**[Netty In Action中文版 - 第十四章：实现自定义的编码解码器](http://blog.csdn.net/tongwaccs/article/details/39298731)**

本章讲述Netty中如何轻松实现定制的编解码器，由于Netty架构的灵活性，这些编解码器易于重用和测试。为了更容易实现，使用Memcached作为协议例子是因为它更方便我们实现。

        Memcached是免费开源、高性能、分布式的内存对象缓存系统，其目的是加速动态Web应用程序的响应，减轻**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "_blank)**负载；Memcache实际上是一个以key-value存储任意数据的内存小块。可能有人会问“为什么使用Memcached？”，因为Memcached协议非常简单，便于讲解。

**14.1 编解码器的范围**

        我们将只实现Memcached协议的一个子集，这足够我们进行添加、检索、删除对象；在Memcached中是通过执行SET,GET,DELETE命令来实现的。Memcached支持很多其他的命令，但我们只使用其中三个命令，简单的东西，我们才会理解的更清楚。

        Memcached有一个二进制和纯文本协议，它们都可以用来与Memcached服务器通信，使用什么类型的协议取决于服务器支持哪些协议。本章主要关注实现二进制协议，因为二进制在网络编程中最常用。

**14.2 实现Memcached的编解码器**

        当想要实现一个给定协议的编解码器，我们应该花一些事件来了解它的运作原理。通常情况下，协议本身都有一些详细的记录。在这里你会发现多少细节？幸运的是Memcached的二进制协议可以很好的扩展。

        在RFC中有相应的规范，并提供了Memcached二进制协议下载地址：http://code.google.com/p/memcached/wiki/BinaryProtocolRevamped。我们不会执行Memcached的所有命令，只会执行三种操作：SET,GET和DELETE。这样做事为了让事情变得简单。

**14.3 了解Memcached二进制协议**

        可以在http://code.google.com/p/memcached/wiki/BinaryProtocolRevamped上详细了解Memcached二进制协议结构。不过这个网站如果不翻墙的话好像访问不了。

**14.4 Netty编码器和解码器**

**14.4.1 实现Memcached编码器**

        先定义memcached操作码(Opcode)和响应状态码(Status)：

**[java]** [view plaincopy](http://blog.csdn.net/abc_key/article/details/38388031)[在CODE上查看代码片](https://code.csdn.net/snippets/443257)

1. **package** netty.in.action.mem;
3. /\*\*
4. \* memcached operation codes
5. \* @author c.king
6. \*
7. \*/
8. **public** **class** Opcode {
10. **public** **static** **final** **byte** GET = 0x00;
11. **public** **static** **final** **byte** SET = 0x01;
12. **public** **static** **final** **byte** DELETE = 0x04;
14. }

**[java]** [view plaincopy](http://blog.csdn.net/abc_key/article/details/38388031)[在CODE上查看代码片](https://code.csdn.net/snippets/443257)

1. **package** netty.in.action.mem;
3. /\*\*
4. \* memcached response statuses
5. \* @author c.king
6. \*
7. \*/
8. **public** **class** Status {
10. **public** **static** **final** **short** NO\_ERROR = 0x0000;
11. **public** **static** **final** **short** KEY\_NOT\_FOUND = 0x0001;
12. **public** **static** **final** **short** KEY\_EXISTS = 0x0002;
13. **public** **static** **final** **short** VALUE\_TOO\_LARGE = 0x0003;
14. **public** **static** **final** **short** INVALID\_ARGUMENTS = 0x0004;
15. **public** **static** **final** **short** ITEM\_NOT\_STORED = 0x0005;
16. **public** **static** **final** **short** INC\_DEC\_NON\_NUM\_VAL = 0x0006;
18. }

        继续编写memcached请求消息体：

**[java]** [view plaincopy](http://blog.csdn.net/abc_key/article/details/38388031)[在CODE上查看代码片](https://code.csdn.net/snippets/443257)

