

SPI 机制 - 插件化扩展功能

作者: [9526xu](#)

原文链接: <https://ld246.com/article/1567043902976>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

SPI(Service Provider Interfaces),中文直译服务提供者接口,一种服务发现机制。可能很多人都不太悉这个机制,但是平常或多或少都用到了这个机制,比如我们使用 JDBC 连接操作数据库的时候。

SPI 主要适用于功能扩展的场景,如一些框架提供某一部分功能可以由第三方开发人员扩展,满足其身业务需求。

假设我们在公司内实现了一个统一登陆框架,框架内部仅提供用户名/密码登陆方式。后来 A 部门使用该框架,但是他们想增加微信登陆授权。正常情况下,我们可以改动登陆框架代码,增加微信登录实现方式。如果后面又增加 QQ 登陆,淘宝登陆那?也只能不断相应的实现。

SPI 实现方式

这种情况如果使用 SPI,可以在不用改动框架代码前提下,增加新的登陆实现方式。下面用代码演示何使用 SPI。

定义接口

首先我们新建一个 maven 项目 `oauth-api`,在这个项目创建一个公共接口。

```
public interface OauthLoginService {  
    void login();  
}
```

第三方实现该接口

再新建一个 maven 项目 `wechat-oauth`,引入上面 `oauth-api` 依赖

```
public class WechatLoginService implements OauthLoginService {  
    @Override  
    public void login() {  
        System.out.println("使用微信登陆授权");  
    }  
}
```

定义配置文件

SPI 需要将接口实现定义在配置文件中,文件名为接口全名称,如 `com.andyxh.OauthLoginService` 配置文件需放在 `resources\META-INF\services` 文件夹下。文件内容如下:

```
com.another.WechatLoginService
```

加载接口实现类

新建 maven 项目 `oauth-login`,在这个项目中引入 `wechat-oauth` 与 `oauth-api` 依赖。SPI 核心将会用 `java.util.ServiceLoader` 读取上面上面定义配置文件,加载所有服务实现类。使用代码如下:

```
ServiceLoader<OauthLoginService> serviceLoader=ServiceLoader.load(OauthLoginService.class);  
serviceLoader.forEach(OauthLoginService::login);
```

打印结果:

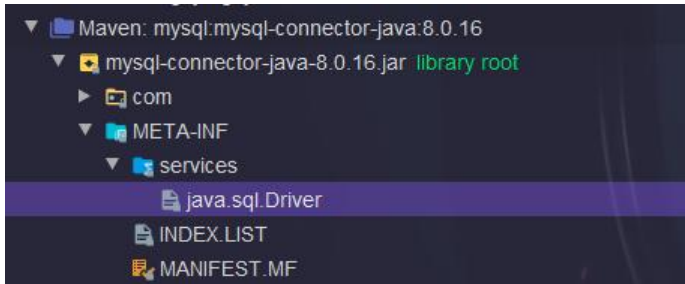
```
使用微信登陆授权
```

SPI 实际应用

上面说过 JDBC 中使用到 SPI 机制。JDK 定义标准数据库接口，相应的数据库厂商实现这类接口。以 `mysql-connector-java` 为例。

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.16</version>
</dependency>
```

mysql jar 包 META-INF/services 中存在 `java.sql.Driver` 文件，这个文件定义了实现类。



`com.mysql.cj.jdbc.Driver`

可以看到 `java.sql.Driver` 是标准 SPI 接口，而 `com.mysql.cj.jdbc.Driver` 是 mysql 标准实现接口。

何时加载 `java.sql.Driver`?

我们将会使用 `DriverManager.getConnection` 获取相应数据库连接。这个类内部存在一个静态代码块，将会使用 `ServiceLoader` 加载实现类。

```
static {
    loadInitialDrivers();
    println("JDBC DriverManager initialized");
}

private static void loadInitialDrivers() {
    ....
    ServiceLoader<Driver> loadedDrivers = ServiceLoader.load(Driver.class);
    Iterator<Driver> driversIterator = loadedDrivers.iterator();
    try{
        while(driversIterator.hasNext()) {
            driversIterator.next();
        }
    } catch(Throwable t) {
        // Do nothing
    }
    return null;
}
....
}
```

Java SPI 存在问题

ServiceLoader 一次性将会实例化所有实现，但是如果没有某一扩展初始化耗时很久，但是却不需要刻使用，就会非常浪费资源。

基于这个问题，Dubbo SPI 机制改进 Java SPI 的不足，做到按需加载并且增加 ioc 与 aop 的功能
下篇文章可以在具体聊聊，敬请期待。