# 【第06章-编解码技术】【02-业界主流的编解码框架】

## Google的Protobuf介绍

Protobuf 全称Google Protocol Buffers，它自谷歌开源而来， 在谷歌内部久经考验。它将数据结构以.proto 文件进行描述，通过代码生成工具可以生成对应数据结构的POJO 对象和Protobuf 相关的方法和属性。

它的特点如下。

* 结构化数据存储格式（ X ML, JSON 等）：
* 高效的编解码性能；
* 语言无关、平台无关、扩展性好：
* 官方支持Java 、C++、C#、GO和Python 六种语言。

XML 的可读性和可扩展性非常好，也非常适合描述数据结构，但是XML 解析的时间开销和XML 为了可读性而牺牲的空间开销都非常大， 因此不适合做高性能的通信协议。Protobuf 使用工进制编码， 在空间和性能上具有更大的优势。

Protobuf 另一个比较吸引人的地方就是它的数据描述文件和代码生成机制，利用数据描述文件对数据结构进行说明的优点如下。

* 文本化的数据结构描述语言， 可以实现语言和平台无关，特别适合异构系统间的集成；
* 通过标识字段的顺序，可以实现协议的前向兼容；
* 自动代码生成， 不需要手工编写同样数据结构的其他语言版本；
* 方便后期的管理和维护。相比于代码． 结构化的文档更容易管理和维护。

