极客大学

极客大学 Java 进阶训练营
第 4 课
NIO 模型与 Netty 入门

KimmKing

Apache Dubbo/ShardingSphere PMC

个人介绍

极客大学

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

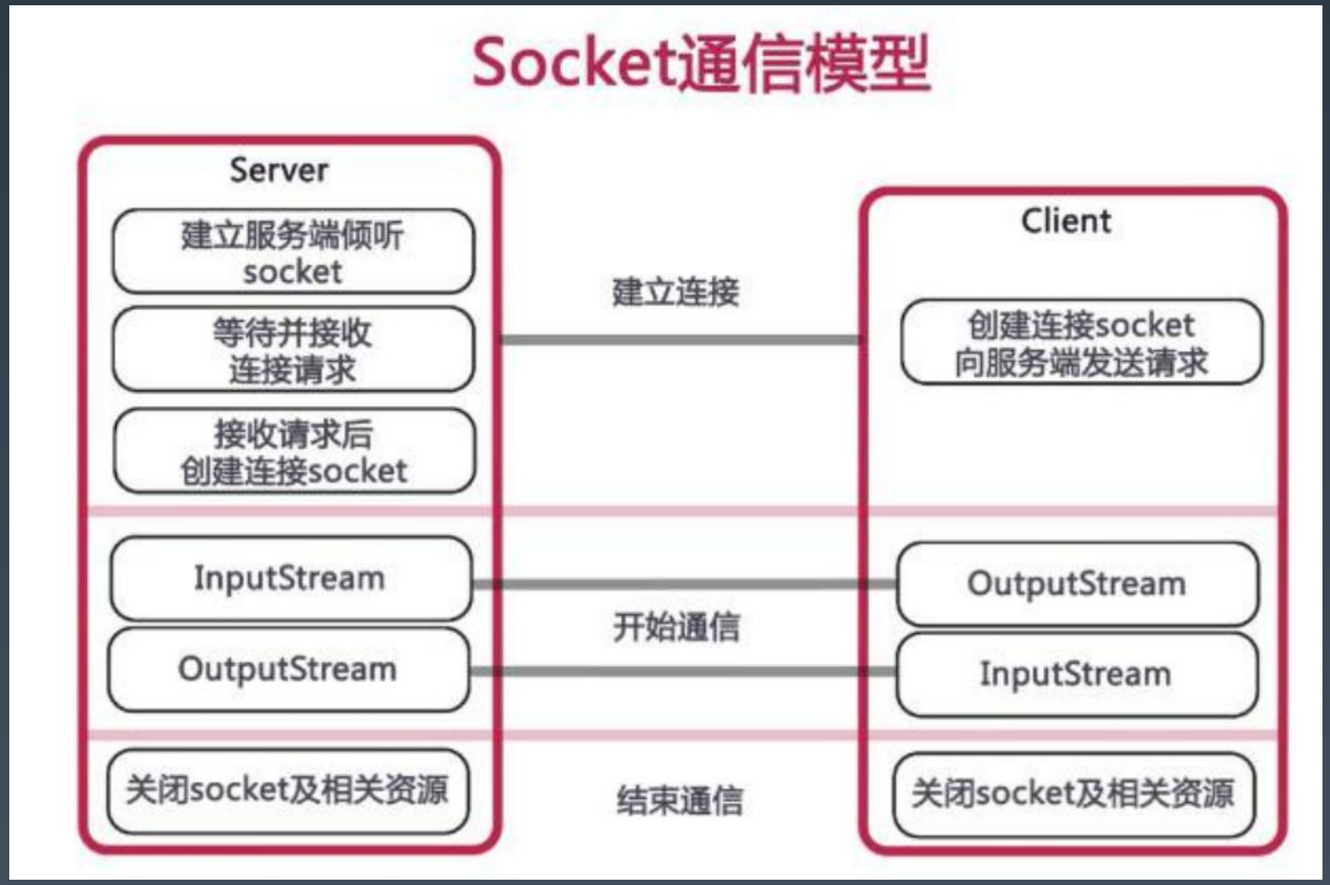
极客大学

目录

极客大学

1. Java Socket 编程

服务器通信原理


Socket通信模型
Server
建立服务端倾听socket
等待并接收连接请求
接收请求后创建连接socket
InputStream
OutputStream
关闭socket及相关资源
建立连接
开始通信
结束通信
Client
创建连接socket向服务端发送请求
OutputStream
InputStream
关闭socket及相关资源

Java 实现一个最简的 HTTP 服务器-01

```java
package java0.nio01;

import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class HttpServer01 {
    public static void main(String[] args) throws IOException{
        ServerSocket serverSocket = new ServerSocket( port: 8801);
        while (true) {
            try {
                Socket socket = serverSocket.accept();
                service(socket);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private static void service(Socket socket) {
        try {
            Thread.sleep( millis: 20);
            PrintWriter printWriter = new PrintWriter(socket.getOutputStream(), autoFlush: true);
            printWriter.println("HTTP/1.1 200 OK");
            printWriter.println("Content-Type:text/html;charset=utf-8");
            printWriter.println();
            printWriter.write( s: "hello,nio");
            printWriter.close();
            socket.close();
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

1. 创建一个 ServerSocket
2. 绑定8801端口
3. 当有客户端请求时通过 accept 方法拿到 Socket，进而可以进行处理
4. sleep 20ms，模拟业务操作(IO)
5. 模拟输出 HTTP 报文头和 hello
6. 关闭 socket

可以浏览器访问 http://localhost:8801

思考一下有什么问题?

Java 实现一个最简的 HTTP 服务器-01

```
D:\test>sb -u http://localhost:8801 -c 40 -N 30
Starting at 2020/10/24 0:45:36
[Press C to stop the test]
1463    (RPS: 41.8)
---------------Finished!----------------
Finished at 2020/10/24 0:46:11 (took 00:00:35.1011860)
1498    (RPS: 42.7)                      Status 200:    1498

RPS: 48.1 (requests/second)
Max: 906ms
Min: 69ms
Avg: 812.9ms

  50%    below 820ms
  60%    below 821ms
  70%    below 822ms
  80%    below 823ms
  90%    below 825ms
  95%    below 827ms
  98%    below 839ms
  99%    below 842ms
99.9%    below 893ms
```

设置-Xmx512 然后启动

压测：

sb -u http://localhost:8801 -c 40 -N 30

wrk -c 40 -d30s http://localhost:8801

Java 实现一个最简的 HTTP 服务器-02

```java
package java0.nio01;

import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class HttpServer02 {
    public static void main(String[] args) throws IOException{
        ServerSocket serverSocket = new ServerSocket( port: 8802);
        while (true) {
            try {
                final Socket socket = serverSocket.accept();
                new Thread(() -> {
                    service(socket);
                }).start();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private static void service(Socket socket) {
        try {
            Thread.sleep( millis: 20);
            PrintWriter printWriter = new PrintWriter(socket.getOutputStream(), autoFlush: true);
            printWriter.println("HTTP/1.1 200 OK");
            printWriter.println("Content-Type:text/html;charset=utf-8");
            printWriter.println();
            printWriter.write( s: "hello,nio");
            printWriter.close();
            socket.close();
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

改进一下，绑定8802端口

每个客户端请求进来时创建一个线程

有什么问题?

