11-网络编程

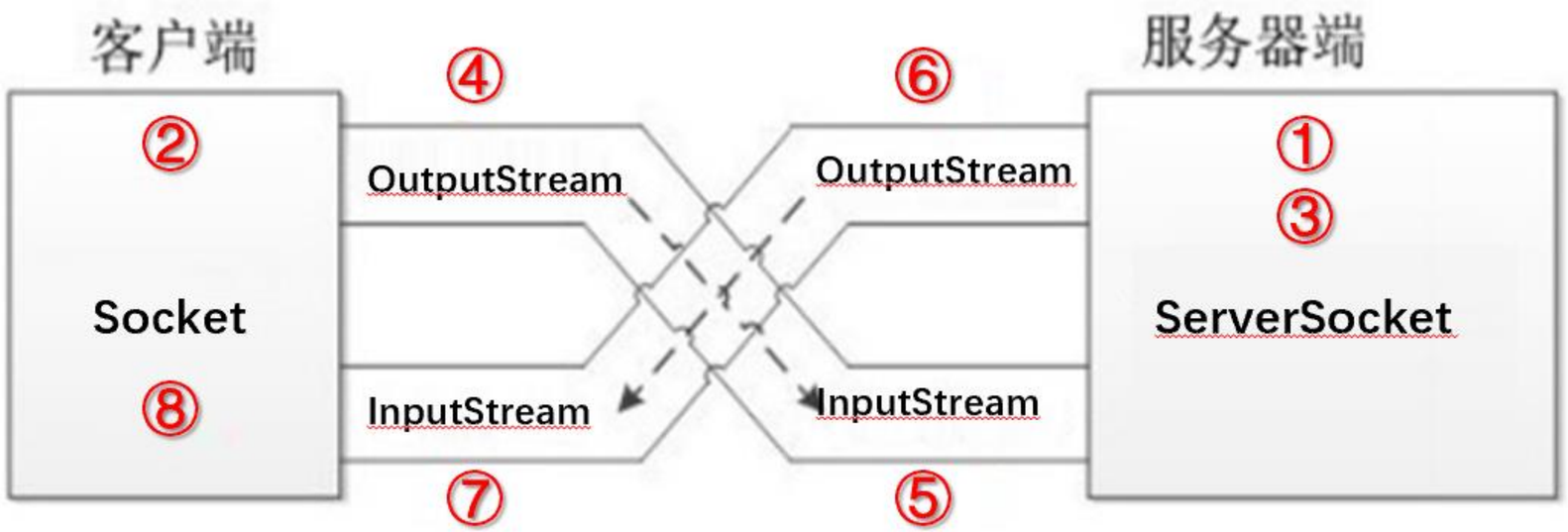
刘亚雄

极客时间-Java 讲师

二、BIO与NIO

2.1 BIO

BIO全称是Basic（基本） IO，Java1.4之前建立网络连接只能使用BIO，处理数据是以**字节**为单位

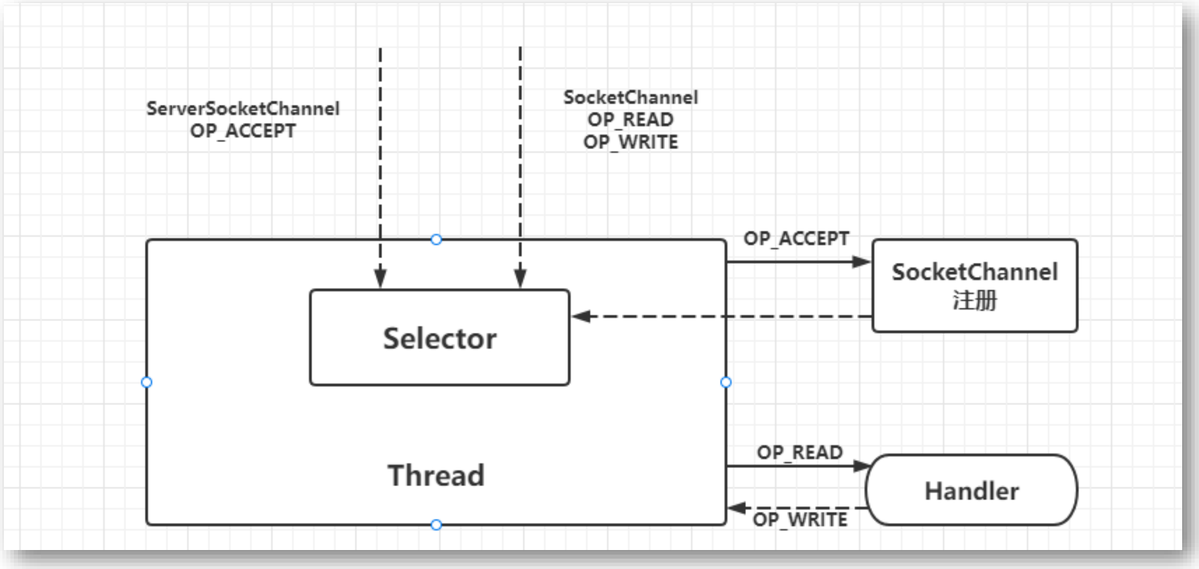


做个案例：看一下阻塞式和以字节为单位传输数据

2.2 NIO

java.nio 全称Java Non-Blocking IO，JDK1.4开始，改进后的IO，NIO和BIO的目的和作用相同，但是实现方式不同

- **效率不同：**BIO以字节为单位处理数据，NIO以块为单位处理数据
- **是否阻塞：**BIO是阻塞式的，NIO是非阻塞式的
- **数据流向：**BIO单向、NIO双向



