1. 线程组，在我们测试方案里面，每个线程模拟一个用户，执行用户的登录、等等等一系列的操作。由于我们的项目是长连接的，如何能实现多个sample公用一个长连接客户端，考虑了很久，最后实现方法如下：  
1 package tea.client.network;  
2 /\*\*  
3  \* @author Teaey  
4  \* @creation 2012-8-25  
5  \*/  
6 public class NetworkClientHolder  
7 {  
8     /\*\*  
9      \* 这里使用ThradLocal存储BaseClient  
10      \* 方便一轮测试的每个sample都是由同一个socketChannel发送  
11      \* 更真实的模拟用户  
12      \*/  
13     private static ThreadLocal<BaseClient> clientHolder = new ThreadLocal<BaseClient>();  
14     public static BaseClient getClient(String ip， String port)  
15     {  
16         BaseClient client = clientHolder.get();  
17         if (null == client)  
18         {  
19             client = new BaseClient(ip， port);  
20             client.connect();  
21             clientHolder.set(client);  
22         }  
23         return client;  
24     }  
25 }  
　　代码中使用thread\_local保存Socket客户端，这样每个sample中发送数据的客户端都是从这里拿的，就可以保证长连接的情况下，socket不会重复创建，很好的模拟了用户。  
　　当然不单单是链接可以保存，所有需要在线程中共享的数据都可以通过这种方法来实现。

　　2. 接下来是如何封装发送请求的客户端，这里用的netty，具体可以根据项目情况使用mina或者nio都可以。代码直接明了^\_^：  
1 package tea.client.network;  
2  
3 import java.net.InetSocketAddress;  
4 import java.util.concurrent.Executors;  
5 import org.jboss.netty.bootstrap.ClientBootstrap;  
6 import org.jboss.netty.channel.Channel;  
7 import org.jboss.netty.channel.ChannelFuture;  
8 import org.jboss.netty.channel.ChannelHandlerContext;  
9 import org.jboss.netty.channel.ChannelPipeline;  
10 import org.jboss.netty.channel.ChannelPipelineFactory;  
11 import org.jboss.netty.channel.ChannelStateEvent;  
12 import org.jboss.netty.channel.Channels;  
13 import org.jboss.netty.channel.ExceptionEvent;  
14 import org.jboss.netty.channel.MessageEvent;  
15 import org.jboss.netty.channel.SimpleChannelHandler;  
16 import org.jboss.netty.channel.socket.nio.NioClientSocketChannelFactory;  
17 import tea.common.network.ClientDecoder;  
18 import tea.common.network.ClientEncoder;  
19 import tea.common.network.ClientMessage;  
20  
21 /\*\*  
22  \* @author Teaey  
23  \* @creation 2012-8-25  
24  \*/  
25 public class BaseClient  
26 {  
27     public BaseClient(String ip， String port)  
28     {  
29         this.ip = ip;  
30         this.port = port;  
31     }  
32     private String           ip;  
33     private String           port;  
34     private Channel          channel;  
35     private ClientBootstrap  bootstrap;  
36     private Object           syn             = new Object();  
37     private static final int Receive\_Timeout = 10000;       //ms  
38     private ClientMessage    response        = null;  
39     public void connect()  
40     {  
41         bootstrap = new ClientBootstrap(new NioClientSocketChannelFactory(Executors.newCachedThreadPool()， Executors.newCachedThreadPool()));  
42         bootstrap.setOption("tcpNoDelay"， true);  
43         bootstrap.setPipelineFactory(new ClientPipelineFactory());  
44         while (true)  
45         {  
46             ChannelFuture future = bootstrap.connect(new InetSocketAddress(ip， Integer.parseInt(port)));  
47             future.awaitUninterruptibly(5000);  
48             if (future.isDone())  
49             {  
50                 channel = future.getChannel();  
51                 if (channel != null && channel.isConnected())  
52                 {  
53                     break;  
54                 }  
55             }  
56         }  
57     }  
58     public void disconnect()  
59     {  
60         if (channel.isConnected())  
61         {  
62             channel.disconnect();  
63         }  
64     }  
65     public boolean isConnected()  
66     {  
67         return channel.isConnected();  
68     }  
69     public void close()  
70     {  
71         if (this.channel.isOpen())  
72         {  
73             this.channel.close();  
74         }  
75         bootstrap.releaseExternalResources();  
76     }  
77     /\*\*  
78      \* 发送消息，无需返回  