1. **package** netty.in.action.mem;
3. **import** java.util.Random;
5. /\*\*
6. \* memcached request message object
7. \* @author c.king
8. \*
9. \*/
10. **public** **class** MemcachedRequest {
12. **private** **static** **final** Random rand = **new** Random();
13. **private** **int** magic = 0x80;// fixed so hard coded
14. **private** **byte** opCode; // the operation e.g. set or get
15. **private** String key; // the key to delete, get or set
16. **private** **int** flags = 0xdeadbeef; // random
17. **private** **int** expires; // 0 = item never expires
18. **private** String body; // if opCode is set, the value
19. **private** **int** id = rand.nextInt(); // Opaque
20. **private** **long** cas; // data version check...not used
21. **private** **boolean** hasExtras; // not all ops have extras
23. **public** MemcachedRequest(**byte** opcode, String key, String value) {
24. **this**.opCode = opcode;
25. **this**.key = key;
26. **this**.body = value == **null** ? "" : value;
27. // only set command has extras in our example
28. hasExtras = opcode == Opcode.SET;
29. }
31. **public** MemcachedRequest(**byte** opCode, String key) {
32. **this**(opCode, key, **null**);
33. }
35. **public** **int** getMagic() {
36. **return** magic;
37. }
39. **public** **byte** getOpCode() {
40. **return** opCode;
41. }
43. **public** String getKey() {
44. **return** key;
45. }
47. **public** **int** getFlags() {
48. **return** flags;
49. }
51. **public** **int** getExpires() {
52. **return** expires;
53. }
55. **public** String getBody() {
56. **return** body;
57. }
59. **public** **int** getId() {
60. **return** id;
61. }
63. **public** **long** getCas() {
64. **return** cas;
65. }
67. **public** **boolean** isHasExtras() {
68. **return** hasExtras;
69. }
71. }

        最后编写memcached请求编码器：

**[java]** [view plaincopy](http://blog.csdn.net/abc_key/article/details/38388031)[在CODE上查看代码片](https://code.csdn.net/snippets/443257)

1. **package** netty.in.action.mem;
3. **import** io.netty.buffer.ByteBuf;
4. **import** io.netty.channel.ChannelHandlerContext;
5. **import** io.netty.handler.codec.MessageToByteEncoder;
6. **import** io.netty.util.CharsetUtil;
8. /\*\*
9. \* memcached request encoder
10. \* @author c.king
11. \*
12. \*/
13. **public** **class** MemcachedRequestEncoder **extends** MessageToByteEncoder<MemcachedRequest> {
15. @Override
16. **protected** **void** encode(ChannelHandlerContext ctx, MemcachedRequest msg, ByteBuf out)
17. **throws** Exception {
18. // convert key and body to bytes array
19. **byte**[] key = msg.getKey().getBytes(CharsetUtil.UTF\_8);
20. **byte**[] body = msg.getBody().getBytes(CharsetUtil.UTF\_8);
21. // total size of body = key size + body size + extras size
22. **int** bodySize = key.length + body.length + (msg.isHasExtras() ? 8 : 0);
23. // write magic int
24. out.writeInt(msg.getMagic());
25. // write opcode byte
26. out.writeByte(msg.getOpCode());
27. // write key length (2 byte) i.e a Java short
28. out.writeShort(key.length);
29. // write extras length (1 byte)
30. **int** extraSize = msg.isHasExtras() ? 0x08 : 0x0;
31. out.writeByte(extraSize);
32. // byte is the data type, not currently implemented in Memcached
33. // but required
34. out.writeByte(0);
35. // next two bytes are reserved, not currently implemented
36. // but are required
37. out.writeShort(0);
38. // write total body length ( 4 bytes - 32 bit int)
39. out.writeInt(bodySize);
40. // write opaque ( 4 bytes) - a 32 bit int that is returned
41. // in the response
42. out.writeInt(msg.getId());
43. // write CAS ( 8 bytes)
44. // 24 byte header finishes with the CAS
45. out.writeLong(msg.getCas());
46. **if**(msg.isHasExtras()){
47. // write extras
48. // (flags and expiry, 4 bytes each), 8 bytes total
49. out.writeInt(msg.getFlags());
50. out.writeInt(msg.getExpires());
51. }
52. //write key
53. out.writeBytes(key);
54. //write value
55. out.writeBytes(body);
56. }
58. }

**14.4.2 实现Memcached解码器**

        编写memcached响应消息体：

**[java]** [view plaincopy](http://blog.csdn.net/abc_key/article/details/38388031)[在CODE上查看代码片](https://code.csdn.net/snippets/443257)

