终打印结果：
<img src="https://ld246.com/images/img-loading.svg" alt="642.png" data-src="https://
3logfile.com/file/2022/12/642-Yg41Ass.png?imageView2/2/interlace/1/format/jpg">
可以看到在代码中动态获取了配置文件中定义好的实现类的实例，并且成功打印了 play 方法中
内容。
<h2 id="应用场景">应用场景</h2>
Java SPI 最典型的应用场景就是数据库驱动，Sun 公司对于数据库驱动提供了统一的 jdbc 接口
但是每个数据库的驱动需要单独开发，使用了 Java SPI 之后，在使用 DriverManager 的时候就会自
的加载系统中所有引入的数据库驱动，接下来我们翻翻源码 ~
首先我们要在项目 pom 文件中添加相应的驱动依赖，这里我们使用 MySQL：
<pre><code class="language-xml highlight-chroma"><span cl
ss="highlight-cl"><dependency>
    <span class="hi
hlight-nt"><groupId>mysql</groupId></spa
>
    <span class="hi
hlight-nt"><artifactId>mysql-connector-java</ar
ifactId>
    <span class="hi
hlight-nt"><version>8.0.27</version></span

<span class="high
ight-nt"></dependency>
</code></pre>
然后我们在 main 方法中编写这样的代码：
<pre><code class="language-java highlight-chroma"><span c

ass="highlight-cl">public class Main {

  public static void main(String[] args) throws SQLException {

      DriverManager.getDriver("jbc:mysql://localhost:3306/test");
  }
}
</code></pre>
接着运行 main 方法，然后我们 debug 看看底层都经过了什么处理 ~
进入 getDriver 方法：


可以看到 getDriver 方法总共分成了两个部分：
<pre><code class="language-md highlight-chroma">1.确认 driver 是否全部初始化;
2.确认 registeredDivers 中是否包含可以处理参数 url 的 driver。
</code></pre>
而我们所需要关注的是 1，这个方法里面就处理了所有的 driver 的初始化，让我们进入 ensureDriversInitialized 方法看看 ~
进入 ensureDriversInitialized 方法：

ensureDriversInitialized 方法体比较长，但我们的重点在于 Java SPI 的部分，我们可以看到这方法中有我们之前写的 Java SPI 获取实现类的代码，可以看到接口是 com.mysql.Driver，那么问题了，我们没有定义过配置文件也没有指定过实现，那这些代码到底加载了什么呢？这个时候我们就得去 mysql 依赖的 jar 包里面看看了 ~


可以看到我们引入的 mysql 驱动依赖中包含了我们需要的配置文件，并且在配置文件中定义了的 Driver 具体实现类 com.mysql.cj.jdbc.Driver，这样一下子就都明了了，在我们使用 DriverManaer 的时候，DriverManager 会首先去确认所有的 Driver 有没有被初始化完成，如果没有被初始化完则进行初始化操作，初始化操作其实分为两个部分，第一个是通过 Java SPI 加载不同的驱动 jar 包中配置文件所指定的实现类，第二个是使用系统配置 jdbc.drivers 来加载驱动类，这样就可以将驱动的载对使用者完全透明，也无需具体指定所需要的驱动实现类，真的是妙呀 ~
<h2 id="总结">总结</h2>
本篇文章介绍了 Java SPI，带领大家从 Java SPI 的理论基础到 demo 编写，也解析了 Java SPI 具体使用场景，文章最后建议大家去看看 DriverManager 使用 Java SPI 加载完驱动之后是怎么处理

入的 url 的，特别是驱动的实例是如何添加到 registeredDrivers 中的，最后，文章中的代码可以 java-spi 在查看。/p>