Java 实现一个最简的 HTTP 服务器-02

```
D:\test>sb -u http://localhost:8802 -c 40 -N 30
Starting at 2020/10/24 0:50:01
[Press C to stop the test]
46044   (RPS: 1303.3)
---------------Finished!----------------
Finished at 2020/10/24 0:50:36 (took 00:00:35.3862188)
46057   (RPS: 1303.6)                    Status 200:    45372
Status 303:    685

RPS: 1484.1 (requests/second)
Max: 160ms
Min: 19ms
Avg: 22.7ms

  50%   below 21ms
  60%   below 22ms
  70%   below 22ms
  80%   below 23ms
  90%   below 25ms
  95%   below 30ms
  98%   below 43ms
  99%   below 47ms
99.9%   below 122ms
```

设置-Xmx512 然后启动

压测：

sb -u http://localhost:8802 -c 40 -N 30

wrk -c 40 -d30s http://localhost:8802

Java 实现一个最简的 HTTP 服务器-03

```java
package java0.nio01;

import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class HttpServer03 {
    public static void main(String[] args) throws IOException{
        ExecutorService executorService = Executors.newFixedThreadPool( nThreads: 40);
        final ServerSocket serverSocket = new ServerSocket( port: 8803);
        while (true) {
            try {
                final Socket socket = serverSocket.accept();
                executorService.execute(() -> service(socket));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private static void service(Socket socket) {
        try {
            Thread.sleep( millis: 20);
            PrintWriter printWriter = new PrintWriter(socket.getOutputStream(), autoFlush: true);
            printWriter.println("HTTP/1.1 200 OK");
            printWriter.println("Content-Type:text/html;charset=utf-8");
            printWriter.println();
            printWriter.write( s: "hello,nio");
            printWriter.close();
            socket.close();
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

再改进一下，绑定8802端口

创建一个固定大小的线程池来处理

why?

有什么问题?

Java 实现一个最简的 HTTP 服务器-02

```
D:\test>sb -u http://localhost:8803 -c 40 -N 30
Starting at 2020/10/24 0:52:39
[Press C to stop the test]
47965   (RPS: 1373.1)
---------------Finished!----------------
Finished at 2020/10/24 0:53:14 (took 00:00:34.9814638)
47995   (RPS: 1373.9)                    Status 200:    47498
Status 303:    497