NIO三个核心概念：

① Channel通道

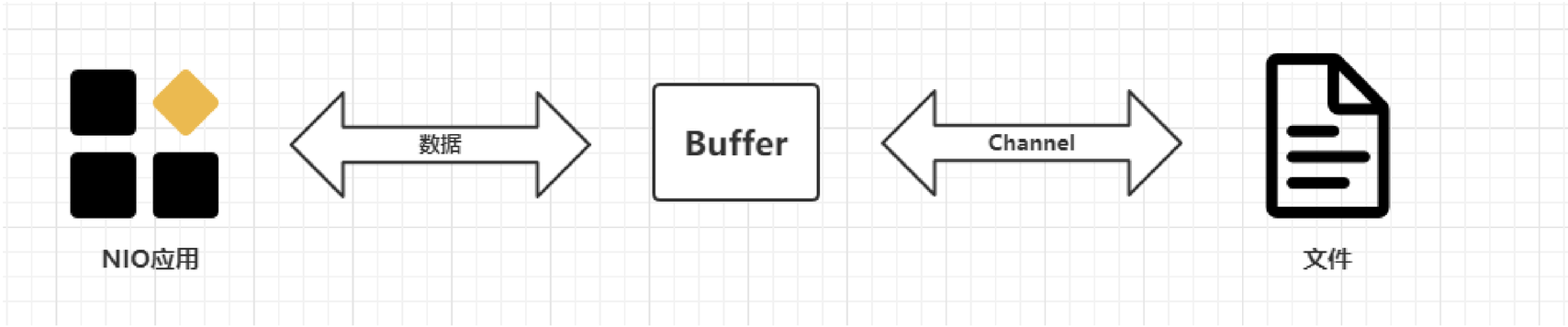
② Buffer缓冲区

③ Selector选择器

NIO基于Channel和Buffer进行操作，数据总是从Channel读取到Buffer中，或从Buffer写入到Channel

Selector监听多个Channel的事件，使用单个线程就可以监听多个客户端Channel

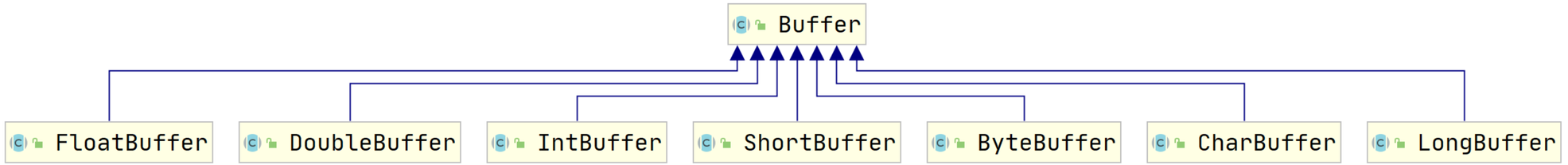
2.3 NIO-文件 IO



Buffer（缓冲区）： 是一个缓冲容器（底层是数组）内置了一些机制能够跟踪和记录缓冲区的状态变化。

Channel（通道）： 提供从文件、网络读取数据的通道， 读取或写入数据都必须经由 Buffer

2.3 NIO-文件 IO-Buffer



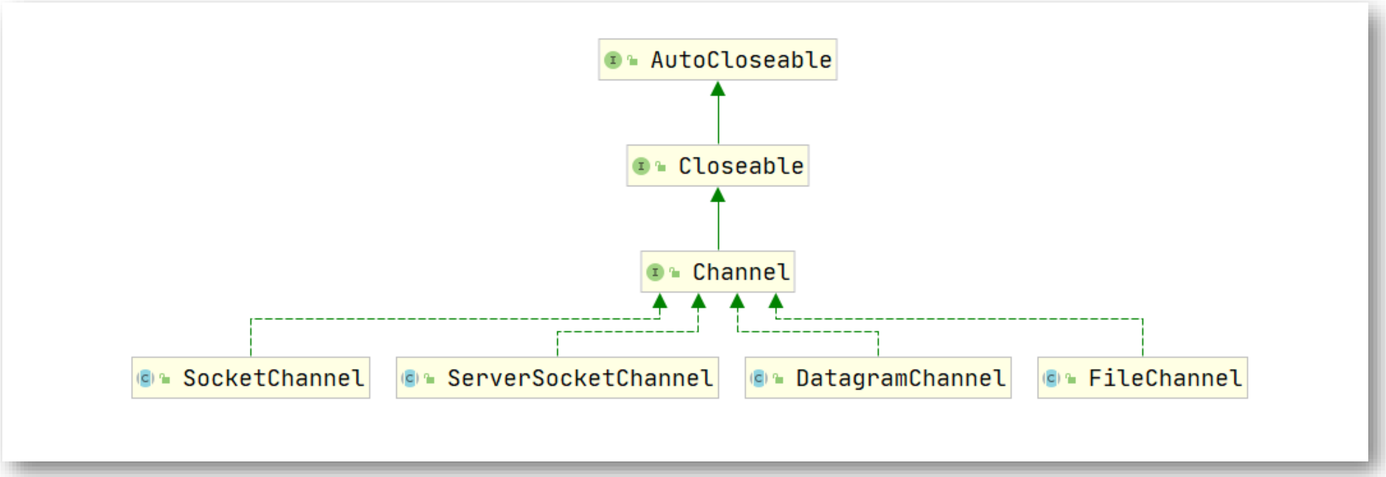
在NIO中Buffer顶层抽象父类，表示一个缓冲区，Channel读写数据都是放入Buffer中进行

常用的 Buffer 子类有： ByteBuffer、ShortBuffer、CharBuffer...

主要方法：

- ByteBuffer put(byte[] b); 存储字节数据到Buffer
- byte[] get(); 从Buffer获得字节数据
- byte[] array(); 把Buffer数据转换成字节数组
- ByteBuffer allocate(int capacity); 设置缓冲区的初始容量
- ByteBuffer wrap(byte[] array); 把数组放到缓冲区中
- **Buffer flip(); 翻转缓冲区，重置位置到初始位置**

2.3 NIO-文件 IO-Buffer



在NIO中Channel是一个接口，表示通道，通道是双向的，可以用来读，也可以用来写数据

常用Channel实现类： FileChannel、DatagramChannel、ServerSocketChannel 和 SocketChannel

