12-序列化与反序列化：如何通过网络传输结构化的数据？

你好，我是李玥。

最近有一些同学留言说，感觉这几节课讲的内容和消息关系不大。这里我解释一下，因为我们课程其中的一个目的，是让同学们不仅会使用消息队列，还可以通过对消息队列的学习，在代码能力上有一个提升，具备“造轮子”的能力。这样，你对消息队列的理解才能足够的深入，而不只是浮于表面。如果你细心可能也会发现，很多大厂在面试时，提到消息队列的问题，也不会仅仅局限在消息队列的使用上，他更多的会考察“你为什么这么实现”。

所以在进阶篇的上半部分，我会把开发一个消息队列需要用到的一些底层的关键技术给大家讲解清楚，然后我们再来一起分析消息队列的源代码。

在上节课中，我们解决了如何实现高性能的网络传输的问题。那是不是程序之间就可以通信了呢？这里面还有一些问题需要解决。

我们知道，在TCP的连接上，它传输数据的基本形式就是二进制流，也就是一段一段的1和0。在一般编程语言或者网络框架提供的API中，传输数据的基本形式是字节，也就是Byte。一个字节就是8个二进制位，8个Bit，所以在这里，二进制流和字节流本质上是一样的。

那对于我们编写的程序来说，它需要通过网络传输的数据是什么形式的呢？是结构化的数据，比如，一条命令、一段文本或者是一条消息。对应到我们写的代码中，这些结构化的数据是什么？这些都可以用一个类（Class）或者一个结构体（Struct）来表示。

那显然，**要想使用网络框架的API来传输结构化的数据，必须得先实现结构化的数据与字节流之间的双向转换。**这种将结构化数据转换成字节流的过程，我们称为序列化，反过来转换，就是反序列化。

序列化的用途除了用于在网络上传输数据以外，另外的一个重要用途是，将结构化数据保存在文件中，因为在文件内保存数据的形式也是二进制序列，和网络传输过程中的数据是一样的，所以序列化同样适用于将结构化数据保存在文件中。

很多处理海量数据的场景中，都需要将对象序列化后，把它们暂时从内存转移到磁盘中，等需要用的时候，再把数据从磁盘中读取出来，反序列化成对象来使用，这样不仅可以长期保存不丢失数据，而且可以节省有限的内存空间。

这节课，我们就来聊聊，怎么来实现高性能的序列化和反序列化。

你该选择哪种序列化实现？

如果说，只是实现序列化和反序列的功能，并不难，方法也有很多，比如我们最常使用的，把一个对象转换成字符串并打印出来，这其实就是一种序列化的实现，这个字符串只要转成字节序列，就可以在网络上传输或者保存在文件中了。

但是，你千万不要在你的程序中这么用，这种实现的方式仅仅只是能用而已，绝不是一个好的选择。

有很多通用的序列化实现，我们可以直接拿来使用。Java和Go语言都内置了序列化实现，也有一些流行的开源序列化实现，比如，Google 的Protobuf、Kryo、Hessian等；此外，像JSON、XML这些标准的数据格

式，也可以作为一种序列化实现来使用。

当然，我们也可以自己来实现私有的序列化实现。

面对这么多种序列化实现，我们该如何选择呢？你需要权衡这样几个因素：

1. 序列化后的数据最好是易于人类阅读的；
2. 实现的复杂度是否足够低；
3. 序列化和反序列化的速度越快越好；
4. 序列化后的信息密度越大越好，也就是说，同样的一个结构化数据，序列化之后占用的存储空间越小越好；

当然，**不会存在一种序列化实现在这四个方面都是最优的**，否则我们就没必要来纠结到底选择哪种实现了。因为，大多数情况下，易于阅读和信息密度是矛盾的，实现的复杂度和性能也是互相矛盾的。所以，我们需要根据所实现的业务，来选择合适的序列化实现。

像JSON、XML这些序列化方法，可读性最好，但信息密度也最低。像Kryo、Hessian这些通用的二进制序列化实现，适用范围广，使用简单，性能比JSON、XML要好一些，但是肯定不如专用的序列化实现。

对于一些强业务类系统，比如说电商类、社交类的应用系统，这些系统的特点是，业务复杂，需求变化快，但是对性能的要求没有那么苛刻。这种情况下，我推荐你使用JSON这种实现简单，数据可读性好的序列化实现，这种实现使用起来非常简单，序列化后的JSON数据我们都可以看得懂，无论是接口调试还是排查问题都非常方便。付出的代价就是多一点点CPU时间和存储空间而已。

比如我们要序列化一个User对象，它包含3个属性，姓名zhangsan，年龄：23，婚姻状况：已婚。

```
User:
  name: "zhangsan"
  age: 23
  married: true
```

使用JSON序列化后：

```
{"name":"zhangsan","age":"23","married":"true"}
```

这里面的数据我们不需要借助工具，是直接可以看懂的。

序列化的代码也比较简单，直接调用JSON序列化框架提供的方法就可以了：

```
byte [] serializedUser = JsonConvert.SerializeObject(user).getBytes("UTF-8");
```