RPS: 1547.5 (requests/second)
Max: 162ms
Min: 19ms
Avg: 21.8ms

  50%   below 21ms
  60%   below 21ms
  70%   below 21ms
  80%   below 21ms
  90%   below 23ms
  95%   below 27ms
  98%   below 42ms
  99%   below 45ms
99.9%   below 104ms
```

设置-Xmx512 然后启动

压测：

sb -u http://localhost:8803 -c 40 -N 30

wrk -c 40 -d30s http://localhost:8803

总结一下，到目前为止，我们做了什么

类似于饭店的服务员

单线程处理 socket

每个请求一个线程

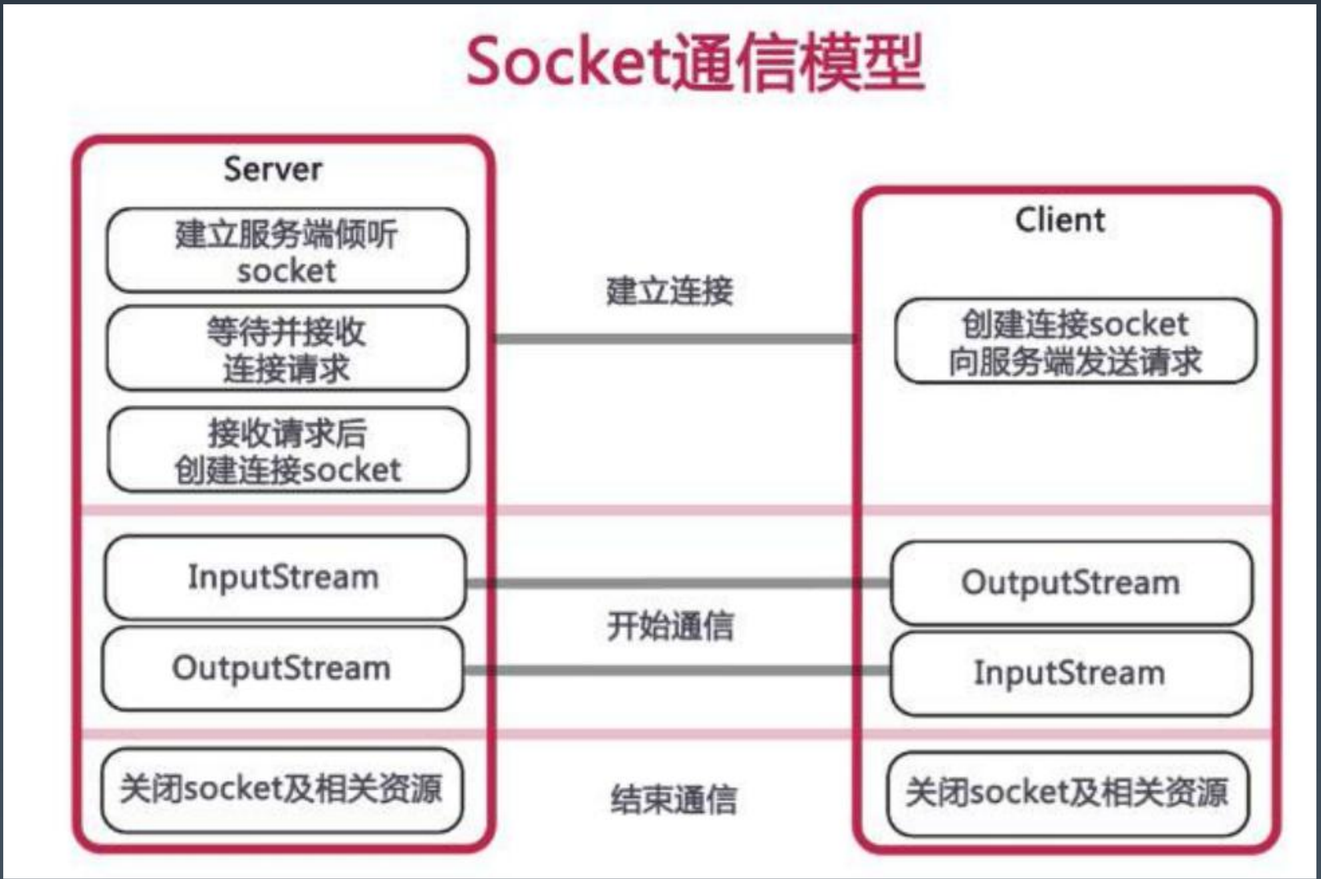
固定大小线程池处理

让我们来对比一下GC

极客大学

2. 深入讨论 IO 通信

服务器通信过程分析



仔细分析一下，

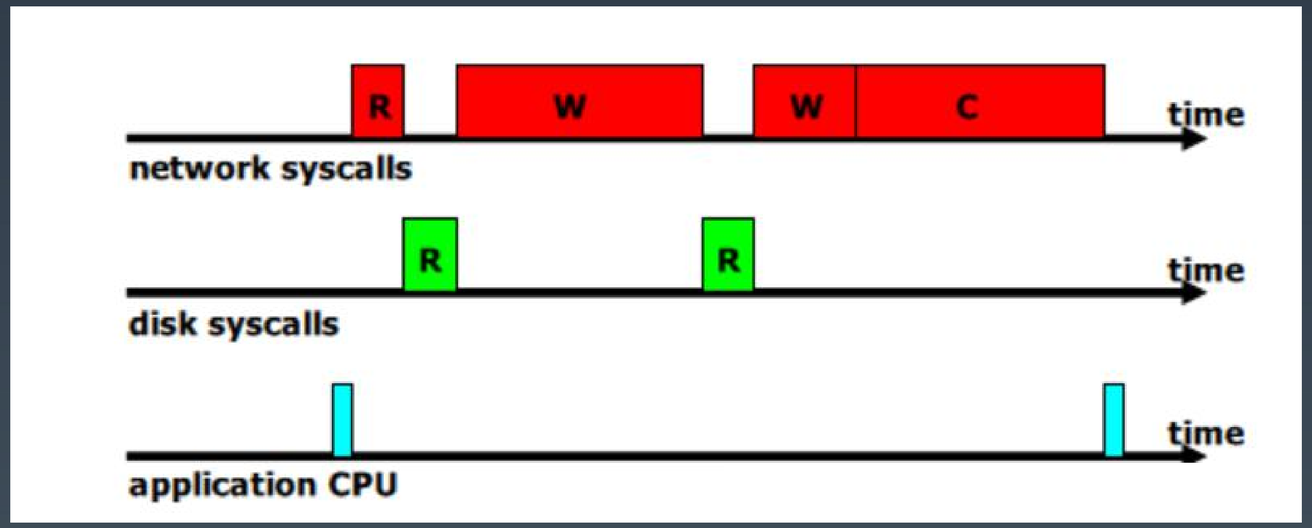
这个过程中，存在两种类型操作：

– CPU 计算/业务处理

– IO 操作与等待/网络、磁盘、数据库

想想我们前面的例子为什么创建大量线程?

