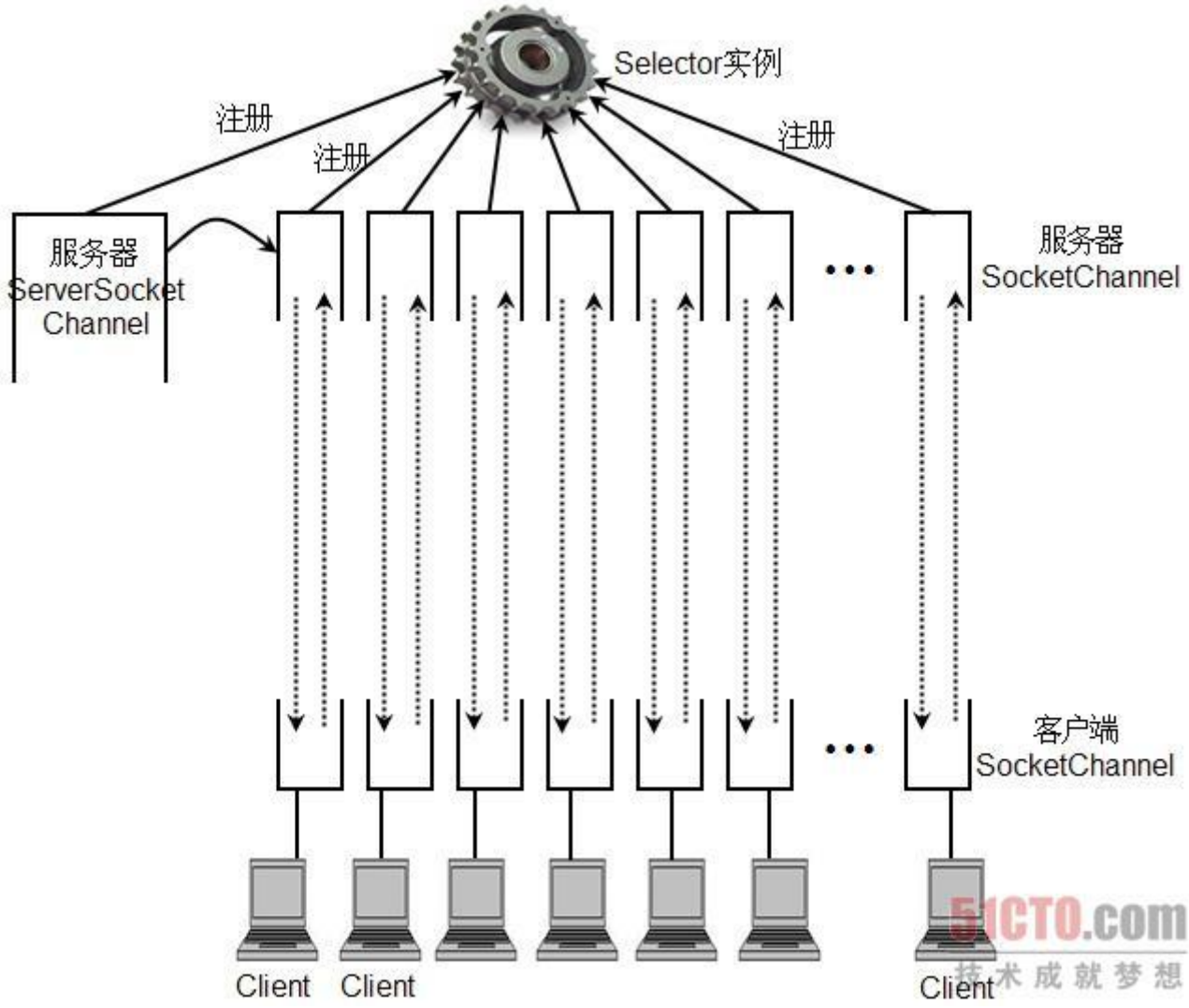
使用 SocketChannel 的 NIO 客户机服务器通信示例

这只是长征路上的一小步，以后还有待改进。

NIO Selector 示意图：



客户端代码：

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.SocketChannel;