1. **package** netty.in.action.mem;
3. /\*\*
4. \* memcached response message object
5. \* @author c.king
6. \*
7. \*/
8. **public** **class** MemcachedResponse {
10. **private** **byte** magic;
11. **private** **byte** opCode;
12. **private** **byte** dataType;
13. **private** **short** status;
14. **private** **int** id;
15. **private** **long** cas;
16. **private** **int** flags;
17. **private** **int** expires;
18. **private** String key;
19. **private** String data;
21. **public** MemcachedResponse(**byte** magic, **byte** opCode, **byte** dataType, **short** status,
22. **int** id, **long** cas, **int** flags, **int** expires, String key, String data) {
23. **this**.magic = magic;
24. **this**.opCode = opCode;
25. **this**.dataType = dataType;
26. **this**.status = status;
27. **this**.id = id;
28. **this**.cas = cas;
29. **this**.flags = flags;
30. **this**.expires = expires;
31. **this**.key = key;
32. **this**.data = data;
33. }
35. **public** **byte** getMagic() {
36. **return** magic;
37. }
39. **public** **byte** getOpCode() {
40. **return** opCode;
41. }
43. **public** **byte** getDataType() {
44. **return** dataType;
45. }
47. **public** **short** getStatus() {
48. **return** status;
49. }
51. **public** **int** getId() {
52. **return** id;
53. }
55. **public** **long** getCas() {
56. **return** cas;
57. }
59. **public** **int** getFlags() {
60. **return** flags;
61. }
63. **public** **int** getExpires() {
64. **return** expires;
65. }
67. **public** String getKey() {
68. **return** key;
69. }
71. **public** String getData() {
72. **return** data;
73. }
75. }

        编写memcached响应解码器：

**[java]** [view plaincopy](http://blog.csdn.net/abc_key/article/details/38388031)[在CODE上查看代码片](https://code.csdn.net/snippets/443257)

1. **package** netty.in.action.mem;
3. **import** io.netty.buffer.ByteBuf;
4. **import** io.netty.channel.ChannelHandlerContext;
5. **import** io.netty.handler.codec.ByteToMessageDecoder;
6. **import** io.netty.util.CharsetUtil;
8. **import** java.util.List;
10. **public** **class** MemcachedResponseDecoder **extends** ByteToMessageDecoder {
12. **private** **enum** State {
13. Header, Body
14. }
16. **private** State state = State.Header;
17. **private** **int** totalBodySize;
18. **private** **byte** magic;
19. **private** **byte** opCode;
20. **private** **short** keyLength;
21. **private** **byte** extraLength;
22. **private** **byte** dataType;
23. **private** **short** status;
24. **private** **int** id;
25. **private** **long** cas;
27. @Override
28. **protected** **void** decode(ChannelHandlerContext ctx, ByteBuf in, List<Object> out)
29. **throws** Exception {
30. **switch** (state) {
31. **case** Header:
32. // response header is 24 bytes
33. **if** (in.readableBytes() < 24) {
34. **return**;
35. }
36. // read header
37. magic = in.readByte();
38. opCode = in.readByte();
39. keyLength = in.readShort();
40. extraLength = in.readByte();
41. dataType = in.readByte();
42. status = in.readShort();
43. totalBodySize = in.readInt();
44. id = in.readInt();
45. cas = in.readLong();
46. state = State.Body;
47. **break**;
48. **case** Body:
49. **if** (in.readableBytes() < totalBodySize) {
50. **return**;
51. }
52. **int** flags = 0;
53. **int** expires = 0;
54. **int** actualBodySize = totalBodySize;
55. **if** (extraLength > 0) {
56. flags = in.readInt();
57. actualBodySize -= 4;
58. }
59. **if** (extraLength > 4) {
60. expires = in.readInt();
61. actualBodySize -= 4;
62. }
63. String key = "";
64. **if** (keyLength > 0) {
65. ByteBuf keyBytes = in.readBytes(keyLength);
66. key = keyBytes.toString(CharsetUtil.UTF\_8);
67. actualBodySize -= keyLength;
68. }
69. ByteBuf body = in.readBytes(actualBodySize);
70. String data = body.toString(CharsetUtil.UTF\_8);
71. out.add(**new** MemcachedResponse(magic, opCode, dataType, status,
72. id, cas, flags, expires, key, data));
73. state = State.Header;
74. **break**;
75. **default**:
76. **break**;
77. }
78. }
80. }

**14.5 测试编解码器**

        基于netty的编解码器都写完了，下面我们来写一个测试它的类：

**[java]** [view plaincopy](http://blog.csdn.net/abc_key/article/details/38388031)[在CODE上查看代码片](https://code.csdn.net/snippets/443257)