服务器通信过程分析



对于一个 IO 相关应用来说，

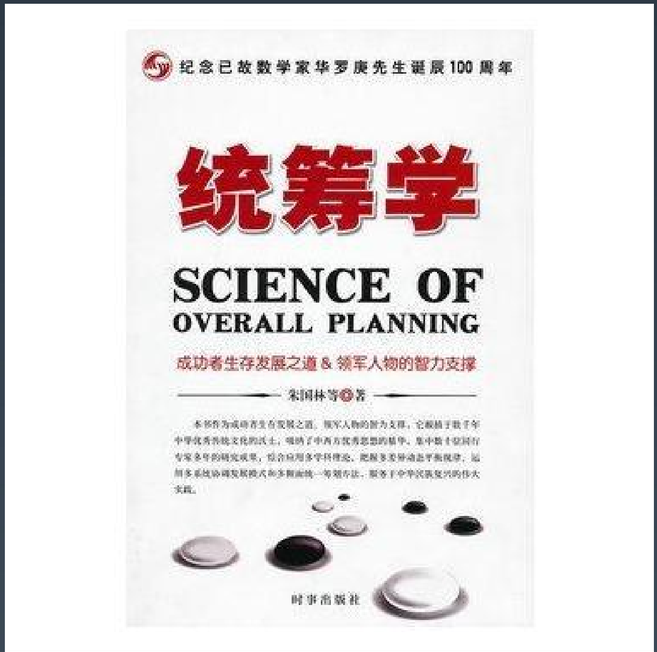
例如通过网络访问，服务器端读取本地文件，再返回给客户端（如左图）。

这种情况下，

大部分 CPU 等资源，可能就被浪费了。

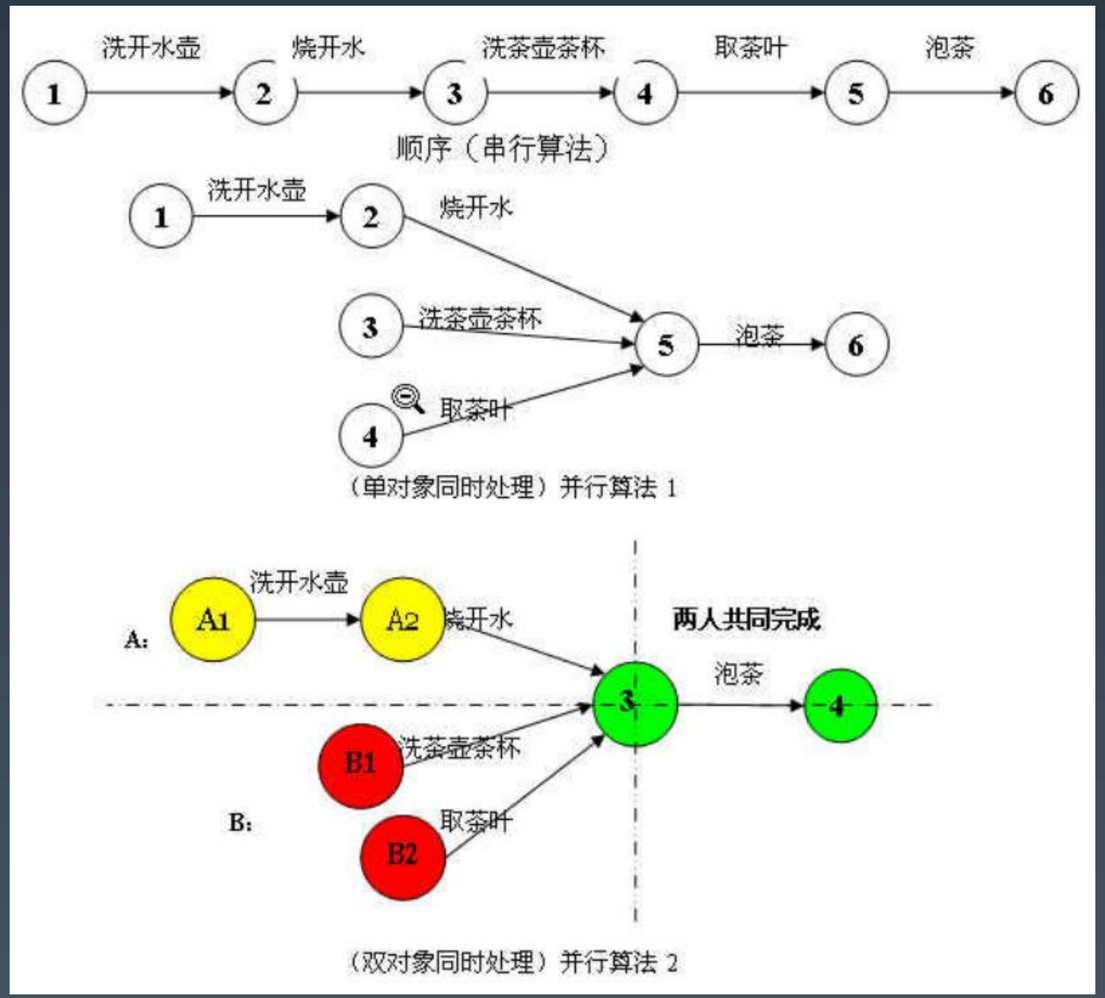
怎么优化呢?

服务器通信过程分析

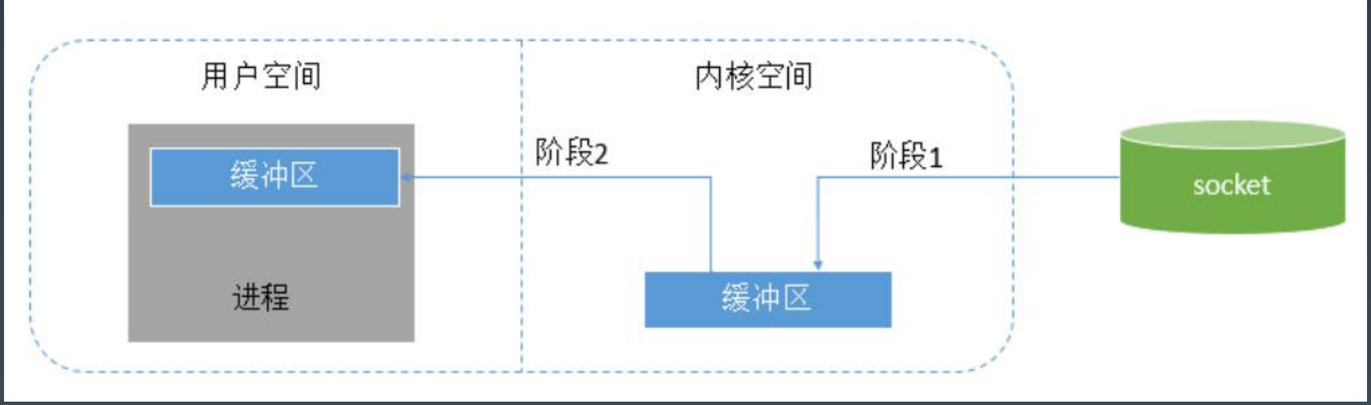


回忆一下GC的并发收集，

GC的时候，同时可以不影响业务线程



再深入一层的看问题



不仅面临线程 /CPU 的问题，

还要面对数据来回复制的问题。

这个一来，对每个业务处理过程，使用一个线程以一竿子通到底的方式，性能不是最优的，还有提升空间。

考虑一下，理想状态，是什么样的?

流水线

极客大学

3. NIO 模型与相关概念

通信模型

考虑一下：

阻塞、非阻塞，
同步、异步，

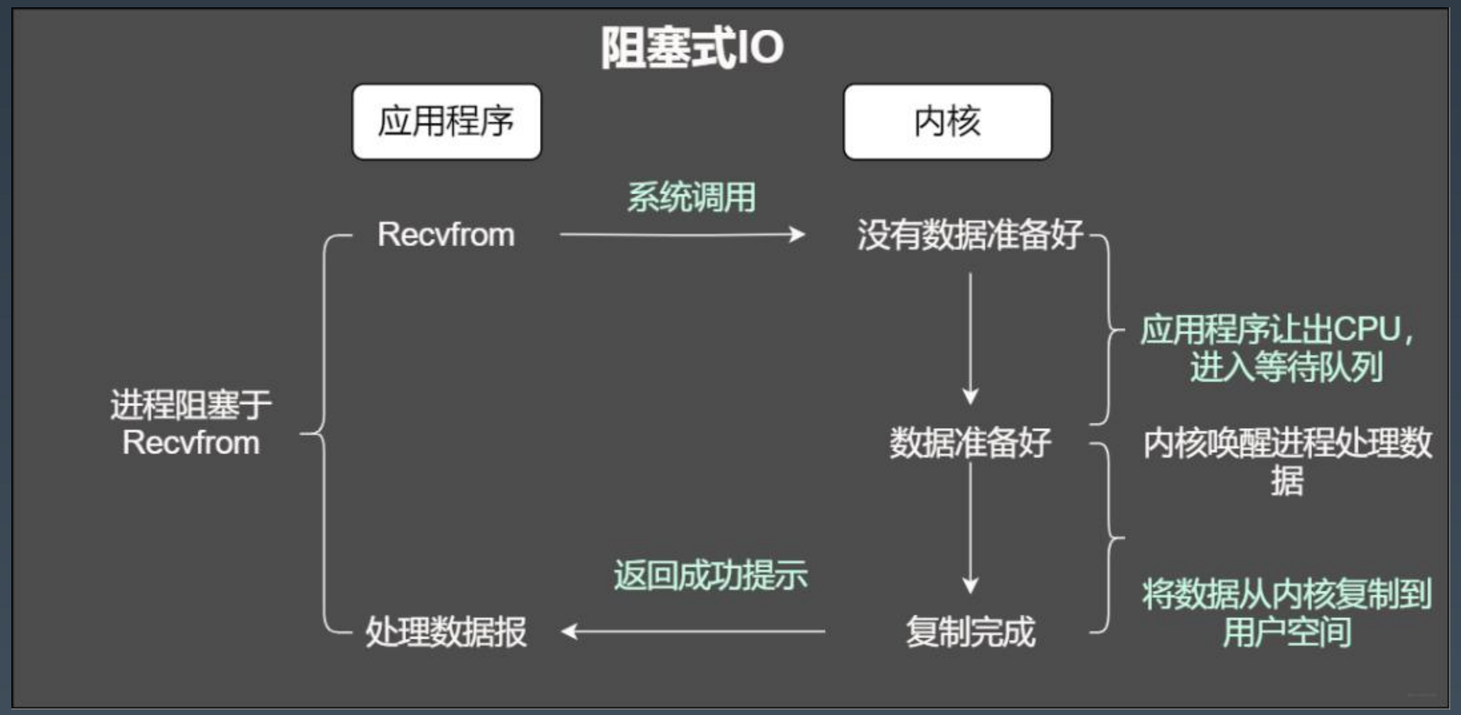
有什么关系和区别?

– 同步异步 是通信模式。
– 阻塞、非阻塞 是 线程处理模式。

五种 IO 模型

基本上都是同步的

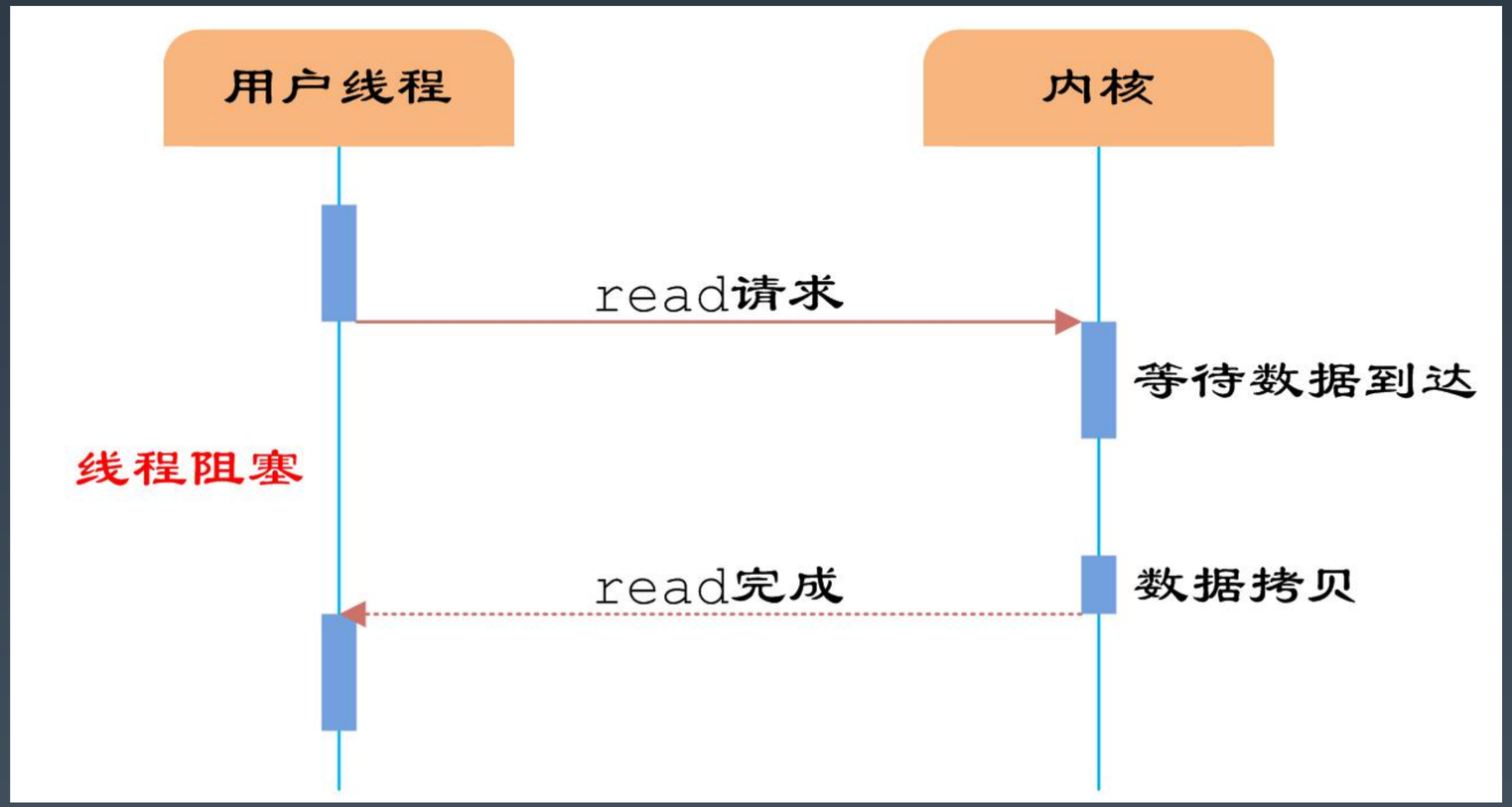
IO 模型-01



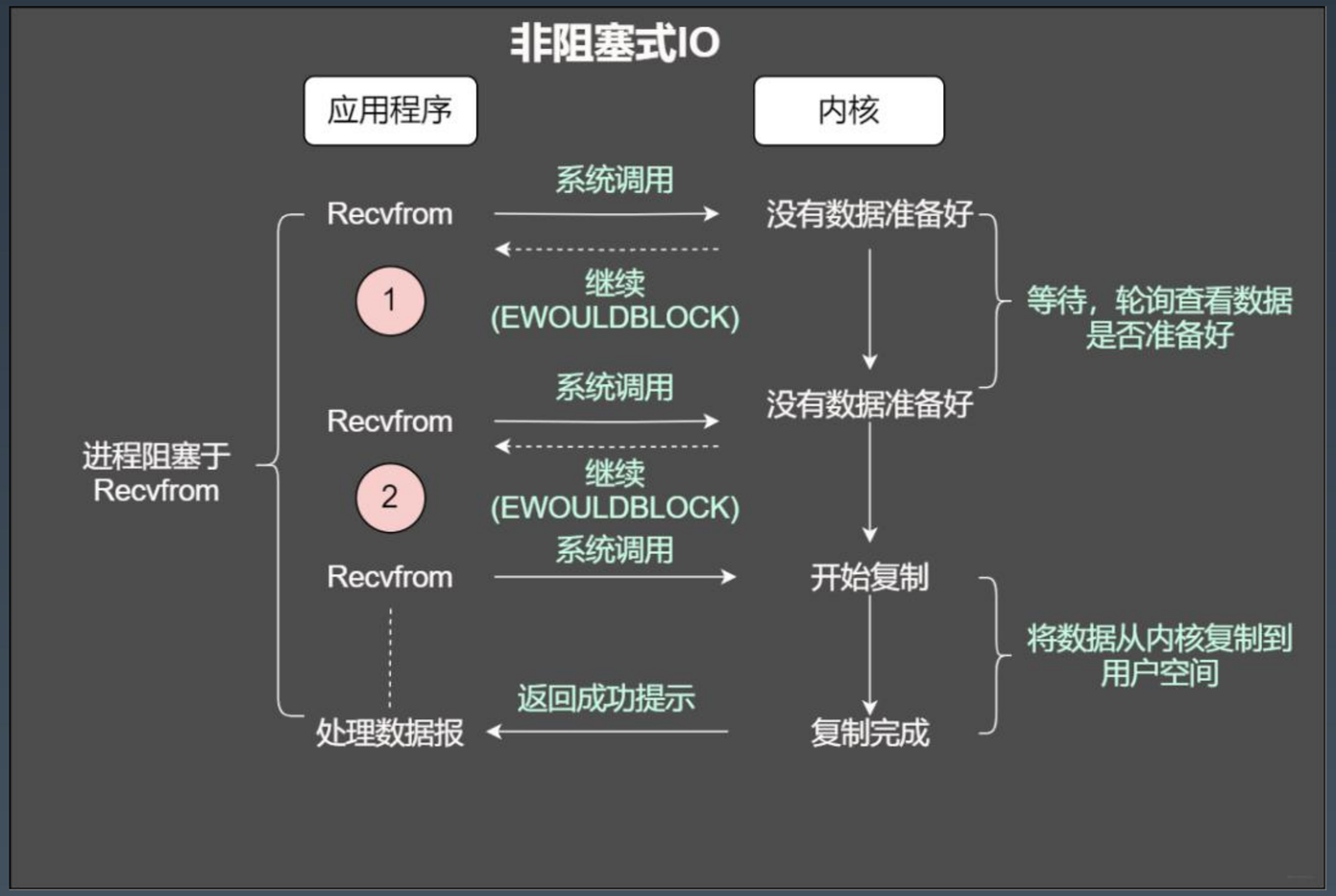
