二

05 深入分析 Jute 的底层实现原理

上个课时我们讲解了 ZooKeeper 中采用 Jute 作为序列化解决的方案，并介绍了其应用层的使用技巧。本课时我们就深入 Jute 框架的内部核心，来看一看其内部的实现原理和算法。而通过研究 Jute 序列化框架的内部的实现原理，能够让我们在日常工作中更加高效安全地使用 Jute 序列化框架。

简述 Jute 序列化

通过前面的课时我们知道了序列化就是将 Java 对象转化成字节码的形式，从而方便进行网络传输和本地化存储，那么具体的序列化方法都有哪些呢？这里我们结合 ZooKeeper 中使用到的序列化解决方案 Jute 来进行介绍，Jute 框架给出了 3 种序列化方式，分别是 Binary 方式、Csv 方式、XML 方式。序列化方式可以通俗地理解成我们将 Java 对象通过转化成特定的格式，从而更加方便在网络中传输和本地化存储。之所以采用这 3 种方式的格式化文件，也是因为这 3 种方式具有跨平台和普遍的规约特性，后面我将会对这三种方法的特性进行具体讲解。接下来我将深入 Jute 的底层，看一下这 3 种实现方式的底层实现过程。

Jute 内部核心算法

上个课时中我们提到过，ZooKeeper 在实现序列化的时候要实现 Record 接口，而在 Record 接口的内部，真正起作用的是两个工具类，分别是 OutPutArchive 和 InputArchive。下边我们分别来看一下它们在 Jute 内部是如何实现的。

OutPutArchive 是一个接口，规定了一系列序列化相关的操作。而要实现具体的相关操作，Jute 是通过三个具体实现类分别实现了 Binary、Csv、XML 三种方式的序列化操作。而这三种方式有什么不同，我们在日常工作中应该如何选择呢？带着这些问题我们来深入到 Jute 的内部实现来找寻答案

Binary 方式的序列化

首先我们来看一下 Jute 中的第 1 种序列化方式：Binary 序列化方式，即二进制的序列化方式。正如我们前边所提到的，采用这种方式的序列化就是将 Java 对象信息转化成二进制的文件格式。

在 Jute 中实现 Binary 序列化方式的类是 BinaryOutputArchive。该 BinaryOutputArchive 类通过实现 OutPutArchive 接口，在 Jute 框架采用二进制的方式实现序列化的时候，采用其作为具体的实现类。

在这里我们通过调用 Record 接口中的 writeString 方法为例，该方法是将 Java 对象的 String 字符类型进行序列化。当调用 writeString 方法后，首先判断所要进行序列化的字符串是否为空。如果是空字符串则采用 writeInt 方法，将空字符串当作值为 -1 的数字类型进行序列化；如果不为空，则调用 stringtoByteBuffer 方法对字符串进行序列化操作。

```
void writeString (String s, Sring tag){
  if(s==null){
    writeInt(-1,"len");
    return
  }
  ByteBuffer bb = stringtoByteBuffer(s);
  ...
}
```

而 stringToByteBuffer 方法也是 BinaryOutputArchive 类的内部核心方法，除了 writeString 序列化方法外，其他的比如 writeInt、wirteDoule 等序列化方法则是调用 DataOutPut 接口中的相关方法来实现具体的序列化操作。

在调用 BinaryOutputArchive 类的 stringToByteBuffer 方法时，在将字符串转化成二进制字节流的过程中，首选将字符串转化成字符数组 CharSequence 对象，并根据 ascii 编码判断字符类型，如果是字母等则使用1个 byte 进行存储。如果是诸如 “¥” 等符号则采用两个 byte 进程存储。如果是汉字则采用3个 byte 进行存储。

```
...
private ByteBuffer bb = ByteBuffer.allocate(1024);
...
ByteBuffer stringToByteBuffer(CharSeuquece s){
    if (c < 0x80) {
                bb.put((byte) c);
    } else if (c < 0x800) {
```

```
            bb.put((byte) (0xc0 | (c >> 6)));
            bb.put((byte) (0x80 | (c & 0x3f)));
        } else {
            bb.put((byte) (0xe0 | (c >> 12)));
            bb.put((byte) (0x80 | ((c >> 6) & 0x3f)));
            bb.put((byte) (0x80 | (c & 0x3f)));
        }
    }
```

Binary 二进制序列化方式的底层实现相对简单，只是采用将对应的 Java 对象转化成二进制字节流的方式。Binary 方式序列化的优点有很多：无论是 Windows 操作系统、还是 Linux 操作系统或者是苹果的 macOS 操作系统，其底层都是对二进制文件进行操作，而且所有的系统对二进制文件的编译与解析也是一样的，所有操作系统都能对二进制文件进行操作，跨平台的支持性更好。而缺点则是会存在不同操作系统下，产生大端小端的问题。