- **FileChannel 文件数据读写**
- DatagramChannel 用于UDP数据读写
- **ServerSocketChannel 和 SocketChannel 用于TCP数据读写**

做个案例：NIO复制文件

FileChannel 类主要方法：

- int read(ByteBuffer dst)，从Channel读取数据并放到Buffer中
- int write(ByteBuffer src)，把Buffer的数据写到Channel中
- long transferFrom(ReadableByteChannel src, position, count)，从**目标Channel**中复制数据到**当前Channel**
- long transferTo(position, count, WritableByteChannel target)，把数据从**当前Channel**复制给**目标Channel**

2.4 NIO-网络 IO

JavaNIO中网络通道是**非阻塞IO，基于事件驱动，**很适合需要维持大量连接，但数据交换量不大的场景

例如：RPC、即时通讯、Web服务器...

Java编写网络应用，有以下几种模式：

- **为每个请求创建线程：**一个客户端连接用一个线程，**阻塞式IO**
  - 优点：程序编写简单
  - 缺点：如果连接非常多，分配的线程也会非常多，服务器可能会因为资源耗尽而崩溃
  - 案例：手写网站服务器-多线程版本
- **线程池：**创建固定数量线程的线程池，来接收客户端连接，**阻塞式IO**
  - 优点：程序编写相对简单， 可以处理大量的连接
  - 缺点：线程的开销非常大，连接如果非常多，排队现象会比较严重
  - 案例：手写网站服务器-线程池版本
- **JavaNIO：可以是阻塞，也可以是非阻塞式IO**
  - **优点：这种模式可以用一个线程，处理大量的客户端连接**
  - 缺点：代码复杂度较高