阻塞式 IO、BIO

一般通过在 while(true) 循环中服务端会调用 accept() 方法等待接收客户端的连接的方式监听请求，请求一旦接收到一个连接请求，就可以建立通信套接字在这个通信套接字上进行读写操作，此时不能再接收其他客户端连接请求，只能等待同当前连接的客户端的操作执行完成， 不过可以通过多线程来支持多个客户端的连接

IO 模型-01



IO 模型-02

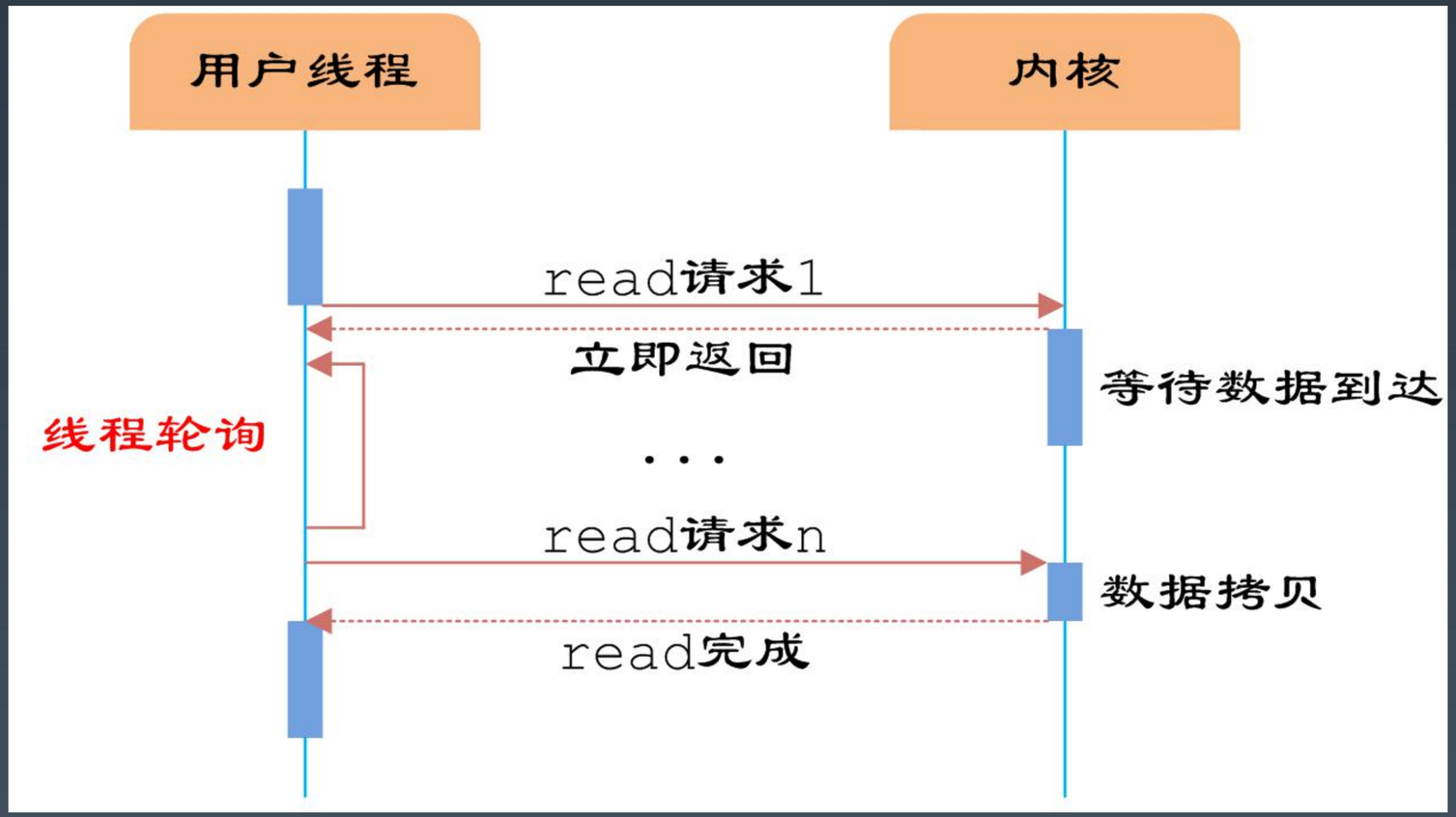


非阻塞式 IO

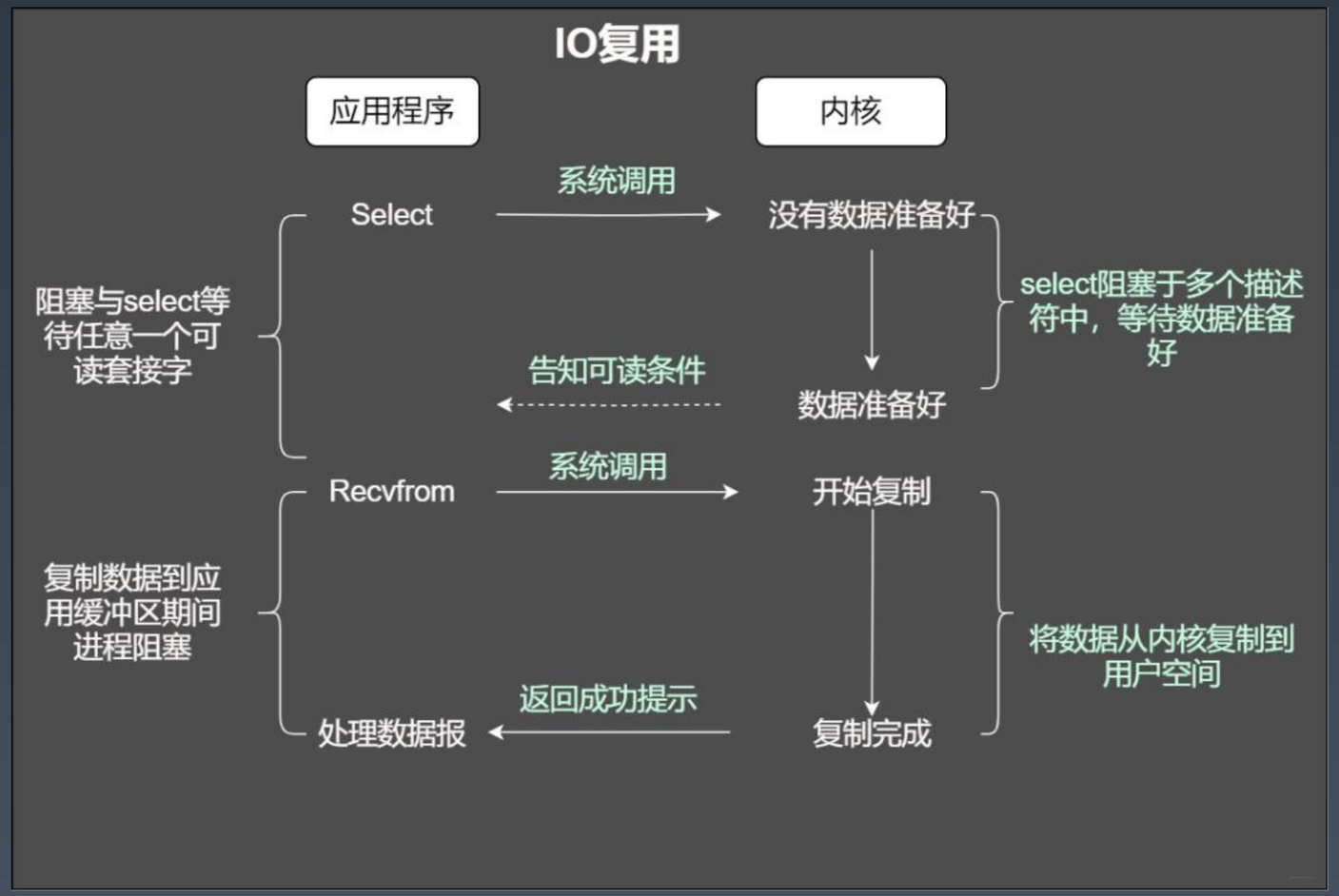
和阻塞 IO 类比，内核会立即返回，返回后获得足够的 CPU 时间继续做其它的事情。

用户进程第一个阶段不是阻塞的,需要不断的主动询问 kernel 数据好了没有；第二个阶段依然总是阻塞的。

IO 模型-02



IO 模型–03

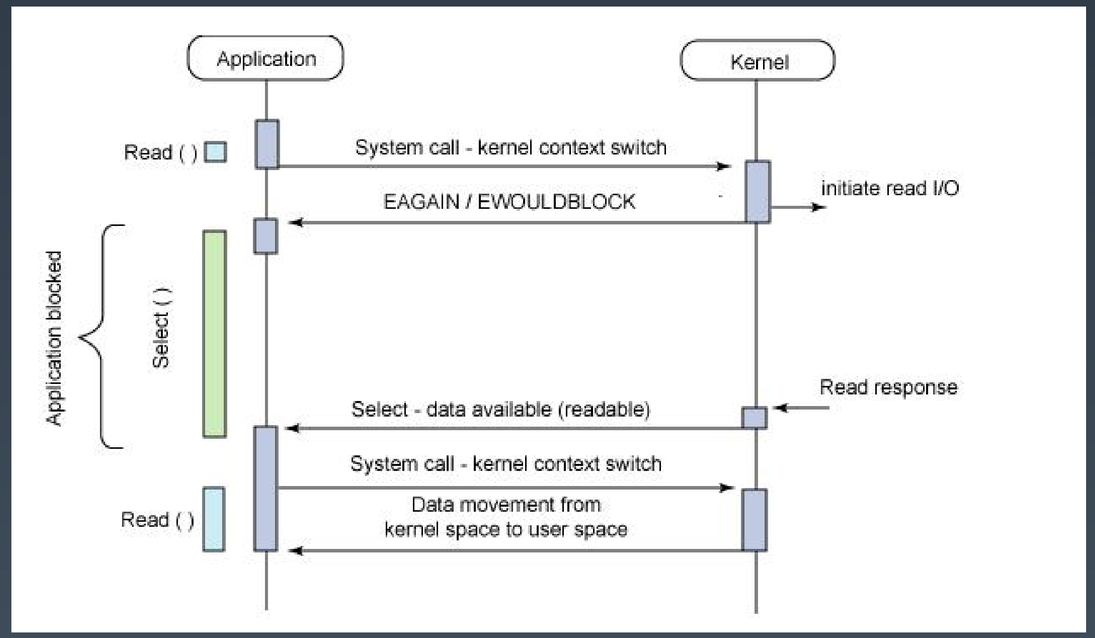


IO 多路复用(IO multiplexing)，也称事件驱动 IO(event-driven IO)，就是在单个线程里同时监控多个套接字，通过 select 或 poll 轮询所负责的所有 socket，当某个 socket 有数据到达了，就通知用户进程。