/**
```

```
 * NIO TCP 客户端
 *
 * @date    2010-2-3
 * @time    下午 03:33:26
 * @version 1.00
 */
public class TCPClient{
 // 信道选择器
 private Selector selector;

 // 与服务器通信的信道
 SocketChannel socketChannel;

 // 要连接的服务器 Ip 地址
 private String hostIp;

 // 要连接的远程服务器在监听的端口
 private int hostListenningPort;

 /**
  * 构造函数
  * @param HostIp
  * @param HostListenningPort
  * @throws IOException
  */
 public TCPClient(String HostIp,int HostListenningPort)throws IOException{
   this.hostIp=HostIp;
   this.hostListenningPort=HostListenningPort;

   initialize();
 }

 /**
  * 初始化
  * @throws IOException
  */
 private void initialize() throws IOException{
   // 打开监听信道并设置为非阻塞模式
   socketChannel=SocketChannel.open(new InetSocketAddress(hostIp, hostListenningPort));
   socketChannel.configureBlocking(false);

   // 打开并注册选择器到信道
   selector = Selector.open();
   socketChannel.register(selector, SelectionKey.OP_READ);
```

```
    // 启动读取线程
    new TCPClientReadThread(selector);
  }

  /**
   * 发送字符串到服务器
   * @param message
   * @throws IOException
   */
  public void sendMsg(String message) throws IOException{
    ByteBuffer writeBuffer=ByteBuffer.wrap(message.getBytes("UTF-16"));
    socketChannel.write(writeBuffer);
  }

  public static void main(String[] args) throws IOException{
    TCPClient client=new TCPClient("192.168.0.1",1978);

    client.sendMsg("你好!Nio!醉里挑灯看剑,梦回吹角连营");
  }
}
```

客户端读取线程代码：

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.SocketChannel;
import java.nio.charset.Charset;

public class TCPClientReadThread implements Runnable{
  private Selector selector;

  public TCPClientReadThread(Selector selector){
    this.selector=selector;

    new Thread(this).start();
  }

  public void run() {
    try {
      while (selector.select() > 0) {
        // 遍历每个有可用 IO 操作 Channel 对应的 SelectionKey
```

```
      for (SelectionKey sk : selector.selectedKeys()) {

        // 如果该 SelectionKey 对应的 Channel 中有可读的数据
        if (sk.isReadable()) {
          // 使用 NIO 读取 Channel 中的数据
          SocketChannel sc = (SocketChannel) sk.channel();
          ByteBuffer buffer = ByteBuffer.allocate(1024);
          sc.read(buffer);
          buffer.flip();

          // 将字节转化为为 UTF-16 的字符串
          String receivedString=Charset.forName("UTF-16").newDecoder().decode(buffer).toString();

          // 控制台打印出来
          System.out.println("接收到来自服务器"+sc.socket().getRemoteSocketAddress()+"的信
息:"+receivedString);

          // 为下一次读取作准备
          sk.interestOps(SelectionKey.OP_READ);
        }

        // 删除正在处理的 SelectionKey
        selector.selectedKeys().remove(sk);
      }
    }
   } catch (IOException ex) {
     ex.printStackTrace();
   }
  }
}
```

服务器端代码：

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.util.Iterator;

/**
 * TCP 服务器端
 *
 * @date    2010-2-3
```

```
 * @time    上午 08:39:48
 * @version 1.00
 */
public class TCPServer{
  // 缓冲区大小
  private static final int BufferSize=1024;

  // 超时时间，单位毫秒
  private static final int TimeOut=3000;

  // 本地监听端口
  private static final int ListenPort=1978;

  public static void main(String[] args) throws IOException{
    // 创建选择器
    Selector selector=Selector.open();

    // 打开监听信道
    ServerSocketChannel listenerChannel=ServerSocketChannel.open();

    // 与本地端口绑定
    listenerChannel.socket().bind(new InetSocketAddress(ListenPort));

    // 设置为非阻塞模式
    listenerChannel.configureBlocking(false);

    // 将选择器绑定到监听信道,只有非阻塞信道才可以注册选择器.并在注册过程中指出该信道可以进行 Accept 操
作
    listenerChannel.register(selector, SelectionKey.OP_ACCEPT);