下Protobuf 编解码和其他几种序列化框架的性能对比数据如图1、图2所示。

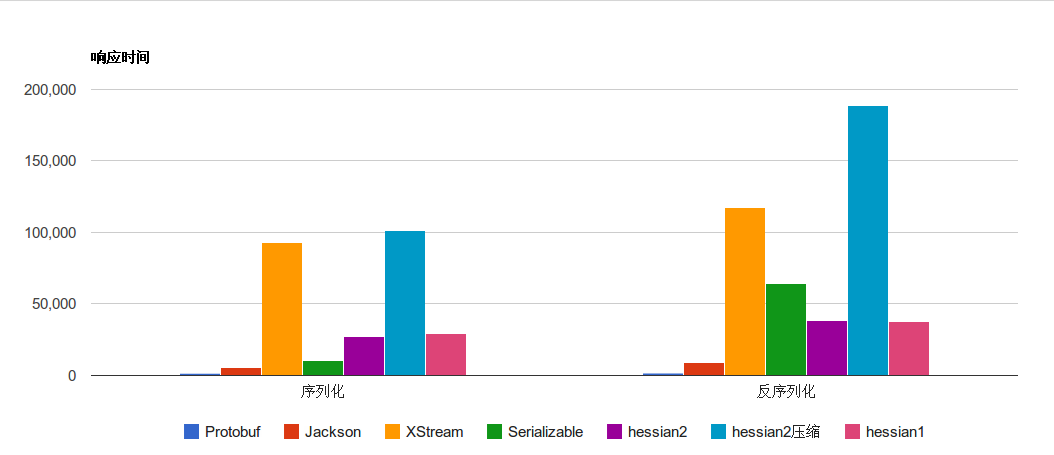


图1 Protobuf 编解码和其他几种序列化枢架的响应时间对比

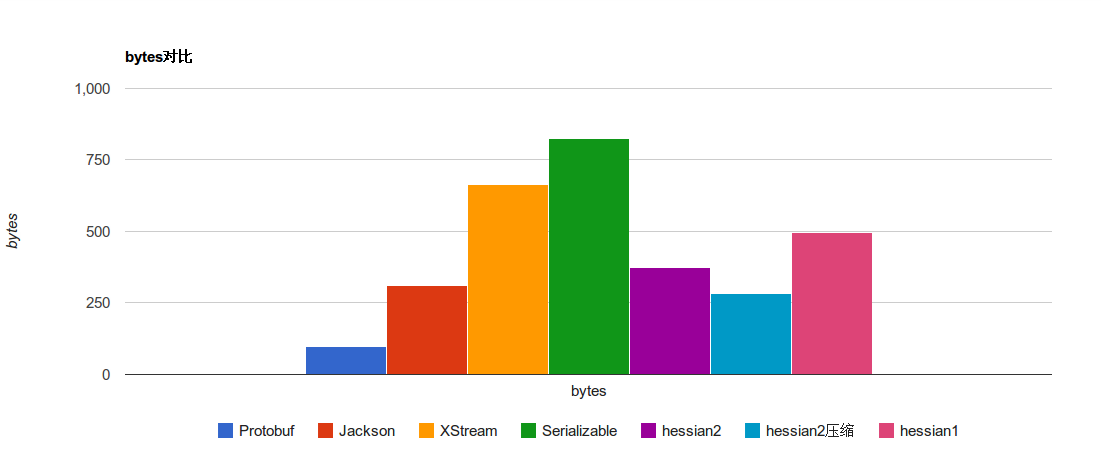


图2 Protobuf 和其他几种序列化拯架的字节数对比

从图中可以发现， Protobuf 的编解码性能远远高于其他几种序列化框架的序列化和反序列化，这也是很多RPC 框架选用Protobuf 做编解码框架的原因。

## Facebook 的Thrift 介绍

Thrift 源于Facebook，Thrift 是为了解决Face book 各系统问大数据量的传输通信以及系统之间语言环境不同需要跨平台的特性，因此Thrift 可以支持多种程序语言。

在多种不同的语言之间通信， Thrift以作为高性能的通信中间件使用，它支持数据（对象）序列化和多种类型的RPC 服务。Thrift适用于静态的数据交换，需要先确定好它的数据结构，当数据结构发生变化时，必须重新编辑IDL 文件， 生成代码和编译。Thrift 适用于搭建大型数据交换及存储的通用工具， 对于大型系统中的内部数据传输，相对于JSON 和XML 在性能和传输大小上都有明显的优势。

Thrift 主要由5 部分组成。

（1）语言系统以及IDL 编译器：负责由用户给定的IDL 文件生成相应语言的接口代码；

（2）TProtocol:：RPC的协议层，可以选择多种不同的对象序列化方式，如JSON和Binary；

（3）TTransport：RPC的传输层，同样可以地择不同的传输层实现，如socket、NIO、MemoryBuffer 等；

（）TProcessor： 作为协议层平和用户提供的服务实现之间的纽带， 负责调用服务实现的接口；

（5）TServer：聚合TProtocol、TTransport 和TProcessor 等对象。

我们关注的是编解码框架，与对应的就是丁Protocol . 由于Thrift的RPC 服务调用和编解码框架绑定在一起， 所以，通常我们使用Thrift的时候会采取RPC 框架的方式。但是，它的TProtocol 编解码框架还是可以以类似的方式独立使用的。

与Protobuf 比较类似的是， Thrift 通过IDL 描述接口和数据结构应义， 它支持8 种Java基本类型、Map 、Set和List ， 支持可选和必选定义，功能非常强大。因为可以定义数据结构中字段的顺序，所以它也可以支持协议的前向兼容。

Thrift 支持三种比较典型的编解码方式。

* 通用的二进制编解码：
* 压缩二进制编解码；
* 优化的可选字段压缩编解码。

由于支持二进制压缩编解码， Thrift 的编解码性能表现也相当优异，远远超过Java 序列化和RMI 等，图3展示了同等测试条件下的编解码耗时信息。

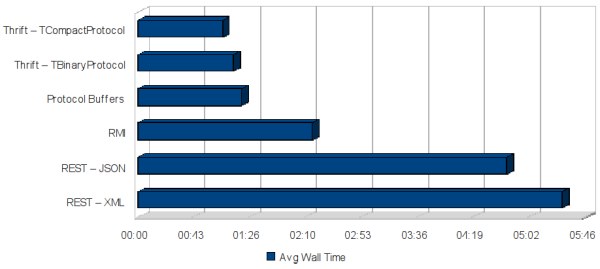


图3 Thrift性能测试对比图

## JBoss Marshalling 介绍

JBoss Marshalling 是一个Java 对象的序列化API 包，修正了JDK 自带的序列化包的很多问题，但又保持跟 java.io.Serializable 接口的兼容。同时，增加了一些可调的参数和附加的恃性， 井且这些参数和特性可通过工厂类进行配置。

相比于传统的Java序列化机制，它的优点如下。

* 可插拔的类解析器，提供更加便捷的类加载定制策略， 通过一个接口即可实现定制：
* 可插拔的对象替换技术，不需要通过继承的方式；
* 可插拔的预定义类缓存表，可以减小序列化的字节数组长度，提升常用类型的对象序列化性能：
* 无须实现java.io.Serializable 接口，即可实现Java 序列化：
* 通过缓存技术提升对象的序列化性能。