IO 复用同非阻塞 IO 本质一样，不过利用了新的 select 系统调用，由内核来负责本来是请求进程该做的轮询操作。看似比非阻塞 IO 还多了一个系统调用开销，不过因为可以支持多路 IO，才算提高了效率。

进程先是阻塞在 select/poll 上，再是阻塞在读操作的第二个阶段上。

IO 模型–03



select/poll 的几大缺点：

（1）每次调用 select，都需要把 fd 集合从用户态拷贝到内核态，这个开销在 fd 很多时会很大

（2）同时每次调用 select 都需要在内核遍历传递进来的所有 fd，这个开销在 fd 很多时也很大

（3）select 支持的文件描述符数量太小了，默认是1024

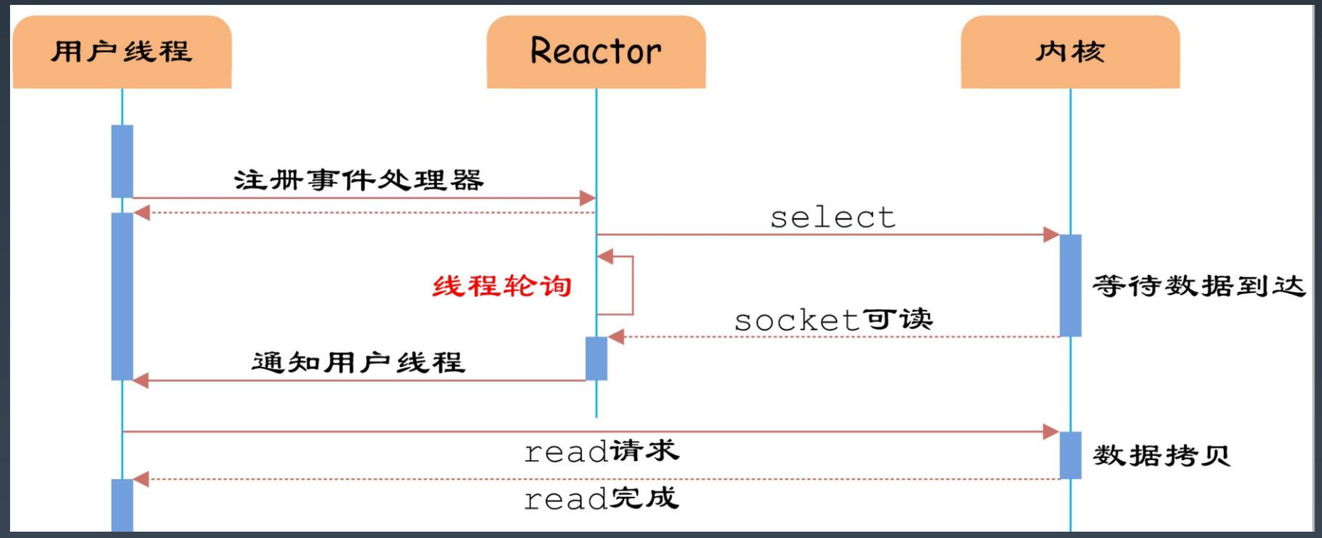
epoll（Linux 2.5.44内核中引入,2.6内核正式引入,可被用于代替 POSIX select 和 poll 系统调用）：

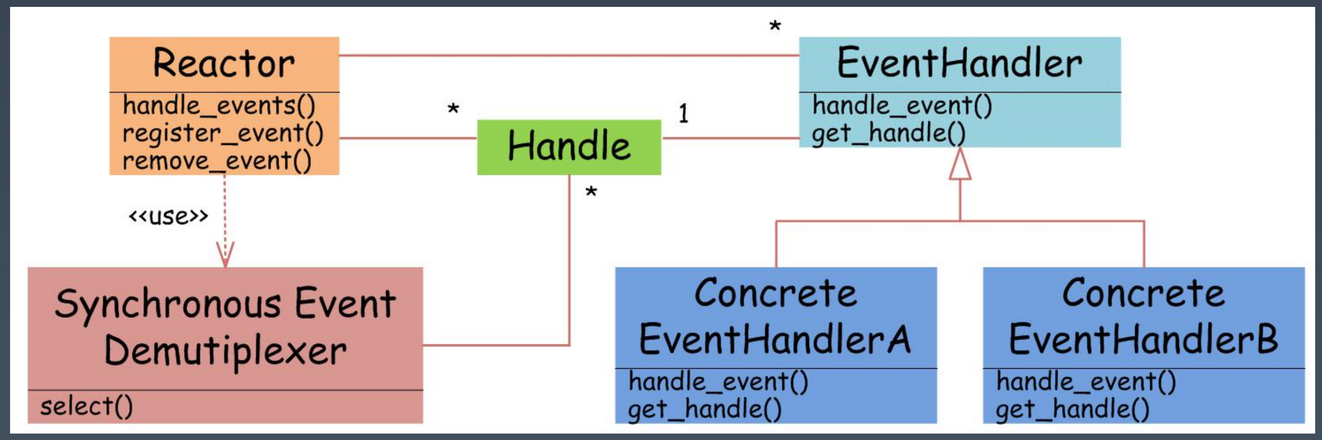
（1）内核与用户空间共享一块内存

（2）通过回调解决遍历问题

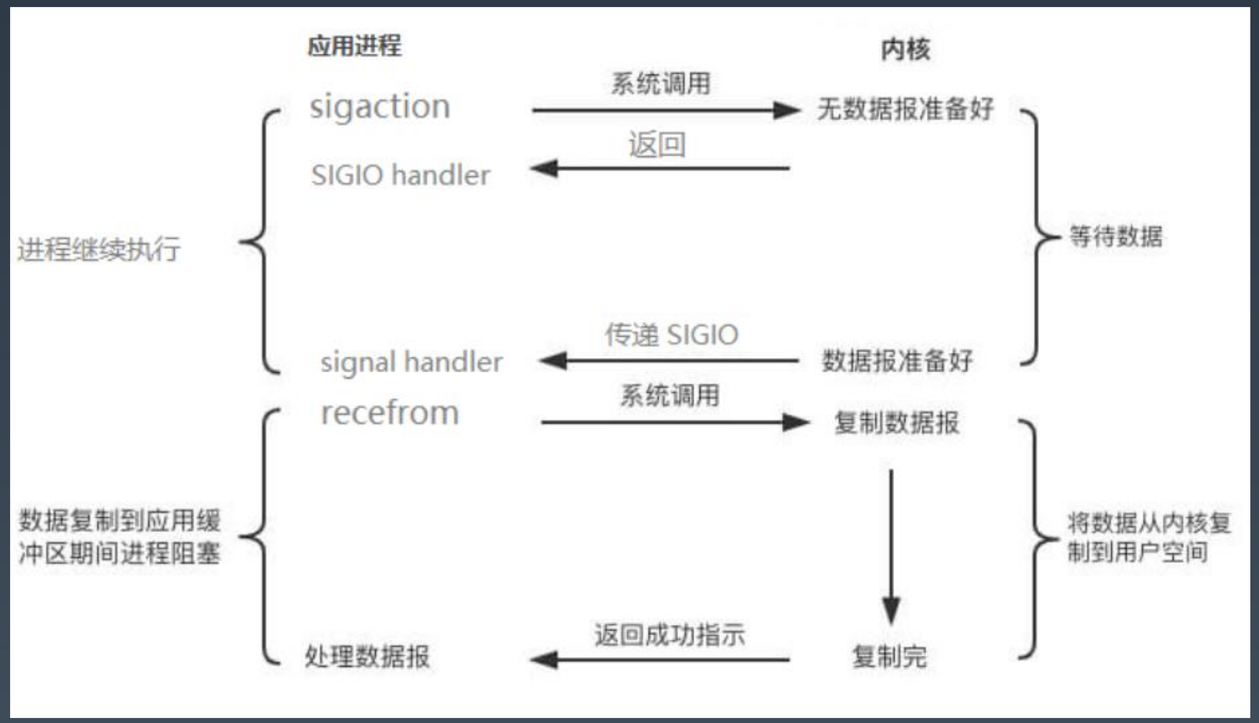
（3）fd 没有限制，可以支撑10万连接

IO 模型-03


用户线程
Reactor
内核
注册事件处理器
select
线程轮询
等待数据到达
socket可读
通知用户线程
read请求
数据拷贝
