Java高级网络通信之Nio和Netty

一、NIO

1.1Java中的IO模型

Java的IO模型：BIO、NIO、AIO

BIO：同步阻塞IO流

NIO：同步非阻塞IO流

AIO：异步非阻塞IO流，NIO2 NIO升级版

1.2 NIO

是jdk1.4之后，才有的。作用跟BIO一样，数据通信

性能要比BIO高，主要的作用：

1、常规的IO流，实现标准输入、输出

2、基于网络编程的IO流，基于Socket实现网络通信

1.3 NIO核心

1、Channel

通道，类似Stream(数据通信的管道)，可以实现数据的交互

常用

1、数据的读取和写出

2、异步的数据处理

3、要么就是先写出，要么就是先读取

常见的Channel：

1、FileChannel

2、SocketChannel

3、ServerSocketChannel

4、DatagramChannel

2、Buffer

缓冲,用于和Channel进行交互的组件，数据从通道中读取到缓存中，数据从缓存中写出到通道中

数据载体

使用要点：

1、数据 写出到Buffer

2、从Buffer读取数据

3、clear(清空缓存的数据)、compact(清空读取过的数据)

4、flip(切换读写模式)

核心属性：

1、limit 写模式：和容量 读模式：可以读取的最多的内容

2、position 当前的指针位置

3、capactily 容量

标记索引 缓冲中的指针

常见的Buffer:

1、ByteBuffer

2、CharBuffer

3、IntBuffer

基本类型的Buffer都有

3、Selector

选择器，可以进行检测有多少个通道，查看通道的状态

一个线程可以出来多个通道

实现效果：选择器可以管理多个Channel

常见的事件：

1、CONNECT 连接

2、ACCEPT 阻塞监听客户端连接

3、READ 读取

4、WRITE 写出

选择器监听四大事件

二、NIO初体验

2.1 Channel应用

2.2 Buffer应用

2.3 Selector应用

2.4 综合体验

三、NIO的缺陷

代码复杂度太高，生产环境中使用不易

四、Netty初识

4.1 简介

Netty封装了JDK的NIO，Netty是一个NIO客户端服务器框架，可以快速轻松地开发协议服务器和客户端等网络应用程序。它极大地简化并

简化了TCP和UDP套接字服务器等网络编程

Netty经过精心设计，具有丰富的协议，如FTP，SMTP，HTTP以及各种二进制和基于文本的传统协议

Netty是 *一个异步事件驱动的网络应用程序框架，* 用于快速开发可维护的高性能协议服务器和客户端

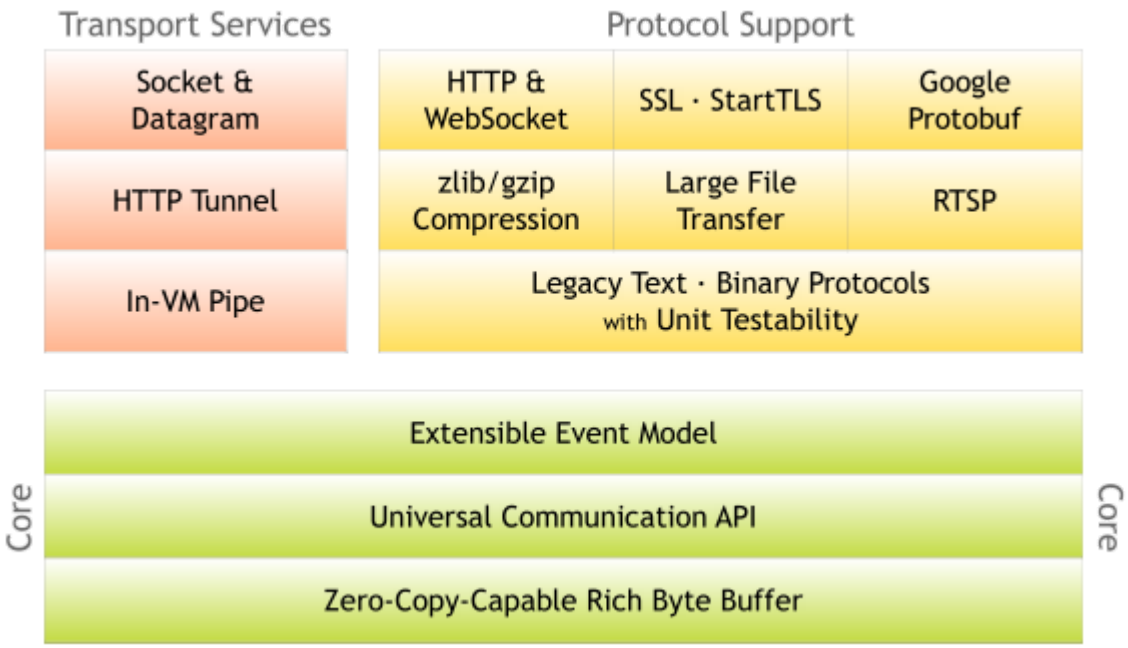
4.2 作用

- 适用于各种传输类型的统一API - 阻塞和非阻塞套接字
- 基于灵活且可扩展的事件模型，可以清晰地分离关注点
- 高度可定制的线程模型 - 单线程，一个或多个线程池，如SEDA
- 真正的无连接数据报套接字支持（自3.1起）

