# Netty源码解析

## 第1节 Netty深入剖析

### ·使用Netty作为底层框架的计数



### ·Netty是什么

·异步事件驱动框架，用于快速开发高性能服务器和客户端

·封装了JDK底层BIO和NIO模型，提供高度可用的API

·自带编解码器解决拆包粘包问题，用户只关心业务逻辑

·精心设计的reactor线程模型支持高并发海量连接

·自带各种协议栈

### ·学习的必要性

·各大开源项目选择Netty作为底层通信框架

·更好的使用，少走弯路

·遇到Bug？单机连接数上不去？性能遇到瓶颈？如何调优

·详解reactor线程模型，实践中举一反三

·庞大的项目是如何组织的，设计模式，体验优秀的设计

·阅读源码其实没那么困难

### ·怎么学

·自己摸索不如前人之路

·对于Socket编程，逐个切入

·踩过的坑，积累的经验总结

·调试，分析技巧

·各章节独立模块进行拆分

·总分总的方式渐进讲解

·实例演示，试图演示，调试分析

### ·学习目标

·掌握Netty底层原理，轻松解决各类疑难杂症，深度调优

·给官方提issue

·实现一个简易版的Netty

·开启阅读源码之旅

### ·适合人群

·系统掌握Netty底层核心原理

·加速掌握基于Nett的各种中间件框架

·对技术有追求，对优秀代码有执念的开发者

·从未阅读过源码，有勇气迈出第一步的开发者

### ·技术储备

·熟悉java基础，熟悉多线程

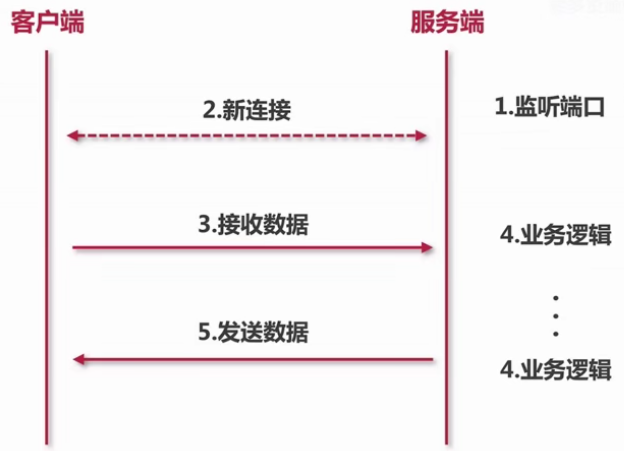
·熟悉TCP原理，NIO

·使用过Netty

·maven，Intellij IDEA

## Netty基本组件

### 2.1 TCP示例

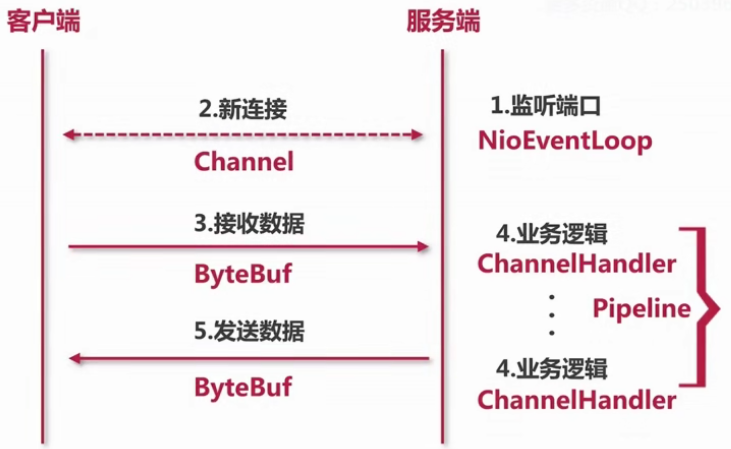


|  |
| --- |
| **import** java.io.IOException; **import** java.net.ServerSocket; **import** java.net.Socket;  **public class** Server {   **private** ServerSocket **serverSocket**;   **public** Server(**int** port) {  **try** {  **this**.**serverSocket** = **new** ServerSocket(port);  System.***out***.println(**"服务端启动成功，端口:"** + port);  } **catch** (IOException exception) {  System.***out***.println(**"服务端启动失败"**);  }  }   **public void** start() {  **new** Thread(**new** Runnable() {  @Override  **public void** run() {  doStart();  }  }).start();  }   **private void** doStart() {  **while** (**true**) {  **try** {  Socket client = **serverSocket**.accept();  **new** ClientHandler(client).start();  } **catch** (IOException e) {  System.***out***.println(**"服务端异常"**);  }  }  } } |

|  |
| --- |
| **import** java.io.IOException; **import** java.io.InputStream; **import** java.net.Socket;  **public class** ClientHandler {   **public static final int *MAX\_DATA\_LEN*** = 1024;  **private final** Socket **socket**;   **public** ClientHandler(Socket socket) {  **this**.**socket** = socket;  }   **public void** start() {  System.***out***.println(**"新客户端接入"**);  **new** Thread(**new** Runnable() {  @Override  **public void** run() {  doStart();  }  }).start();  }   **private void** doStart() {  **try** {  InputStream inputStream = **socket**.getInputStream();  **while** (**true**) {  **byte**[] data = **new byte**[***MAX\_DATA\_LEN***];  **int** len;  **while** ((len = inputStream.read(data)) != -1) {  String message = **new** String(data, 0, len);  System.***out***.println(**"客户端传来消息: "** + message);  **socket**.getOutputStream().write(data);  }   }    } **catch** (IOException e) {  e.printStackTrace();  }  } } |

|  |
| --- |
| **public class** ServerBoot {   **private static final int *PORT*** = 8000;   **public static void** main(String[] args) {  Server server = **new** Server(***PORT***);  server.start();  }  } |
| **import** java.io.IOException; **import** java.net.Socket;  **public class** Client {  **private static final** String ***HOST*** = **"127.0.0.1"**;  **private static final int *PORT*** = 8000;  **private static final int *SLEEP\_TIME*** = 5000;   **public static void** main(String[] args) **throws** IOException {  **final** Socket socket = **new** Socket(***HOST***, ***PORT***);   **new** Thread(**new** Runnable() {  @Override  **public void** run() {  System.out.println(**"客户端启动成功!"**);  **while** (**true**) {  **try** {  String message = **"hello world"**;  System.out.println(**"客户端发送数据: "** + message);  socket.getOutputStream().write(message.getBytes());  } **catch** (Exception e) {  System.out.println(**"写数据出错!"**);  }  sleep();  }    }  }).start();   }   **private static void** sleep() {  **try** {  Thread.*sleep*(***SLEEP\_TIME***);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  } } |
| **服务端启动成功，端口:8000**  **新客户端接入**  **客户端传来消息: hello world**  **客户端传来消息: hello world**  **客户端传来消息: hello world**  **客户端传来消息: hello world** |
| **客户端启动成功!**  **客户端发送数据: hello world**  **客户端发送数据: hello world**  **客户端发送数据: hello world**  **客户端发送数据: hello world**  **客户端发送数据: hello world** |

### 2.2Netty对于socket的抽象



### 2.3Netty组件介绍

·NioEventLoop == Socket示例中的Thread

1. 服务端接收客户端连接线程；
2. 处理每个连接的读写线程

|  |
| --- |
| Tips：intelij iead快捷键  Ctrl + n 模糊搜索类名  Ctrl + f12 类中搜索方法  Ctrl + e 弹出最近使用的文件，可快速切换  Alt + Shift + a 最近修改过的class |