Tair 中的类型转换

xuyun@taobao.com

November 16, 2011

Contents

1 支持的数据类型

Tair 客户端和服务器端之间通信是通过 Mina 进行的。Mina 的数据传输是基于 New IO 的。Java 内置的各种 buffer 为 NIO 的数据读写提供了便利。Tair 的网络通信方案决定了服务器和客户端必须有统一的数据类型，这种数据类型是基于 Byte Buffer 的。数据由客户端传向服务端进行存储时，各种 Java 的数据对象首先需要转换成 Byte 数组，反之，服务器推送给客户端的 Byte 数组集合又需要通过一定的规则来解码成 Java 自身的数据对象。Tair 客户端提供了一个叫做 TranscoderUtil 的工具进行 Java 数据对象的编码和译码。Tair 客户端初始化的时候，做的第一件事情就是初始化 Transcoder 实例。目前 Tair 支持大部分的 Java 数据对象类型。具体如下：

- INT
- STRING
- BOOL
- LONG
- DATE
- BYTE
- FLOAT
- DOUBLE
- BYTEARRAY
- SERIALIZE
- INCDATA

2 编码和译码

2.1 LONG 型

长整型是 Java 中占字节较长的基本数据类型。Long 类型的编码和译码可以为 Int 型提供参考。

编码 Long 型占 64 bit，8 个 Byte。因此需要一个大小为 8 的字节数组 (Byte[8]) 来存放。对 64 位的 Long 型以 8 bit 为单位进行物理切割。代码如 Lst. 1 所示。

```
1  public static byte[] encodeLong(long number) {
2          byte[] rt = new byte[8];
3
4          rt[7] = (byte) (number & 0xFF);
5          rt[6] = (byte) ((number >> 8) & 0xFF);
6          rt[5] = (byte) ((number >> 16) & 0xFF);
7          rt[4] = (byte) ((number >> 24) & 0xFF);
8          rt[3] = (byte) ((number >> 32) & 0xFF);
9          rt[2] = (byte) ((number >> 40) & 0xFF);
10         rt[1] = (byte) ((number >> 48) & 0xFF);
11         rt[0] = (byte) ((number >> 56) & 0xFF);
12         return rt;
13 }
```

Listing 1: encodeLong.java

Table 1: Byte Array of Long

数组索引	存放内容
Byte[1]	1111 1111
Byte[2]	1111 1111
Byte[3]	1111 1111
Byte[4]	1111 1111
Byte[5]	1111 1111
Byte[6]	1111 1111
Byte[7]	1111 1111
Byte[8]	1111 1110

以长整型 -2 为例，编码后的字节数组如表 1 所示。

解码 Long 型的解码是编码的逆过程，这个过程还是需要一点小的技巧的。不能简单地认为是从 Byte 数组中把数据取出进行简单地拼装就可以了。就像钉子钉到墙上后再从墙体拔出，墙上会留下一个洞洞。先来看下 Long 型解码的代码。

如代码 2 所示，对于 byte 型数据，如果为正，只需要从字节数组中把取出进行简单封装即可。但是对于小于 0 的 byte 数据则需要加上 256。

```
1 public static long decodeLong(byte[] data) {
2         long rv = 0;
3         for (byte i : data) {
4             rv = (rv << 8) | ((i < 0) ? (256 + i)
5                                       : i);
6         }
7         return rv;
8 }
```

Listing 2: decodeLong.java

之所以这么做的原因是：对于 byte 型，如果向上扩展成 Long 型，它的符号位为填充到高位。举例来看，对于表 1 中的 Byte[8], 如果转成 Long 型则变成

1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1110

带着符号位的数字再继续进行按位与的操作时就不能复原原来的 Long 型了。

加上 Int 型 256 的效果是消除被扩展的符号位。然数据变回原来的真实面目，256 二进制表示成

1 0000 0000

加上 256 就类似多米诺骨牌，将被 byte 型被扩展的高位的符号位变成 0, 这样再继续进行按位与的操作时就能保解码的正确。对于 1 中的 Byte[8]，加上 256 后的二进制表示如下：

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1111 1110

2.2 INT 型

Int 型的编码和解码与 Long 型的原理类似，只不过 Int 型的数据位数是 32 bit，而对应的 Byte 数组的大小为 4，即 Byte[4]。编码和解码的代码如代码 3 所示。

```
1  public static byte[] encodeInt(int number) {
2          byte[] fg = new byte[4];
3
4          fg[3] = (byte) (number & 0xFF);
5          fg[2] = (byte) ((number >> 8) & 0xFF);
6          fg[1] = (byte) ((number >> 16) & 0xFF);
7          fg[0] = (byte) ((number >> 24) & 0xFF);
8          return fg;
9      }
10
11 public static int decodeInt(byte[] data) {
12     assert data.length <= 4 : "Too long to be an int (" + data.length + ")
           bytes";
13         return (int) decodeLong(data);
14 }
```