如果JSON序列化的性能达不到你系统的要求，可以采用性能更好的二进制序列化实现，实现的复杂度和JSON序列化是差不多的，都很简单，但是序列化性能更好，信息密度也更高，代价就是失去了可读性。

比如我们用Kryo来序列化User对象，它的代码如下：

```
kryo.register(User.class);
Output output = new Output(new FileOutputStream("file.bin"));
kryo.writeObject(output, user);
```

在这段代码里，先要向Kryo注册一下User类，然后创建一个流，最后调用writeObject方法，将user对象序列化后直接写到流中。这个过程也是非常简单的。

实现高性能的序列化和反序列化

绝大部分系统，使用上面这两类通用的序列化实现都可以满足需求，而像消息队列这种用于解决通信问题的中间件，它对性能要求非常高，通用的序列化实现达不到性能要求，所以，很多的消息队列都选择自己实现高性能的专用序列化和反序列化。

使用专用的序列化方法，可以提高序列化性能，并有效减小序列化后的字节长度。

在专用的序列化方法中，不必考虑通用性。比如，我们可以固定字段的顺序，这样在序列化后的字节里面就不必包含字段名，只要字段值就可以了，不同类型的数据也可以做针对性的优化：

对于同样的User对象，我们可以把它序列化成这样：

```
03   | 08 7a 68 61 6e 67 73 61 6e | 17 | 01
User |    z  h  a  n  g  s  a  n  | 23 | true
```

我解释一下，这个序列化方法是怎么表示User对象的。

首先我们需要标识一下这个对象的类型，这里面我们用一个字节来表示类型，比如用03表示这是一个User类型的对象。

我们约定，按照name、age、married这个固定顺序来序列化这三个属性。按照顺序，第一个字段是name，我们不存字段名，直接存字段值“zhangsan”就可以了，由于名字的长度不固定，我们用第一个字节08表示这个名字的长度是8个字节，后面的8个字节就是zhangsan。

第二个字段是年龄，我们直接用一个字节表示就可以了，23的16进制是17 。

最后一个字段是婚姻状态，我们用一个字节来表示，01表示已婚，00表示未婚，这里面保存一个01。

可以看到，同样的一个User对象，JSON序列化后需要47个字节，这里只要12个字节就够了。

专用的序列化方法显然更高效，序列化出来的字节更少，在网络传输过程中的速度也更快。但缺点是，需要为每种对象类型定义专门的序列化和反序列化方法，实现起来太复杂了，大部分情况下是不划算的。

小结

进程之间要通过网络传输结构化的数据，需要通过序列化和反序列化来实现结构化数据和二进制数据的双向转换。在选择序列化实现的时候，需要综合考虑数据可读性，实现复杂度，性能和信息密度这四个因素。

大多数情况下，选择一个高性能的通用序列化框架都可以满足要求，在性能可以满足需求的前提下，推荐优先选择JSON这种可读性好的序列化方法。

如果说我们需要超高的性能，或者是带宽有限的情况下，可以使用专用的序列化方法，来提升序列化性能，节省传输流量。不过实现起来很复杂，大部分情况下并不划算。

思考题

课后，你可以想一下这个问题：在内存里存放的任何数据，它最基础的存储单元也是二进制比特，也就是说，我们应用程序操作的对象，它在内存中也是使用二进制存储的，既然都是二进制，为什么不能直接把内存中，对象对应的二进制数据直接通过网络发送出去，或者保存在文件中呢？为什么还需要序列化和反序列化呢？欢迎在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言：

- a、 2019-08-17 06:18:06
  1.因为应用程序里的对象，除了属性和属性值以外，还有一些其他的信息，比如jdk编译的版本，类的全限定名，类继承的父类和实现的接口等信息。如果服务端是jdk1.8编译的对象，发给客户端，客户端用的是jdk1.7，肯定会报错。
  2.这些其他的信息是多余的，传输中会增加网络负担 [8赞]

- 许童童 2019-08-17 11:57:59
  内存在每个平台的分布都是不一样的，一个对象不光有用户定义的属性，还包括平台定义的属性，如果不经过序列化就传输过去，一方面会浪费大量的带宽，另一方面还可能因为平台不同等问题导致不兼容，从而无法解析。 [3赞]

- ly 2019-08-19 09:16:19
  个人对今天的内容进行简单的描述：
  1. 序列化：是一种规则，它定义了数据表达的规则；
  2. 反序列化：依靠给定的规则，还原数据。
  3. 今天的问题：
  内存中的对象数据应该具有语言独特性，例如表达相同业务的User对象(id/name/age字段),Java和PHP在内存中的数据格式应该不一样的，如果直接用内存中的数据，可能会造成语言不通。通常两个服务之间没有严格要求语言必须一致，只要对序列化的数据格式进行了协商，任何2个语言直接都可以进行序列化传输、接收。 [2赞]