    // 创建一个处理协议的实现类,由它来具体操作
    TCPProtocol protocol=new TCPProtocolImpl(BufferSize);

    // 反复循环,等待 IO
    while(true){
      // 等待某信道就绪(或超时)
      if(selector.select(TimeOut)==0){
        System.out.print("独自等待.");
        continue;
      }

      // 取得迭代器.selectedKeys()中包含了每个准备好某一 I/O 操作的信道的 SelectionKey
      Iterator<SelectionKey> keyIter=selector.selectedKeys().iterator();
```

```
    while(keyIter.hasNext()){
      SelectionKey key=keyIter.next();

      try{
        if(key.isAcceptable()){
          // 有客户端连接请求时
          protocol.handleAccept(key);
        }

        if(key.isReadable()){
          // 从客户端读取数据
          protocol.handleRead(key);
        }

        if(key.isValid() && key.isWritable()){
          // 客户端可写时
          protocol.handleWrite(key);
        }
      }
      catch(IOException ex){
        // 出现 IO 异常（如客户端断开连接）时移除处理过的键
        keyIter.remove();
        continue;
      }

      // 移除处理过的键
      keyIter.remove();
    }
   }
  }
}
```

协议接口代码：

```
import java.io.IOException;
import java.nio.channels.SelectionKey;

/**
 * TCPServerSelector 与特定协议间通信的接口
 *
 * @date    2010-2-3
 * @time    上午 08:42:42
 * @version 1.00
 */
public interface TCPProtocol{
```

```
 /**
  * 接收一个 SocketChannel 的处理
  * @param key
  * @throws IOException
  */
 void handleAccept(SelectionKey key) throws IOException;

 /**
  * 从一个 SocketChannel 读取信息的处理
  * @param key
  * @throws IOException
  */
 void handleRead(SelectionKey key) throws IOException;

 /**
  * 向一个 SocketChannel 写入信息的处理
  * @param key
  * @throws IOException
  */
 void handleWrite(SelectionKey key) throws IOException;
}
```

协议实现类代码：

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.nio.charset.Charset;
import java.util.Date;

/**
 * TCPProtocol 的实现类
 *
 * @date    2010-2-3
 * @time    上午 08:58:59
 * @version 1.00
 */
public class TCPProtocolImpl implements TCPProtocol{
  private int bufferSize;

  public TCPProtocolImpl(int bufferSize){
    this.bufferSize=bufferSize;
  }
```

```
 public void handleAccept(SelectionKey key) throws IOException {
   SocketChannel clientChannel=((ServerSocketChannel)key.channel()).accept();
   clientChannel.configureBlocking(false);
   clientChannel.register(key.selector(), SelectionKey.OP_READ,ByteBuffer.allocate(bufferSize));
 }

 public void handleRead(SelectionKey key) throws IOException {
   // 获得与客户端通信的信道
   SocketChannel clientChannel=(SocketChannel)key.channel();

   // 得到并清空缓冲区
   ByteBuffer buffer=(ByteBuffer)key.attachment();
   buffer.clear();

   // 读取信息获得读取的字节数
   long bytesRead=clientChannel.read(buffer);

   if(bytesRead==-1){
     // 没有读取到内容的情况
     clientChannel.close();
   }
   else{
     // 将缓冲区准备为数据传出状态
     buffer.flip();

     // 将字节转化为为 UTF-16 的字符串
     String receivedString=Charset.forName("UTF-16").newDecoder().decode(buffer).toString();

     // 控制台打印出来
     System.out.println("接收到来自"+clientChannel.socket().getRemoteSocketAddress()+"的信
息:"+receivedString);

     // 准备发送的文本
     String sendString="你好,客户端. @"+new Date().toString()+",已经收到你的信息"+receivedString;
     buffer=ByteBuffer.wrap(sendString.getBytes("UTF-16"));
     clientChannel.write(buffer);

     // 设置为下一次读取或是写入做准备
     key.interestOps(SelectionKey.OP_READ | SelectionKey.OP_WRITE);
   }
 }

 public void handleWrite(SelectionKey key) throws IOException {
```

```
    // do nothing
  }
}
```