复习课（四）| Thrift

2021-11-17 黄金

《大数据经典论文解读》 课程介绍 >

讲述：王惠

时长 08:47 大小 8.05M

你好，我是黄金。这次我们要复习的论文内容，是 Facebook 在 2007 年发表的这篇 Thrift 的论文。

Thrift 介绍

2007 年前后，随着 Facebook 的业务发展，流量激增，服务之间的关系变得越来越复杂，他们的工程师开始尝试使用多种编程语言，来提升服务组合的性能、开发的简易性和速度，以及现有库的可用性，他们试图寻找一种透明的、高效的，并且能够沟通不同编程语言的协议框架。不过最后，Facebook 并没有找到适合自己口味的开源软件，同时其 Protobuf 还处于闭源状态，所以工程师们就开发了 Thrift 这个项目。

论文中提到，在 Facebook 内部，Thrift 作为搜索服务的协议层和传输层，它允许服务端团队使用高效的 C++ 语言、前端团队使用 PHP 语言访问搜索服务，允许运维团队使用 Python 语言获取服务状态信息。另外，Thrift 还能用于记录日志、追踪请求的处理。

那么接下来，我们就一起来具体复习下 Thrift 这个框架。

跨语言

首先，作为一种跨语言的序列化协议框架，Thrift 需要定义好支持的数据类型，以透明地适配不同语言的类型系统。

在论文中提到，Thrift 支持的类型包括了基础类型 bool、byte、i16、i32、i64、double、string，容器类型 list、set、map，以及结构体类型。一个结构体是由基础类型、容器类型和子结构体组合而成的。一个这样的通用类型系统，让使用者可以灵活地定义协议字段，而不用关心如何适配到不同的语言，以及在对应的语言中如何解析该字段。

紧凑的二进制编码

Thrift 采用二进制编码格式。其实在网络中传输数据有很多种选择，包括语言内置的序列化方式，比如 Java 的 Serializable；文本格式，比如 JSON 和 XML；还有二进制格式。

而作为跨语言的序列化协议框架，Thrift 可选的只有文本格式和二进制格式。文本格式可读性强，但是数据大小比二进制格式大很多。通过 GFS、MapReduce 和 Bigtable 论文的学习，我们知道网络带宽往往是大数据系统的瓶颈所在，**越是节约传输流量的方式，在大数据领域就越有竞争力**。

Thrift 有两种不同的二进制编码格式，一种是 BinaryProtocol，另一种是 CompactProtocol。

> **BinaryProtocol 是普通的二进制格式**，在编码一个字段的时候，我们会固定用 1 个字节表示字段类型，2 个字节表示字段编号，接着用一组字节表示值的长度和内容。
>
> **CompactProtocol 是紧凑的二进制格式**，它是用 1 个字节来表示字段类型和编号，低 4 位是字段类型，高 4 位是相对于上一个字段编号的增量。由于 Thrift 的所有类型加起来不到 16 种，所以 4 位就足够表示所有的可能性了。不过，两个相邻的字段编号是可能超过 16 的，碰到这种情况，我们就用 2 个字节分开表示字段类型和编号增量即可。

此外，CompactProtocol 的紧凑还体现在**整型的编码**上，即采用 **ZigZag+VQL** 可变长数值编码。这种编码把整数按 0、-1、1、-2……的方式，正负交替顺序排列，让排在前面的整数用更少的字节来编码，比如 -64 到 63 这 128 个数，用 1 个字节表示就够了。

而在整型的传输中，小数值比大数值出现得更频繁，比如字符串的长度、数组的长度，肯定都是小数值，这种编码方式有助于减少编码后的数据量。

可扩展

Thrift 采用了分层的设计方式提供扩展性。

在 Transport 层，默认通过 TCP/IP 流套接字通信，我们可以在这一层增加逻辑，把 Thrift 请求写到磁盘上，这也是 Facebook 通过 Thrift 记录日志的做法。

在 Protocol 层，默认通过二进制编解码数据，可以修改成 JSON、XML 等其他编解码格式。

在 Processors 层，使用者需要为每一条协议实现处理逻辑，指明逻辑的执行线程。

向前向后兼容

作为服务之间的通信框架，Thrift 的一个重要的能力是**要能够支持服务不断向前演化**。

我们的服务需要不断更新，以便提供新的功能，或者修复存在的问题。服务可能由多个实例构成的集群来提高，升级服务一般采用滚动更新，也就是先更新集群中的几个实例，通过监控观察这几个实例的运行情况，当结果符合预期之后，继续分批更新剩余的实例，直至所有的实例更新完成。

但如果升级服务涉及改动某条通信协议，麻烦的事就出现了。在滚动更新期间，同一条协议存在两个版本，未升级的服务实例提供的是版本 1，升级后的服务实例提供的是版本 2。如果我们改动的是一条请求协议，那么尚未升级的客户端，把老版的请求发给了已经升级的服务端，服务端能解析吗？如果改动的是一条响应协议，已经升级的服务端，把新版的响应发给了尚未升级的客户端，客户端能解析吗？

实际上，要处理这种情况，就需要 Thrift 提供**向前向后兼容**的能力了。所谓向前兼容，就是老代码能读取新代码编码的数据，所谓向后兼容，就是新代码能读取老代码编码的数

据。有了向前兼容的能力，尚未升级的客户端就能解析服务端发来的新版协议，而有了向后兼容的能力，已经升级的服务端就能解析客户端发来的老版协议。

Thrift 通过为每个字段定义了一个编号，并在协议中传输字段类型，来获得向前向后兼容的能力。协议的改动来自两个方面，第一，新增或删除字段，第二，修改字段类型。

Thrift 要求新增字段采用不同编号，当老代码解析字段编号时，发现本地的协议定义文件并不包含这个编号，就能认识到这是一个新增的字段，由于协议的字节序列中传输了字段类型，老代码也能解析出这个新字段。反过来，对于新代码而言，如果收到了老代码发过来的数据，发现某些字段缺失了，补上默认值即可。

如果改变的不是字段的数量，而是字段的类型，那么先按协议的字节序列中，指定的类型解析字段，然后按本地的协议定义文件中声明的类型去转换即可。

小结

好了，到这里 Thrift 的核心内容我们就复习完了。在整个复习课中，我并没有提到 Thrift 的接口定义语言 IDL，因为我相信如果你是服务端工程师，你肯定熟悉 Thrift 或 Protobuf 之类的序列化协议框架。

当我们回看自己熟悉的工具，把它还原到当年研发的背景，以及大数据领域面临的挑战下观察时，我们其实可以获得新的启发。我们能看到，Thrift 为什么要支持跨语言，为什么使用紧凑的二进制编码，为什么要提供向前向后的兼容性，以及它的可扩展设计所带来的灵活性和生命力。

如果你已经阅读了 Thrift 的论文，会发现论文中有一半的内容，在今天看来价值并不大，阅读的时候可以快速跳过这些内容。而徐老师在加餐 1 中就提到了如何读论文，并推荐了吴恩达教授的一个讲座，也是谈如何阅读论文，相信你读完之后会有新的收获，敢于跳过论文中不重要的细节。

上一篇 复习课（三）| Bigtable

下一篇 15 | Hive：来来去去的DSL，永生不死的SQL

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。