read完成


Reactor
handle_events()
register_event()
remove_event()
*
Handle
1
*
EventHandler
handle_event()
get_handle()
*
<<use>>
Synchronous Event Demutiplexer
select()
Concrete EventHandlerA
handle_event()
get_handle()
Concrete EventHandlerB
handle_event()
get_handle()

IO 模型-04

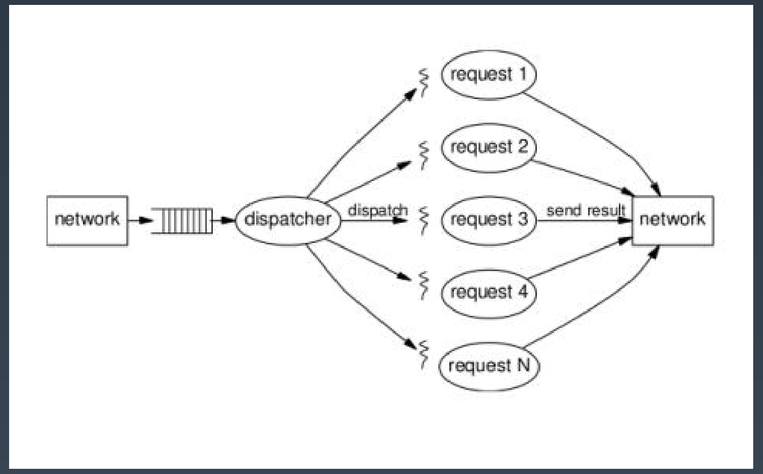


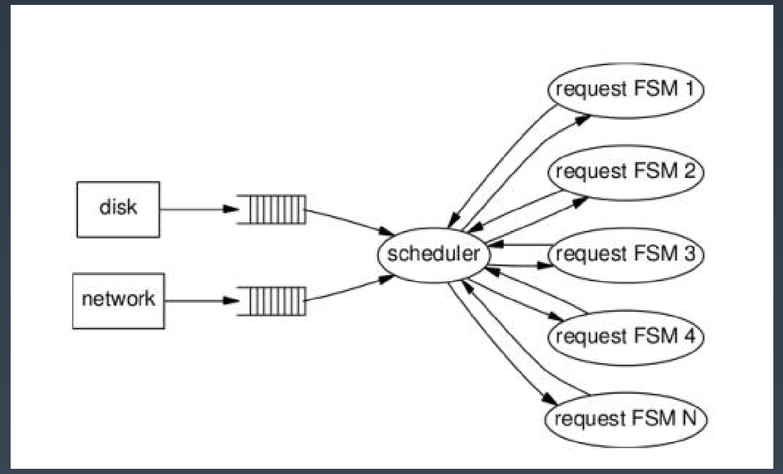
信号驱动 I/O

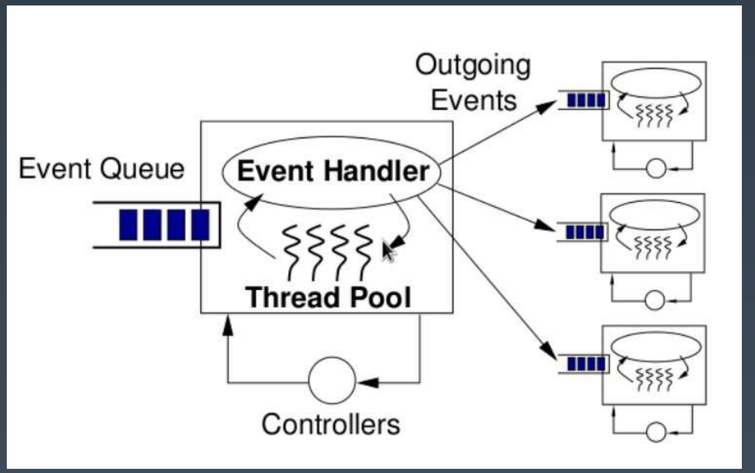
信号驱动 IO 与 BIO 和 NIO 最大的区别就在于，在 IO 执行的数据准备阶段，不会阻塞用户进程。

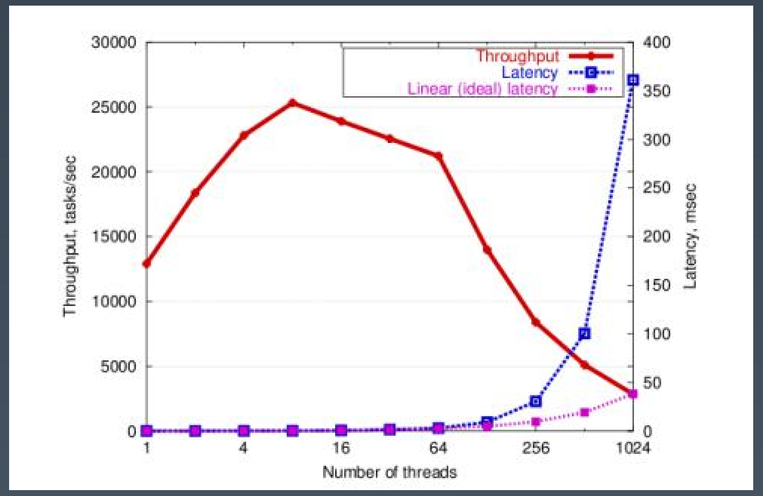
如图所示：当用户进程需要等待数据的时候，会向内核发送一个信号，告诉内核我要什么数据，然后用户进程就继续做别的事情去了，而当内核中的数据准备好之后，内核立马发给用户进程一个信号，说”数据准备好了，快来查收“，用户进程收到信号之后，立马调用 recvfrom，去查收数据。