  作者回复2019-08-19 09:29:40
  👍👍👍

- humor 2019-08-19 23:46:49
  内存中的对象并不是独立存在的，他们会有各种引用关系，直接存储的话，可能存储的很多都是内存地址吧

- leslie 2019-08-19 23:04:32
  打卡：没太想明白，不过问题记在大脑里总归会想出点东西。

- oscarwin 2019-08-19 09:55:39
  虽然都是二进制的数据，但是序列化的二进制数据是通过一定的协议将数据字段进行拼接。第一个优势是：不同的语言都可以遵循这种协议进行解析，实现了跨语言。第二个优势是：这种数据可以直接持久化到磁盘，从磁盘读取后也可以通过这个协议解析出来。如果是内存中的数据不能直接存盘的，直接存盘后再读出来我们根本无法辨识这是个什么数据。

- 😊 2019-08-19 07:59:50
  内存数据要搭配元数据定义，这个定义在不同语言平台等兼容非常难通用。拿内存本身也不是特别容易的事情

- whhbbq 2019-08-18 20:30:39
  服务端和客户端可能是不同平台，同样的数据在内存中的二进制比特是不一样的。通过序列化和反序列化实现系统通信的跨平台。

- 深藏Blue 2019-08-18 20:09:27
  老师好，寻求个帮助。我遇到个这样的需求：一个c/s的架构应用，需要实现client之间的点对点数据通信以及群组通信 实际上就是一个即时通讯应用 由于没有即时通讯相关的经验 还请老师能够指导一下。其中的数据传输使用MQ转发 还是基于netty自定义？自己两种方式都琢磨了一下，基于MQ的话，topic tag会很多 只要涉及一端client 操作（比如：打开某个界面）需要同步到其他client的话 就需要对topic进行生产以及订阅消费。 第二就是基于netty自定义，这种情况下c端和s端都要定义一个类似servlet或者spring mvc里面的dispatcher根据相关参数分发到具体的业务方法。这方面我是小白，还请老师以及知道的朋友给予指点

  作者回复2019-08-18 21:12:25
  一般来说，即时通信类系统并不适合用消息队列来实现。很多即时通讯软件都是使用一些P2P技术，数据

直接点对点传输，不经过服务端转发的。

- A9 2019-08-18 17:10:33
  即使只传输数据的内存结构，对于不同版本的代码，每种编程语言的不同版本，不同语言，都会导致兼容问题。序列化处理可以避开这些问题，保证程序的长期稳定运行，也符合中间件的使用方式

- 羽球 2019-08-18 08:40:12
  本地应用通过基地址+偏移量的方式访问内存中的成员变量，因为它知道偏移量代表什么。
  通过网络发给对方的话，对方并不知道偏移量代表什么，所以无法解析。

- Jian 2019-08-18 06:45:27
  本机的操作系统或虚拟机管理着存于内存中的数据。
  通过网络传输或是文件转存的方式，其目的是给下一个系统使用，那么这里就需要一个协议完成两个系统的数据交换。
  内存中的数据基本是瞬时态，如果只是传送内存中的部分数据话，数据如何存储呢？数据属性可能是分布在内存中的非连续地址，属性中间的非目标数据该如何处理？具体该如何传输呢？
  举个例子，系统A的内存中存储着所有的汉子，我跟系统A说，”请给我三百首唐诗“，结果我收到的是几千个散列的汉字。。。
  （不知道这个例子是否恰当）

- 张学磊 2019-08-17 22:50:24
  举一个例子，如果直接取LinkedList在内中的字节流作为序列化结果，那么只能包含一个头节点和一个子节点，必须使用LinkedList自己实现的writeObject方法进行序列化

- 游弋云端 2019-08-17 11:07:25
  需要面临的问题：
  1、网络字节序与主机字节序问题，业务要感知和处理大小端问题；
  2、平台差异，各平台对基本数据类型的长度定义不一致、结构体对齐策略不一致，无法实现平台兼容；
  3、连续内存问题，一个对象可能引用，指向其他对象，指针就是一个地址，传输后在另外的设备上是无效值；

  如果解决这些问题了，也变相的实现了自己的序列化框架了。

- 石维康 2019-08-17 10:54:53
  还有一个因素是每台机器的大小端可能不同，会对内存中的数据有不同的解释。

- 张三 2019-08-17 09:59:12
  一直对序列化反序列化的概念很含糊，今天有了进步，公司用的就是专用的序列化实现，有时候需要通过约定好的文档去查看每一位的意义。当然也学到了这种专用的实现的代价。

- Geek_0cf83d 2019-08-17 09:42:29
  数据在内存中大多以地址链接的离散数据为主，且不同平台有大小端问题

- 马成 2019-08-17 09:19:11
  因为内存里的对象不是一串连续的字节流，而是通过地址相互引用，比如map，其值是一个地址，表示值在哪里，而不是值本身

- 飞翔 2019-08-17 00:37:58
  最后真是个好问题 就是不知道答案 我还以为在内存中存的是对象呢， 如果内存中存的是二进制 那谁吧二

进制换成对象的?