理解NIO关键在于理解Selector

2.4 NIO-网络 IO-Selector

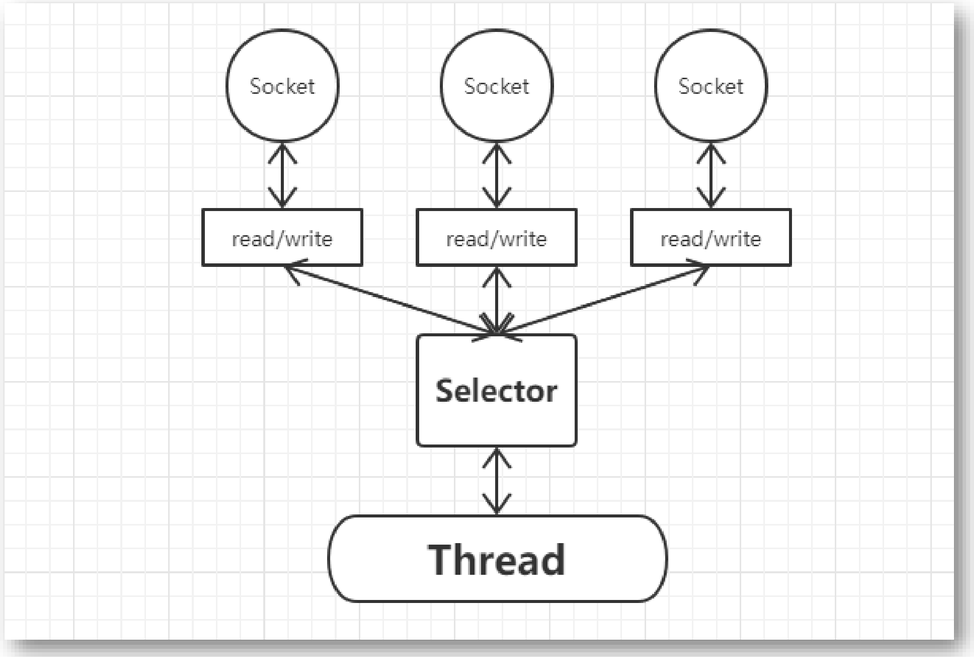
Selector选择器也叫多路复用器

- NIO三大核心组件之一
- 用于检测多个注册Channel上是否有事件发生（读、写、连接）如果有就获取事件，并对每个事件进行处理
- 只需一个单线程就可以管理多个Channel，也就是多个连接。
- 只有连接真正有读写事件时，才会调用方法来处理，大大降低了系统分配线程与线程上下文切换的开销

常用方法：

- Selector open()，开启一个**Selector**
- int select(long timeout)，监控所注册的通道
- selectedKeys()，从Selector获取所有SelectionKey

SelectionKey是什么？



2.4 NIO-网络 IO-Selector

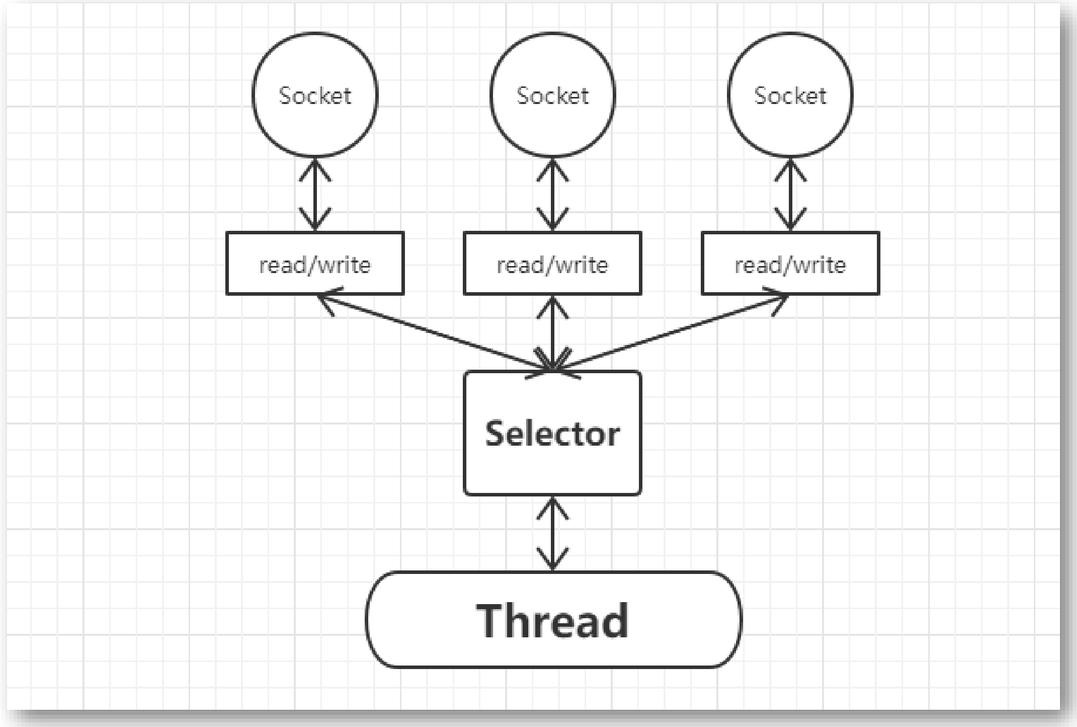
SelectionKey：代表了 Selector 和网络SocketChannel的注册关系