Listing 3: EncodeAndDecodeInt.java

2.3 BYTE 型

Byte 型是 Tair 的基本原子单位。只需要把 byte 型放入 Byte 数组中即可，这也意味着该数组的大小永远为 1，即 Byte[1]。编码和解码的代码如代码 4 所示。

```
1  public static byte[] encodeByte(byte in) {
2    return new byte[] {in};
3  }
4  public static byte decodeByte(byte[] in) {
```

```
5   assert in.length <= 1 : "Too long for a byte";
6   byte rv = 0;
7   if (in.length == 1) {
8     rv = in[0];
9   }
10  return rv;
11 }
```

Listing 4: EncodeAndDecodeByte.java

对于 Byte 数组型，Tair 的编码和解码就是为它而生的。因为编码和解码就是以 Byte 数组为基础的。此时，只需要直接存入和取出即可。

2.4 BOOL 型

Bool 型有两个值需要存储，即 True 和 False，在编码的过程中 True 用字符 '1' 的 ASCII 表示，即 0011 0001, 反之 False 用 0011 0000 表示。数组的大小永远为 1，即 Byte[1]。编码和解码的代码如代码 5 所示。

```
1  public static byte[] encodeBoolean(boolean b) {
2   byte[] rv = new byte[1];
3
4   rv[0] = (byte) (b ? '1'
5             : '0');
6   return rv;
7     }
8
9 public static boolean decodeBoolean(byte[] in) {
10  assert in.length == 1 : "Wrong length for a boolean";
11  return in[0] == '1';
12 }
```

Listing 5: EncodeAndDecodeBool.java

2.5 STRING 型

对于 String 型的数据，Java 的 String 类型提供了 getBytes(charset) 的方法来把 Sting 型转换成 Byte 数组，在 String 编码的过程中可以利用该方法，而对于解码则可以利用 String 的构造函数来巧妙生成字符串。String 编码和解码的代码如代码 6 所示。

```
1 /**
2  * Encode a string into the current character set.
3  */
4 public static byte[] encodeString(String in, String charset) {
5   byte[] rv = null;
6
7   try {
```

```
8       rv = in.getBytes(charset);
9     } catch (Exception e) {
10      throw new RuntimeException(e);
11    }
12
13    return rv;
14  }
15
16  public static String decodeString(byte[] data, String charset) {
17          String rv = null;
18
19          try {
20              if (data != null) {
21                  rv = new String(data, charset);
22              }
23          } catch (Exception e) {
24              throw new RuntimeException(e);
25          }
26
27          return rv;
28      }
```

Listing 6: EncodeAndDecodeString.java

代码中的 charset 在 Tair 中默认成 UTF-8。

2.6 类 Long 型

之所以称之为类 Long 型，是因为这些数据类型可以转换成 Long 型或者更短的 Int 型来进行编码和解码。这些类型有 DATE、Float、Double。Date 可以通过 getTime() 方法来转换成 Long 型，同时通过 new Date(long time) 的构造函数来生成 Date 型数据。Float 通过 floatToRawIntBits 方法转换成 Int 型，反之通过 intBitsToFloat 复原成 Float 型。类似地，Double 通过 doubleToRawLongBits 方法转换成 Long 型，并通过 longBitsToDouble 方法复原成 Double 型。

2.7 Object 型

如果数据类型不在以上的各种情况中，那么该数据类型可以当做 Object 来处理。Object 的编码和解码主要运用了流的概念来进行数据的读写。这对于 Java 语言相当简单，只要 Java 中的 object 类型实现了 serializable 接口，Tair 就能做到编码和解码。代码如 7 所示。

```
1  public static byte[] serialize(Object o) {
2          if (o == null) {
3              throw new NullPointerException("Can't serialize null");
```

```
4         }
5
6         byte[] rv = null;
7
8         try {
9             ByteArrayOutputStream bos = new ByteArrayOutputStream();
10            ObjectOutputStream    os  = new ObjectOutputStream(bos);
11
12            os.writeObject(o);
13            os.close();
14            bos.close();
15            rv = bos.toByteArray();
16        } catch (IOException e) {
17            throw new IllegalArgumentException("Non-serializable object", e)
                  ;
18        }
19
20        return rv;
21    }
22
23 public static Object deserialize(byte[] in) {
24   Object rv = null;
25
26   try {
27     if (in != null) {
28       ByteArrayInputStream bis = new ByteArrayInputStream(in);
29       ObjectInputStream    is  = new ObjectInputStream(bis);
30
31       rv                       = is.readObject();
32       is.close();
33       bis.close();
34     }
35   } catch (Exception e) {
36     throw new RuntimeException("deserialize failed", e);
37   }
38
39   return rv;
40 }
```

Listing 7: EncodeAndDecodeObject.java

3 结论

磨刀不误砍柴工，数据类型的解码和编码作为 Tair 初始化时的第一项工作，为后续数据向分布式的 Tair 服务器进行读写做好了统一的数据格式准备。