4.3 性能

- 更高的吞吐量，更低的延迟
- 减少资源消耗
- 最小化不必要的内存复制

4.4 Netty组成原理



五、Netty初体验

2.1 依赖jar

``

```
<dependency>
        <groupId>io.netty</groupId>
        <artifactId>netty-all</artifactId>
        <version>4.1.37.Final</version>
    </dependency>
```

2.2 代码实现

2.2.1 服务端

``

```
public class NettyServer {
    public static void main(String[] args) {
        System.out.println("服务器已经启动.....");
        new NettyServer().startServer();
    }
    public void startServer(){
        ServerBootstrap serverBootstrap = new ServerBootstrap();

        NioEventLoopGroup boos = new NioEventLoopGroup();
        NioEventLoopGroup worker = new NioEventLoopGroup();
        serverBootstrap
```

```
                .group(boos, worker)
                .channel(NioServerSocketChannel.class)
                .childHandler(new ChannelInitializer<NioSocketChannel>() {
                    @Override
                    protected void initChannel(NioSocketChannel ch) {
                        ch.pipeline().addLast(new StringDecoder());
                        ch.pipeline().addLast(new SimpleChannelInboundHandler<String>() {
                            @Override
                            protected void channelRead0(ChannelHandlerContext ctx, String
msg) {
                                System.out.println(msg);
                            }
                        });
                    }
                })
                .bind(8989);
    }
}
```

2.2.2 客户端

``

```
public class NettyClient {

    public static void main(String[] args) {
        new NettyClient().startClient();
    }
    public void startClient(){
        Bootstrap bootstrap = new Bootstrap();
        NioEventLoopGroup group = new NioEventLoopGroup();

        bootstrap.group(group)
                .channel(NioSocketChannel.class)
                .handler(new ChannelInitializer<Channel>() {
                    @Override
                    protected void initChannel(Channel ch) {
                        ch.pipeline().addLast(new StringEncoder());
                    }
                });

        Channel channel = bootstrap.connect("127.0.0.1", 8989).channel();

        while (true) {
            channel.writeAndFlush(new Date() + ": 这里是Netty");

        }
    }
}
```

源码地址：https://github.com/xingpenghui/FourStudy1901

三、Netty核心

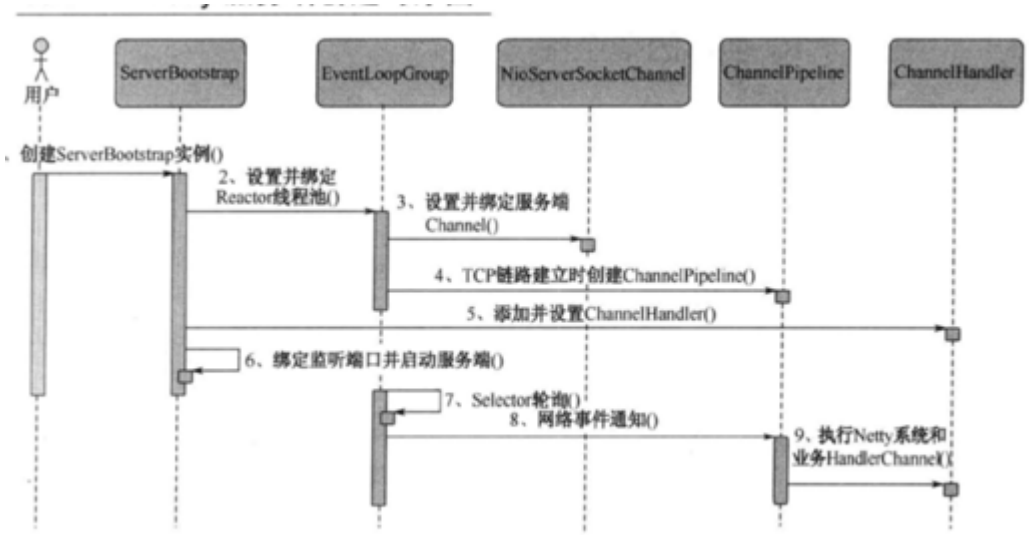


图 13-1　Netty 服务端创建时序图

JDK8:

Lambda表达式：简写形式 匿名内部类

可以用来实现接口

语法格式：

([参数])->{方法的重写}

语法要求：

1、接口只能有一个抽象方法

函数式编程