链滴

通过BIO及NIO方式实现简单Echo服务器

作者：lconline

原文链接：https://ld246.com/article/1465010248762

来源网站：链滴

对于分布式Java应用，各个系统间的通信方式有消息方式及远程调用方式。基于消息方式实现的系统通信，消息可以是字节流、字节数组，甚至是Java对象，系统接收到消息后进行相应的业务处理。消方式的系统间通信，通常基于网络协议来实现，例如TCP和UDP。TCP和UDP协议可用于完成数据传，但要完成系统间通信，还需要对数据进行处理，比如数据的读写操作。按照POSIX标准分为同步IO异步IO两种。其中同步IO中最常用的是BIO（Blocking IO）和NIO（Non-Blocking IO）。

Blocking IO，顾名思义，就是发起IO的读写操作是，均为阻塞方式，只有当程序读到了流或将流写操作系统后，才会释放资源。

Non-Blocking IO是基于事件驱动思想的，实现上通常采用Reactor模式。当发起IO的读写操作时，非阻塞的；当Socket有流可读或可写入Socket时，操作系统会相应地通知应用程序进行处理，应用序再将流读取到缓冲区，或写入操作系统。使用Java自身包可实现基于TCP或UDP的BIO或NIO四种统间通信方式。这里使用TCP+BIO以及TCP+NIO来实现一个简单的Echo服务器（即从客户端控制台入一个字符串，该字符串发送到服务器后，再回传到客户端，客户端再将该字符在控制台原样打印）以对这两种方式做一个总结记录。

最最基本的Socket程序

首先是一个最基本最简单的基于Socket的系统间通信的例子，对于服务器端程序，建立一个ServerSocket监听，并使用Socket获取输出流输出，关键代码如下：

```
try (ServerSocket server = new ServerSocket()) {
  InetSocketAddress isa = new InetSocketAddress(ADDR, PORT);
  server.bind(isa);
  while(true) {
    Socket s = server.accept();
    PrintStream ps = new PrintStream(s.getOutputStream());
    ps.println("you have connected to the server");
  }
} catch (Exception e) {
  e.printStackTrace();
}
```