79      \*/  
80     public void send(ClientMessage message)  
81     {  
82         channel.write(message);  
83     }  
84     /\*\*  
85      \* 发送消息，等待返回  
86      \*/  
87     public ClientMessage sendWaitBack(ClientMessage message)  
88     {  
89         response = null;  
90         try  
91         {  
92             channel.write(message);  
93             synchronized (syn)  
94             {  
95                 try  
96                 {  
97                     syn.wait(Receive\_Timeout);  
98                 } catch (InterruptedException e)  
99                 {  
100                     e.printStackTrace();  
101                 }  
102             }  
103             if (null == response)  
104             {  
105                 System.err.println("Receive response timeout");  
106             }  
107         } catch (Exception e)  
108         {  
109             e.printStackTrace();  
110         }  
111         return response;  
112     }  
113     class ClientPipelineFactory implements ChannelPipelineFactory  
114     {  
115         public ChannelPipeline getPipeline() throws Exception  
116         {  
117             ChannelPipeline p = Channels.pipeline();  
118             p.addLast("frameDecoder"， new ClientDecoder());  
119             p.addLast("fremeEncoder"， new ClientEncoder());  
120             p.addLast("logicHandler"， new ClientMsgHandler());  
121             return p;  
122         }  
123     }  
124     class ClientMsgHandler extends SimpleChannelHandler  
125     {  
126         public void messageReceived(ChannelHandlerContext ctx， MessageEvent e) throws Exception  
127         {  
128             Object obj = e.getMessage();  
129             if (obj instanceof ClientMessage)  
130             {  
131                 ClientMessage msg = (ClientMessage) obj;  
132                 response = msg;  
133                 synchronized (syn)  
134                 {  
135                     syn.notifyAll();  
136                 }  
137             }  
138         }  
139         public void channelConnected(ChannelHandlerContext ctx， ChannelStateEvent e) throws Exception  
140         {  
141             System.out.println("connected server:" + ctx.getChannel());  
142         }  
143         public void channelDisconnected(ChannelHandlerContext ctx， ChannelStateEvent e) throws Exception  
144         {  
145             System.out.println("disconnected server:" + ctx.getChannel());  
146         }  
147         public void exceptionCaught(ChannelHandlerContext ctx， ExceptionEvent e) throws Exception  
148         {  
149             System.out.println("Error in exceptionCaught:" + e.getCause());  
150         }  
151     }  
152 }  
　　这段代码展示了我们的客户端，这里所有的请求有两种发送模式，一种是发送并阻塞等待返回（sendWaitBack），第二种就是直接发送（send）。

　　3. 有了发送请求的客户端，那如何能够更简单的实现一个协议好让客户端发送，再贴一段代码^\_^：  
1 package tea.client.network;  
2  
3 import org.apache.jmeter.config.Arguments;  
4 import org.apache.jmeter.protocol.java.sampler.AbstractJavaSamplerClient;  
5 import org.apache.jmeter.protocol.java.sampler.JavaSamplerContext;  
6 import org.apache.jmeter.samplers.SampleResult;  
7 import com.google.protobuf.InvalidProtocolBufferException;  
8 import com.google.protobuf.MessageLite;  
9  
10 /\*\*  
11  \* @author Teaey  
12  \* @creation 2012-8-25  
13  \*/  
14 public abstract class BaseSample extends AbstractJavaSamplerClient  
15 {  
16     public static final String PARAM\_IP   = "ip";  
17     public static final String PARAM\_PORT = "port";  
18     public static final String VAR\_IP     = "${ip}";  
19     public static final String VAR\_PORT   = "${port}";  
20     protected BaseClient       client;  
21     public void addParameter(Arguments params)  
22     {  
23     }  
24     /\*\*  
25      \* Jmeter获取消息参数，默认配置ip和port两个参数  
26      \* 如果子类有更多参数，调用super.getDefaultParameters()获取Arguments后，继续设置其他方法  
27      \*/  
28     @Override  
