简单易懂的 Java SPI 入门指南

SPI（Service Provider Interface，服务提供者接口）是一种API（Application Programming Interface，应用程序编程接口），它允许框架或模块在运行时动态地加载实现了某个接口或抽象类的具体实现类。SPI 主要用于模块化系统中，帮助应用程序实现松耦合、扩展性强的设计。学习**SPI** ，能够让你更好地了解Java中各种主流的框架，因为SPI机制在许多广泛使用的Java框架中扮演着关键角色。

一、介绍

1. SPI：**Service Provider Interface**，服务提供接口机制。允许框架和应用程序从外部实现或插件中加载类，以达到拓展和**易插拔**的功能。
2. SPI 核心思想：将接口的**定义**和**实现**分离，通过配置文件的形式来动态加载实现类，从而实现解耦。
3. SPI 优点：

* **松耦合**：通过接口和实现分离，实现类的变化不会影响调用者。
* **灵活性**：可以在运行时动态加载实现类。
* **扩展性强**：可以通过新增实现类和配置文件，无需修改现有代码即可扩展系统功能。

4. SPI 缺点：

* **性能开销**：需要遍历所有的实现类，不能按需加载，效率低。
* **配置繁琐**：每次新增实现类都需要修改配置文件。

二、应用流程

1. **定义服务接口**：首先定义一个服务接口（Service Interface），该接口描述了服务的行为和方法。
2. **提供服务实现**：然后，一个或多个服务提供者（Service Provider）实现该服务接口。
3. **创建配置文件**：在`META-INF/services`目录下创建一个以服务接口全限定名命名的文件，文件内容为具体实现类的全限定名。
4. **加载服务实现**：通过`ServiceLoader`类动态加载并实例化服务提供者的实现类。

> `META-INF/services` 目录是在 Java 项目中用于存放服务提供者配置文件的特殊目录。 Java 提供的 `ServiceLoader` 类会自动查找并加载 `META-INF/services` 目录中的配置文件，根据文件中的内容加载相应的实现类。

三、SPI 的常见应用场景

SPI机制广泛应用于各种Java框架和库中，以下是一些常见的应用场景：

1. **日志框架**：如SLF4J，通过SPI机制加载不同的日志实现（如Logback，Log4j）。
2. **JDBC**：Java数据库连接（JDBC）使用SPI机制加载不同的数据库驱动程序。
3. **Servlet容器**：如Tomcat，Jetty，通过SPI机制加载和扩展不同的Servlet实现。

四、例子

1. 步骤 1：定义服务接口——定义一个`GreetingService`的服务接口，它包含一个`greet`方法。

```
public interface GreetingService {
    void greet(String name);
}
```

2. 步骤 2：提供服务实现——提供两个实现这个接口的类，分别是`EnglishGreetingService`和`SpanishGreetingService`。它们都实现了`GreetingService`接口，并提供了具体的问候语。

```
public class EnglishGreetingService implements GreetingService {
    @Override
```

```
    public void greet(String name) {
        System.out.println("Hello, " + name);
    }
}

public class SpanishGreetingService implements GreetingService {
    @Override
    public void greet(String name) {
        System.out.println("Hola, " + name);
    }
}
```

3. 步骤 3：创建配置文件——为了让SPI机制能够找到这些实现类，我们需要在`META-INF/services`目录下创建一个名为`com.example.GreetingService`的文件。这个文件的内容是服务实现类的全类名，每行一个；这个文件告诉`ServiceLoader`要加载哪些类作为`GreetingService`的实现。

```
com.example.EnglishGreetingService
com.example.SpanishGreetingService
```

4. 步骤 4：加载服务实现——使用`ServiceLoader`来加载并使用这些服务实现。

```
import java.util.ServiceLoader;

public class Main {
    public static void main(String[] args) {
        ServiceLoader<GreetingService> serviceLoader =
ServiceLoader.load(GreetingService.class);
        for (GreetingService service : serviceLoader) {
            service.greet("World");
        }
    }
}
```

> `ServiceLoader`会读取`META-INF/services`目录下的配置文件，加载配置文件中指定的实现类，并创建其实例，按需配置需要的实现类则可以实现动态插拔效果。

* 运行结果：

> Hello, World
>
>
> Hola, World

* 若配置文件中只有`com.example.EnglishGreetingService`这条记录，则只会调用`EnglishGreetingService` 的 `greet`方法，只输出`Hello, World`。由于两个实现类都进行了配置，所以都调用了对应的 `greet`方法实现。
 原文链接: https://juejin.cn/post/7393306884537548809