一共四种：

- **OP_ACCEPT：**有新的网络连接可以 accept
- **OP_CONNECT：**代表连接已经建立
- **OP_READ和OP_WRITE：**代表了读和写操作

常用方法：

- Selector selector()，得到与之关联的 Selector 对象
- SelectableChannel channel()，得到与之关联的通道
- Object attachment()，得到与之关联的共享数据
- boolean isAcceptable()，是否可接入
- boolean isReadable()，是否可以读
- boolean isWritable()，是否可以写



2.4 NIO-网络 IO-Selector

ServerSocketChannel：用来在Server端监听新的Client的Socket连接，常用方法如下：

- ServerSocketChannel open()，开启一个ServerSocketChannel 通道
- ServerSocketChannel bind(SocketAddress local)，设置Server端口号
- SelectableChannel configureBlocking(block)，设置阻塞模式， false表示采用非阻塞模式
- SocketChannel accept()，接受一个连接，返回值代表这个连接的Channel对象
- SelectionKey register(Selector sel, int ops)，注册Selector并设置监听事件

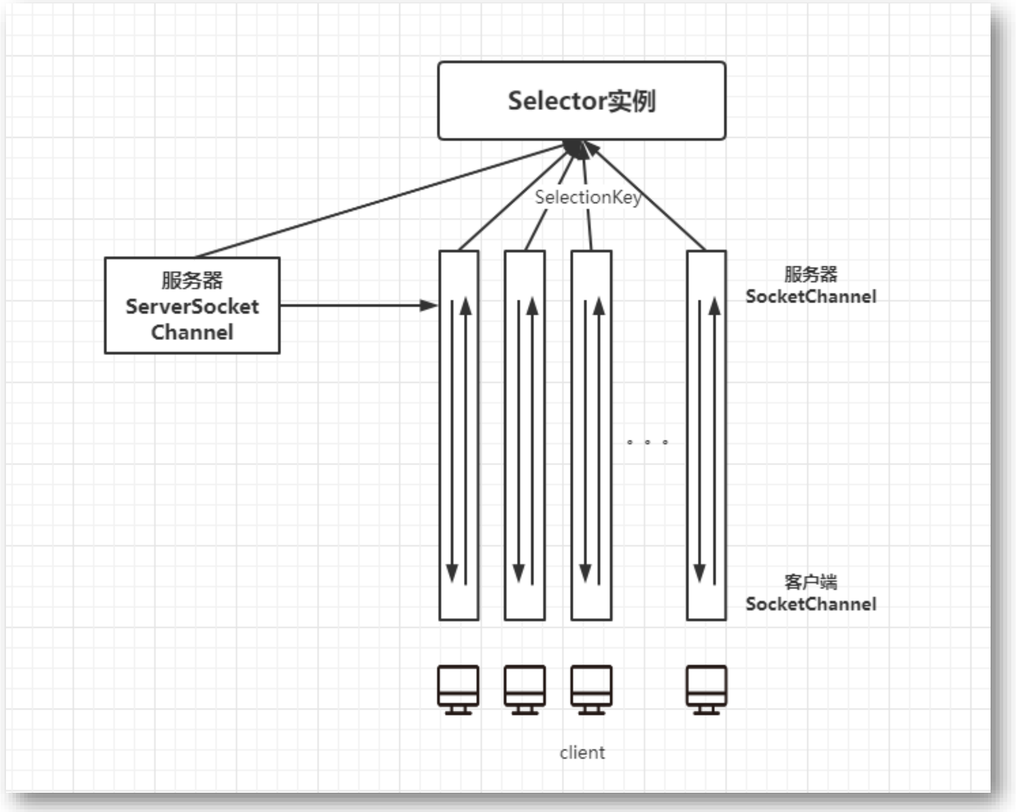
SocketChannel：网络IO通道，具体负责进行读写操作，常用方法如下：

- SocketChannel open()，得到一个 SocketChannel 通道
- SelectableChannel configureBlocking(block)，设置阻塞模式， false表示采用非阻塞模式
- boolean connect(SocketAddress remote)，连接服务器
- boolean finishConnect()，如果connect连接失败，接下来就要通过本方法完成连接
- int write(ByteBuffer src)，往通道里写数据
- int read(ByteBuffer dst)，从通道里读数据
- SelectionKey register(Selector sel, ops, att)，注册Selector并设置监听事件
- void close()，关闭通道