XML 方式的序列化

说完了 Binary 的序列化方式，我们再来看看 Jute 中的另一种序列化方式 XML 方式。XML 是一种可扩展的标记语言。当初设计的目的就是用来传输和存储数据，很像我们都很熟悉的 HTML 语言，而与 HTML 语言不同的是我们需要自己定义标签。在 XML 文件中每个标签都是我们自己定义的，而每个标签就对应一项内容。一个简单的 XML 的格式如下面这段代码所示：

```
<note>
<to>学生</to>
<from>老师</form>
<heading>上课提醒</heading>
<body>记得9:00来上课</body>
</note>
```

大概了解了 XML 文件，接下来我们看一下 Jute 框架中是如何采用 XML 方式进行序列化操作的。在 Jute 中使用 XmlOutPutArchive 类作用 XML 方式序列化的具体实现类。与上面讲解二进制的序列化实现一样，这里我们还是以 writeString 方法的 XML 序列化方式的实现为例。首先，当采用XML 方式进行序列化时，调用 writeString 方法将 Java 中的 String 字

符串对象进行序列化时，在 writeString 内部首先调用 printBeginEnvelope 方法并传入 tag 参数，标记我们要序列化的字段名称。之后采用“”和“”作用自定义标签，封装好传入的 Java 字符串。

```
void writeString(String s, String tag){
   printBeginEnvelope(tag);
   stream.print("<string>");
   stream.print(Utils.toXMLString(s));
   stream.print("</string>");
   printEndEnvelope(tag);
}
```

而在 printBeginEnvelope 方法中，其主要作用就是添加该字段的名称、字段值等信息，用于之后反序列化的过程中。

```
void printBeginEnvelope (String tag){
  ...
  if ("struct".equals(s)) {
        putIndent();
        stream.print("<member>\n");
        addIndent();
        putIndent();
        stream.print("<name>"+tag+"</name>\n");
        putIndent();
        stream.print("<value>");
    } else if ("vector".equals(s)) {
        stream.print("<value>");
    } else if ("map".equals(s)) {
        stream.print("<value>");
    }
} else {
```

```
    stream.print("<value>");
}
}
```

通过上面在 Jute 框架中，对采用 XML 方式序列化的实现类：XmlOutPutArchive 中的底层实现过程分析，我们可以了解到其实现的基本原理，也就是根据 XML 格式的要求，解析传入的序列化参数，并将参数按照 Jute 定义好的格式，采用设定好的默认标签封装成对应的序列化文件。

而采用 XML 方式进行序列化的优点则是，通过可扩展标记协议，不同平台或操作系统对序列化和反序列化的方式都是一样的，不存在因为平台不同而产生的差异性，也不会出现如 Binary 二进制序列化方法中产生的大小端的问题。而缺点则是序列化和反序列化的性能不如二进制方式。在序列化后产生的文件相比与二进制方式，同样的信息所产生的文件更大。

Csv 方式的序列化

最后我们来学习一下 Jute 序列化框架的最后一种序列化方式：Csv，它和 XML 方式很像，只是所采用的转化格式不同，Csv 格式采用逗号将文本进行分割，我们日常使用中最常用的 Csv 格式文件就是 Excel 文件。

在 Jute 框架中实现 Csv 序列化的类是 CsvOutputArchive，我们还是以 String 字符对象序列化为例，在调用 CsvOutputArchive 的 writeString 方法时，writeString 方法首先调用 printCommaUnlessFirst 方法生成一个逗号分隔符，之后将要序列化的字符串值转换成 CSV 编码格式追加到字节数组中。

```
void writeString(String s, String tag){
  printCommaUnlessFirst();
  stream.print(Utils.toCSVString(s));
  throwExceptionOnError(tag);
}
```

到这里我们已经对 Jute 框架的 3 种序列化方式的底层实现有了一个整体了解，这 3 种方式相比，二进制底层的实现方式最为简单，性能也最好。而 XML 作为可扩展的标记语言跨平台性更强。而 CSV 方式介于两者之间实现起来也相比 XML 格式更加简单。

结束

本课时我们对 Jute 序列化的底层实现过程做了进一步地深度分析，和开篇中提到的一样，我们之所以深入底层学习 Jute 的底层实现原理，是为了在日常开发中更好的使用 ZooKeeper，在发生诸如客户端与服务端通信中信息发送不完整或解析错误等情况时，能通过底层分析从序列化模块入手排查信息错误的原因，进一步提高我们在日常设计和解决问题的能力。

接下来请你思考这样一个问题，在 ZooKeeper 中，默认的序列化实现方式是哪种？

在 ZooKeeper 中默认的序列化实现方式是 Binary 二进制方式。这是因为二进制具有更好的性能，以及大多数平台对二进制的实现都不尽相同。

上一页 下一页