Dubbo序列化和反序列化模块（三） 设计模式和高级设计意图

曹瀚云

2018K8009907008

1. 前言

本文作为王伟老师OOP课程的大作业——Dubbo序列化和反序列化模块源码分析报告的第三部分。在前面两部分中，第一部分是对Dubbo项目的内容作了简要介绍，并说明了什么是序列化，简单列出了在Github的Dubbo仓库serialization部分的文件层次和模块（类/接口）之间的关系。考虑到序列化部分Dubbo提供了包括avro，FastJson，fst，kyro在内的多种序列化/反序列化方法，由于本人能力有限，暂时不去详细分析各种方法的具体源码（后续的补充可能会涉及），只是以常见的FastJson作为一个例子进行分析。第二部分则更详细地去剖析源码，以FastJson为例理清了接口和实现类之间的关系以及多态思想的体现。在这一部分，我们主要来分析该模块等设计模式和高级设计意图。

2. 设计模式和高级设计意图分析

在Serialization.java文件的import语句后的第一句注释如下：

```
Serialization strategy interface that specifies a serializer. (SPI, Singleton, ThreadSafe)
```

它涉及到了两个很重要的知识点：SPI和Singleton。

（1）SPI

SPI全称Service Provider Interface，是Java提供的一套用来被第三方实现或者扩展的API，它可以用来启用框架扩展和替换组件。

其整体机制图如下：

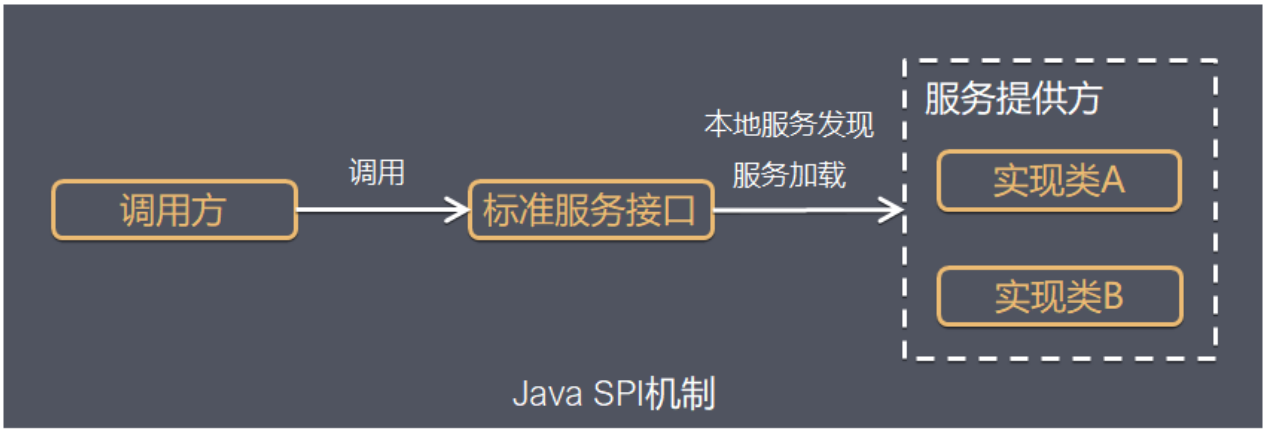

图 1. SPI整体机制图

Java SPI 实际上是“**基于接口的编程＋策略模式＋配置文件**”组合实现的动态加载机制。

系统设计的各个抽象，往往有很多不同的实现方案，在面向的对象的设计里，一般推荐模块之间基于接口编程，模块之间不对实现类进行硬编码。一旦代码里涉及具体的实现类，就违反了可拔插的原则，如果需要替换一种实现，就需要修改代码。为了实现在模块装配的时候能不在程序里动态指明，这就需要一种服务发现机制。 Java SPI就是提供这样的一个机制：为某个接口寻找服务实现的机制。有点类似IOC的思想，就是将装配的控制权移到程序之外，在模块化设计中这个机制尤其重要。所以SPI的核心思想就是**解耦**。

由于本人对服务发现和服务加载只停留在理论理解层面上，没有具体分析代码，所以不继续具体分析。反而是在dubbo-serialization部分中，通过设计接口和不同的实现类的方式，实现了十分便捷的api管理，这样任何使用者都可以便捷地插入自己的序列化方法了。

（2）Singleton（单例模式）

单例模式，是指一个类有且只有自己的一个实例化对象，只提供一个全局访问点，这是为了保证一个类只有一个对象，降低对象之间的耦合度，使得只有一个对象被操作。它的优点如下：

1. 提供了对唯一实例的受控访问；
2. 由于在系统内存中只存在一个对象，因此可以节约系统资源，对于一些需要频繁创建和销毁的对象单例模式无疑可以提高系统的性能；
3. 可以根据实际情况需要，在单例模式的基础上扩展做出双例模式，多例模式；

但它也存在如下缺点：

1. 单例类的职责过重，里面的代码可能会过于复杂，在一定程度上违背了“单一职责原则”。
2. 如果实例化的对象长时间不被利用，会被系统认为是垃圾而被回收，这将导致对象状态的丢失。

但针对Dubbo序列化部分源码而言，我没有看出单例模式的体现，故不继续分析，如果此后有新发现会继续补充。

3.参考资料及总结

在这三篇报告的完成过程中，我参考了很多网上的博客，也略读了《深入理解Apache Dubbo与实战》这本书的第六章RPC部分和第八章的8.3节——Remote层扩展点部分，他们都对我的报告有着很大的帮助，感谢这些作者的分享。主要参考资料链接如下：

1.https://cloud.tencent.com/developer/article/1109453

2.https://www.jianshu.com/p/46b42f7f593c

3.https://www.jianshu.com/p/b8c578b07fbc

在这三篇报告完成的过程中，考虑到本人此前薄弱的Java基础，第一篇主要是初学Java知识以及对dubbo和序列化作了初步的了解，在报告中进行了介绍；第二篇则是将王伟老师课上讲的OOP的主体知识运用到 `serialization` 部分源码的分析中，是这份报告的主体；最后一篇由于自己对Java的进阶知识以及高级分析意图的知识没能很好地消化理解，完成的较为简略，希望自己能在之后不断学习精进Java和OOP知识，继续补充。