2.5 NIO-网络 IO-Selector

Selector与SelectionKey、ServerSocketChannel、SocketChannel的关系

- Server端有一个Selector对象
- ServerSocketChannel通道要注册给selector， selector#accept方法负责接收Client连接请求
- 有一个Client连接过来， Server就会建立一个SocketChannel
- Selector会监控所有注册的SocketChannel ，检查通道中是否有事件发生【连接、断开、读、写等事件】
- 如果某个SocketChannel有事件发生则做相应的处理



NIO案例：网络聊天室V1.0

2.6 IO对比总结

IO的方式通常分为几种：同步阻塞的 BIO、同步非阻塞的 NIO、异步非阻塞的 AIO

- **BIO 方式：**适用于连接数目较小且固定的架构
- 对服务器资源要求较高，并发局限于应用中
- JDK1.4以前的唯一选择，**同步阻塞式**，程序直观简单易理解
- 举个栗子 ： 食堂排队取餐，中午去食堂吃饭，排队等着，啥都干不了，到你了选餐，付款，然后找位子吃饭
- **NIO 方式：**适用于连接数目多且连接比较短（轻操作）的架构
- 比如：聊天服务器，并发局限于应用中，编程比较复杂
- JDK1.4 开始支持，**同步非阻塞式**
- 举个栗子 ： 下馆子，点完餐，就去商场玩儿了。玩一会儿，就回饭馆问一声：好了没
- **AIO 方式：**使用于连接数目多且连接比较长（重操作）的架构
- 比如：相册服务器，充分调用 OS 参与并发操作，编程比较复杂
- JDK1.7开始支持，**异步非阻塞式**
- 举个栗子：海底捞外卖火锅，打电话订餐。海底捞会说，我们知道您的位置，一会给您送过来，请您安心工作。

2.6 IO对比总结

IO的方式通常分为几种：同步阻塞的 BIO、同步非阻塞的 NIO、异步非阻塞的 AIO

对比	BIO	NIO	AIO
IO方式	同步阻塞	同步非阻塞（多路复用）	异步非阻塞
API使用难度	简单	复杂	复杂
可靠性	差	好	好
吞吐量	低	高	高

三、Netty核心技术

3.1 Netty简介

01-什么是Netty？

- Netty是一个被广泛使用的Java**网络应用**编程框架。
- Netty框架帮助开发者**快速、简单**的实现一个客户端/服务端的网络应用程序。
- Netty利用 Java 语言的NIO网络编程的能力，并隐藏其背后的复杂性从而提供了简单易用的 API