29     public Arguments getDefaultParameters()  
30     {  
31         System.out.println("1.getDefaultParameters");  
32         Arguments params = new Arguments();  
33         params.addArgument(PARAM\_IP， VAR\_IP);  
34         params.addArgument(PARAM\_PORT， VAR\_PORT);  
35         addParameter(params);  
36         return params;  
37     }  
38     /\*\*  
39      \* runTest的前置方法  
40      \*/  
41     @Override  
42     public void setupTest(JavaSamplerContext context)  
43     {  
44         System.out.println("2.setupTest:" + context.containsParameter(PARAM\_IP));  
45         String ip = context.getParameter(PARAM\_IP);  
46         String port = context.getParameter(PARAM\_PORT);  
47         this.client = NetworkClientHolder.getClient(ip， port);  
48         System.out.println("thread--->" + Thread.currentThread().getId() + " client--->" + client);  
49     }  
50     /\*\*  
51      \* Jmeter调用，用于实际的测试  
52      \*/  
53     @Override  
54     public SampleResult runTest(JavaSamplerContext context)  
55     {  
56         SampleResult sample = getSample();  
57         sample.sampleStart();  
58         try  
59         {  
60             MessageLite response = doTest();  
61             String msg = response == null ? "" : response.toString();  
62             sample.setResponseMessage(msg);  
63             sample.setSuccessful(true);  
64         } catch (Exception e)  
65         {  
66             sample.setSuccessful(false);  
67             e.printStackTrace();  
68         } finally  
69         {  
70             sample.sampleEnd();  
71         }  
72         return sample;  
73     }  
74     /\*\*  
75      \* 获取本Sample的标签，子类实现  
76      \*/  
77     public abstract String getLabel();  
78     /\*\*  
79      \* 获取一个带标签的Sample  
80      \*/  
81     public SampleResult getSample()  
82     {  
83         SampleResult sample = new SampleResult();  
84         sample.setSampleLabel(getLabel());  
85         return sample;  
86     }  
87     /\*\*  
88      \* Jmeter调用，用于  
89      \*/  
90     @Override  
91     public void teardownTest(JavaSamplerContext context)  
92     {  
93         System.out.println("4.teardownTest");  
94     }  
95     /\*\*  
96      \* 需实现，具体测试的方法，调用client的send/sendWithBack发送请求  
97      \* 如无返回，放回null即可  
98      \*/  
99     public abstract MessageLite doTest() throws InvalidProtocolBufferException;  
100 }  
　　好的，这里封装了下AbstractJavaSamplerClient，每个消息默认包含ip和port参数，这可以再jmeter的用户变量中定义好。为了方便大家添加消息的参数，这里实现了空的

　　addParameter(Arguments params)方法，这样在具体消息中直接重写这个方法，来添加具体的参数。是不是很方便？^\_^，具体协议还需要实现的两个方法分别是：getLabel和doTest。第一个方法时用于报告显示的请求名字，一般定义为消息名字+“Label”就OKay。第二个方法就是我们重点重写的方法，这里再贴段代码，是一个具体消息的实现：  
1 package tea.client;  
2  
3 import com.google.protobuf.InvalidProtocolBufferException;  
4 import com.google.protobuf.MessageLite;  
5 import tea.client.network.BaseSample;  
6 import tea.common.network.ClientMessage;  
7 import tea.common.network.RPC.HeartBeat\_C2S;  
8 import tea.common.network.RPC.HeartBeat\_S2C;  
9  
10 /\*\*  
11  \* @author Teaey  
12  \* @creation 2012-8-24  
13  \*/  
14 public class HeartBeatSample extends BaseSample  
15 {  
16     @Override  
17     public MessageLite doTest() throws InvalidProtocolBufferException  
18     {  
19         HeartBeat\_C2S.Builder request = HeartBeat\_C2S.newBuilder();  
20         request.setTimestamp(System.currentTimeMillis());  
21         ClientMessage cm = new ClientMessage();  
22         cm.setContent(request.build().toByteArray());  
23         cm.setName("HeartBeat");  
24         ClientMessage sm = client.sendWaitBack(cm);  
25         HeartBeat\_S2C response = HeartBeat\_S2C.parseFrom(sm.getContent());  
26         return response;  
27     }  
28     @Override  
29     public String getLabel()  
30     {  
31         return "HeartBeatSample";  
32     }  
33 }  
　　可以看到doTest的工作就是封装请求，并拿到父类的client发送，然后返回响应（send方式返回null），Okay，大功告成。