1. **package** netty.in.action.mem;
3. **import** io.netty.buffer.ByteBuf;
4. **import** io.netty.channel.embedded.EmbeddedChannel;
5. **import** io.netty.util.CharsetUtil;
7. **import** org.junit.Assert;
8. **import** org.junit.Test;
10. /\*\*
11. \* test memcached encoder
12. \* @author c.king
13. \*
14. \*/
15. **public** **class** MemcachedRequestEncoderTest {
17. @Test
18. **public** **void** testMemcachedRequestEncoder() {
19. MemcachedRequest request = **new** MemcachedRequest(Opcode.SET, "k1", "v1");
20. EmbeddedChannel channel = **new** EmbeddedChannel(
21. **new** MemcachedRequestEncoder());
22. Assert.assertTrue(channel.writeOutbound(request));
23. ByteBuf encoded = (ByteBuf) channel.readOutbound();
24. Assert.assertNotNull(encoded);
25. Assert.assertEquals(request.getMagic(), encoded.readInt());
26. Assert.assertEquals(request.getOpCode(), encoded.readByte());
27. Assert.assertEquals(2, encoded.readShort());
28. Assert.assertEquals((**byte**) 0x08, encoded.readByte());
29. Assert.assertEquals((**byte**) 0, encoded.readByte());
30. Assert.assertEquals(0, encoded.readShort());
31. Assert.assertEquals(2 + 2 + 8, encoded.readInt());
32. Assert.assertEquals(request.getId(), encoded.readInt());
33. Assert.assertEquals(request.getCas(), encoded.readLong());
34. Assert.assertEquals(request.getFlags(), encoded.readInt());
35. Assert.assertEquals(request.getExpires(), encoded.readInt());
36. **byte**[] data = **new** **byte**[encoded.readableBytes()];
37. encoded.readBytes(data);
38. Assert.assertArrayEquals((request.getKey() + request.getBody())
39. .getBytes(CharsetUtil.UTF\_8), data);
40. Assert.assertFalse(encoded.isReadable());
41. Assert.assertFalse(channel.finish());
42. Assert.assertNull(channel.readInbound());
43. }
45. }

**[java]** [view plaincopy](http://blog.csdn.net/abc_key/article/details/38388031)[在CODE上查看代码片](https://code.csdn.net/snippets/443257)

1. **package** netty.in.action.mem;
3. **import** io.netty.buffer.ByteBuf;
4. **import** io.netty.buffer.Unpooled;
5. **import** io.netty.channel.embedded.EmbeddedChannel;
6. **import** io.netty.util.CharsetUtil;
8. **import** org.junit.Assert;
9. **import** org.junit.Test;
11. /\*\*
12. \* test memcached decoder
13. \*
14. \* @author c.king
15. \*
16. \*/
17. **public** **class** MemcachedResponseDecoderTest {
19. @Test
20. **public** **void** testMemcachedResponseDecoder() {
21. EmbeddedChannel channel = **new** EmbeddedChannel(
22. **new** MemcachedResponseDecoder());
23. **byte** magic = 1;
24. **byte** opCode = Opcode.SET;
25. **byte** dataType = 0;
26. **byte**[] key = "Key1".getBytes(CharsetUtil.UTF\_8);
27. **byte**[] body = "Value".getBytes(CharsetUtil.UTF\_8);
28. **int** id = (**int**) System.currentTimeMillis();
29. **long** cas = System.currentTimeMillis();
30. ByteBuf buffer = Unpooled.buffer();
31. buffer.writeByte(magic);
32. buffer.writeByte(opCode);
33. buffer.writeShort(key.length);
34. buffer.writeByte(0);
35. buffer.writeByte(dataType);
36. buffer.writeShort(Status.KEY\_EXISTS);
37. buffer.writeInt(body.length + key.length);
38. buffer.writeInt(id);
39. buffer.writeLong(cas);
40. buffer.writeBytes(key);
41. buffer.writeBytes(body);
42. Assert.assertTrue(channel.writeInbound(buffer));
43. MemcachedResponse response = (MemcachedResponse) channel.readInbound();
44. assertResponse(response, magic, opCode, dataType, Status.KEY\_EXISTS, 0,
45. 0, id, cas, key, body);
46. }
48. **private** **static** **void** assertResponse(MemcachedResponse response, **byte** magic,
49. **byte** opCode, **byte** dataType, **short** status, **int** expires, **int** flags,
50. **int** id, **long** cas, **byte**[] key, **byte**[] body) {
51. Assert.assertEquals(magic, response.getMagic());
52. Assert.assertArrayEquals(key,
53. response.getKey().getBytes(CharsetUtil.UTF\_8));
54. Assert.assertEquals(opCode, response.getOpCode());
55. Assert.assertEquals(dataType, response.getDataType());
56. Assert.assertEquals(status, response.getStatus());
57. Assert.assertEquals(cas, response.getCas());
58. Assert.assertEquals(expires, response.getExpires());
59. Assert.assertEquals(flags, response.getFlags());
60. Assert.assertArrayEquals(body,
61. response.getData().getBytes(CharsetUtil.UTF\_8));
62. Assert.assertEquals(id, response.getId());
63. }
65. }

**14.6 Summary**

        本章主要是使用netty写了个模拟memcached二进制协议的处理。至于memcached二进制协议具体是个啥玩意，可以单独了解，这里也没有详细说明。