02-特点：

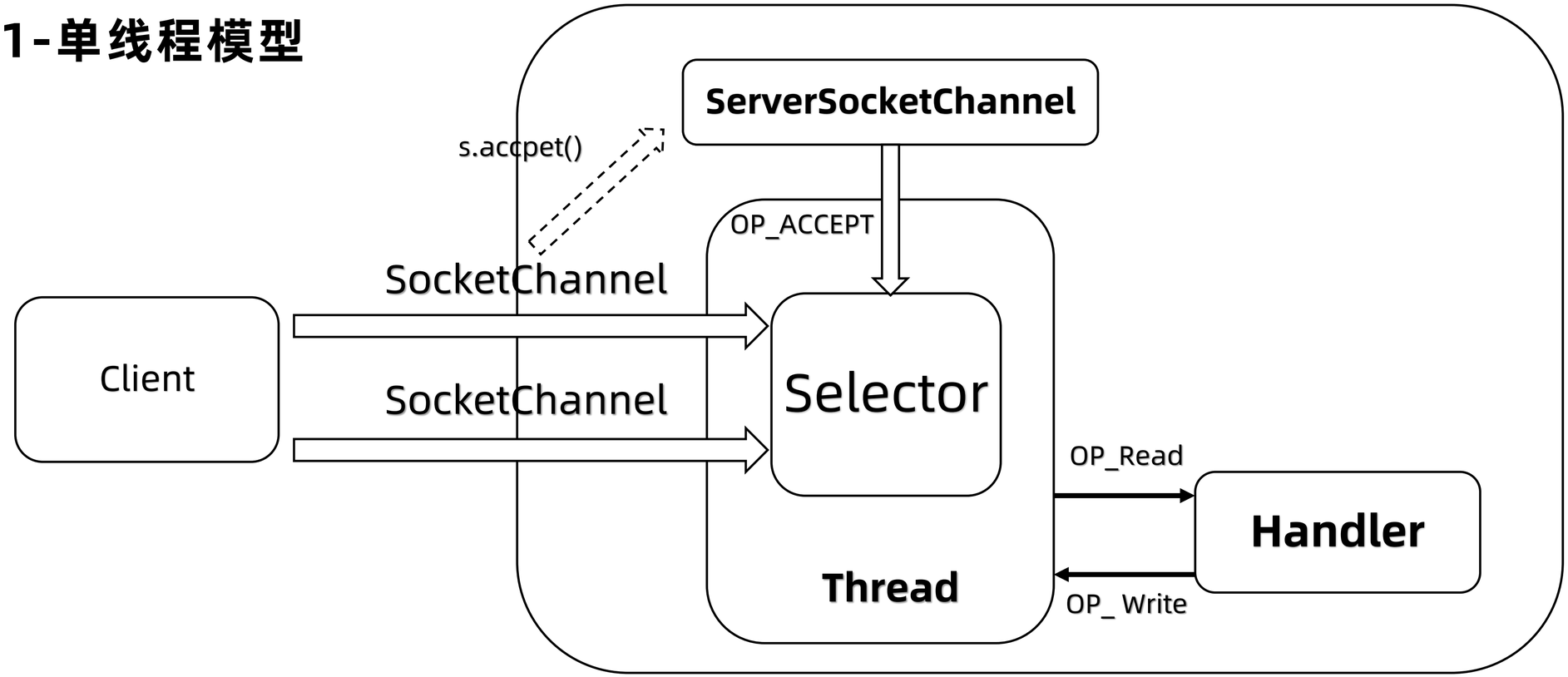
- API简单易用：支持阻塞和非阻塞式的socket
- 基于事件模型：可扩展性和灵活性更强
- 高度定制化的线程模型：支持单线程和多线程
- 高通吐、低延迟、资源占用率低
- 完整支持SSL和TLS

03-应用场景：

- 互联网行业：分布式系统远程过程调用，高性能的RPC框架
- 游戏行业：大型网络游戏高性能通信
- 大数据：Hadoop的高性能通信和序列化组件 Avro 的 RPC 框架

3.2 线程模型

01-单线程模型

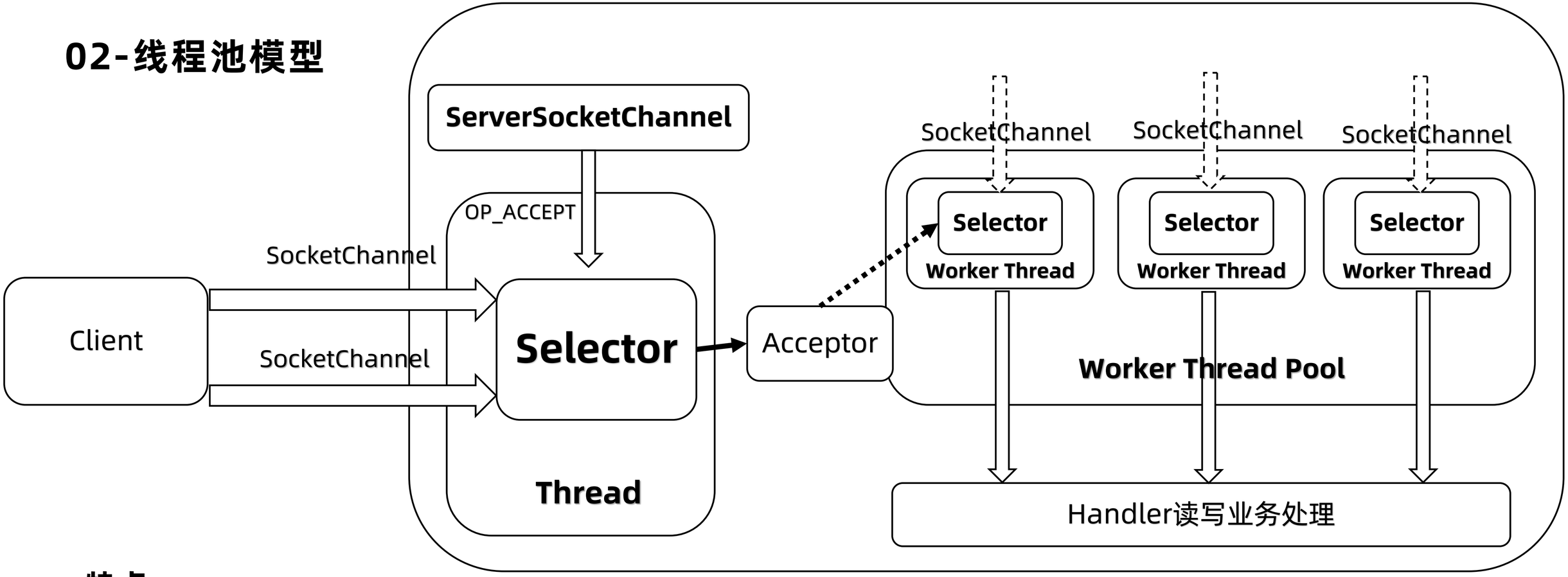


特点：

- 通过IO多路复用，一个线程搞定所有Client连接，代码简单，清晰明了
- 如果Client连接数量过多则无法支撑

3.2 线程模型

02-线程池模型

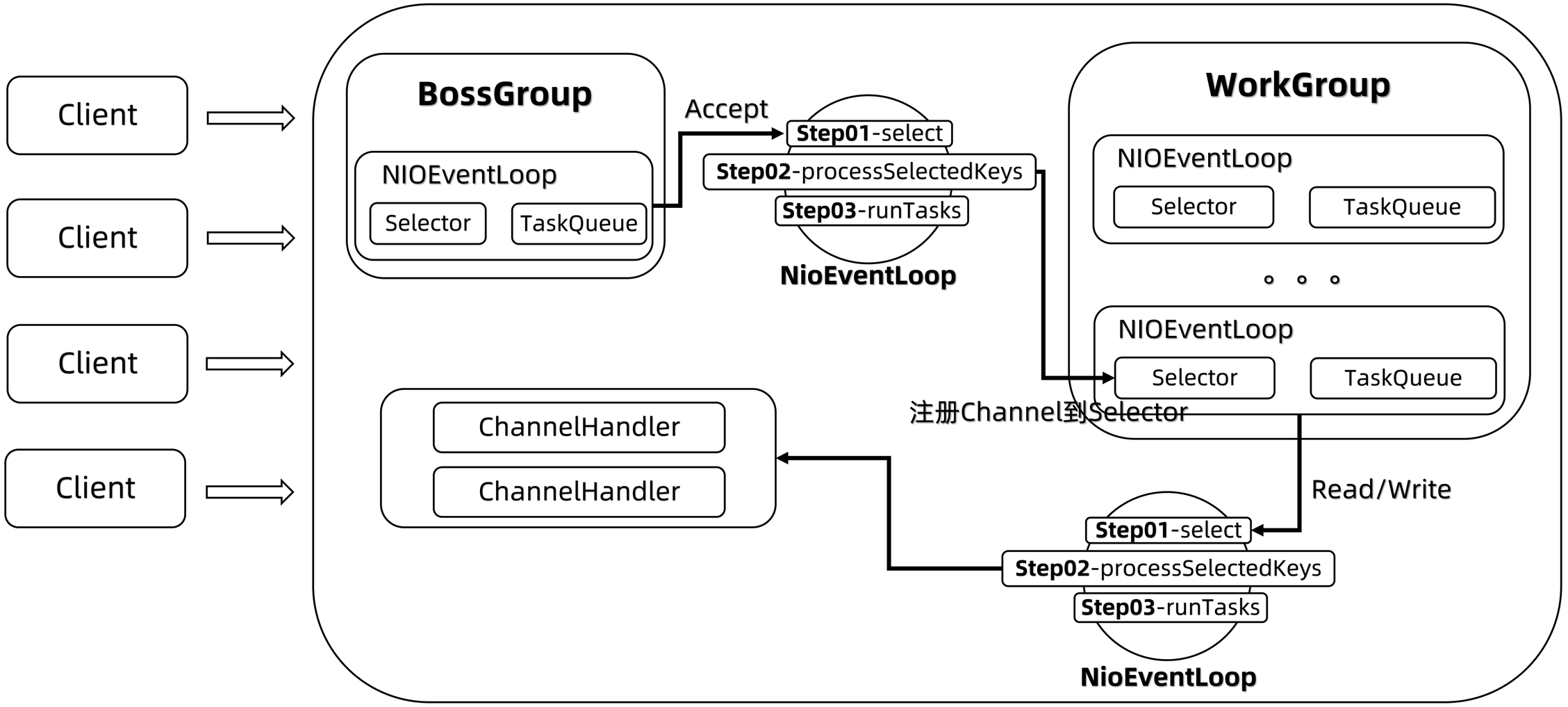


特点：

- 通过IO多路复用，一个线程专门负责连接，线程池负责处理请求
- 大多数场景下，此模型都能满足网络编程需求

3.2 线程模型

03-Netty线程模型



THANKS

教育不是注满一桶水，而是点燃一把火