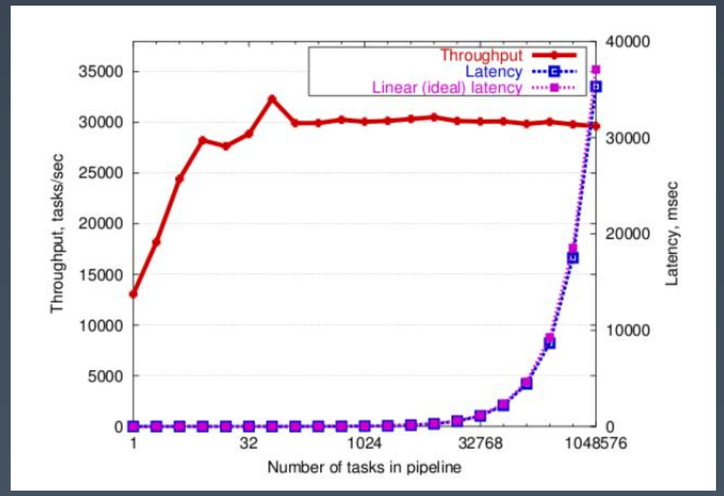
IO 模型-04(线程池->EDA->SEDA)



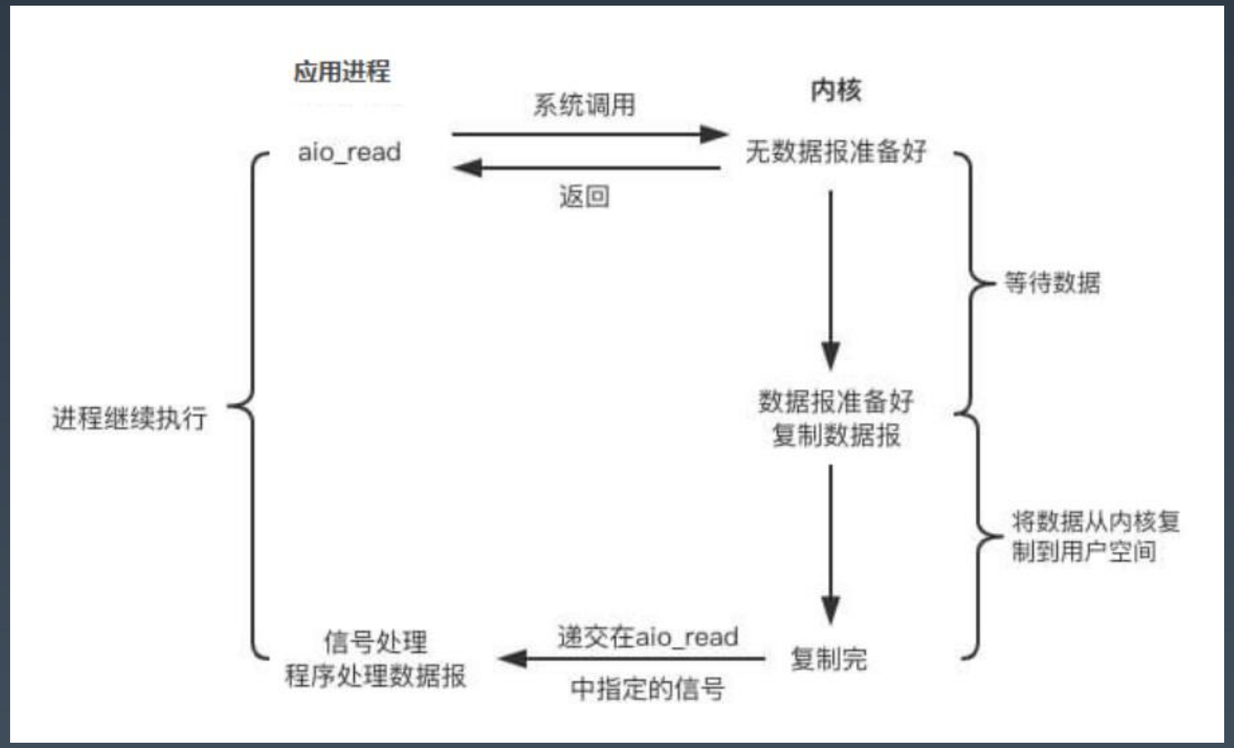








IO 模型-05

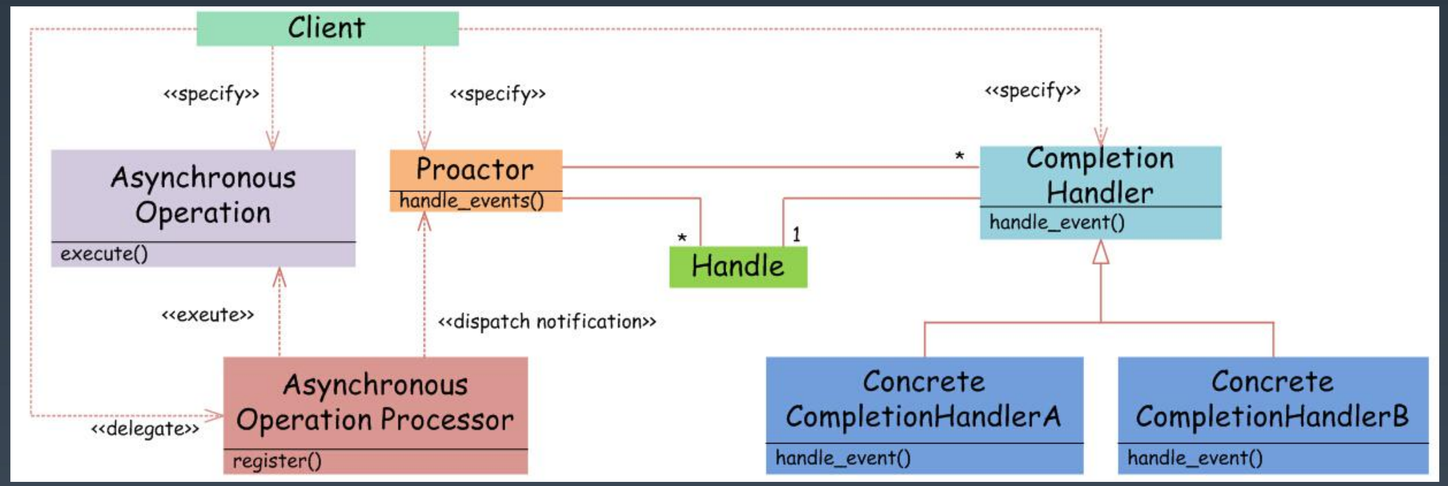


异步式 IO

异步 IO 真正实现了 IO 全流程的非阻塞。用户进程发出系统调用后立即返回，内核等待数据准备完成，然后将数据拷贝到用户进程缓冲区，然后发送信号告诉用户进程 IO 操作执行完毕（与 SIGIO 相比，一个是发送信号告诉用户进程数据准备完毕，一个是IO执行完毕）。

windows 的 IOCP 模型

IO 模型-05



EPoll	Linux	2002
kqueue	FreeBsd（MAC）	2000
iocp	Windows	1993

一个场景，去打印店打印文件。

- 同步阻塞

  直接排队，别的啥也干不成，直到轮到你使用打印机了，自己打印文件

- Reactor

  拿个号码，回去该干嘛干嘛，等轮到你使用打印机了，店主通知你来用打印机，打印文件

- Proactor

  拿个号码，回去该干嘛干嘛，等轮到你使用打印机了，店主直接给你打印好文件，通知你来拿。

极客大学

4. Netty 框架简介

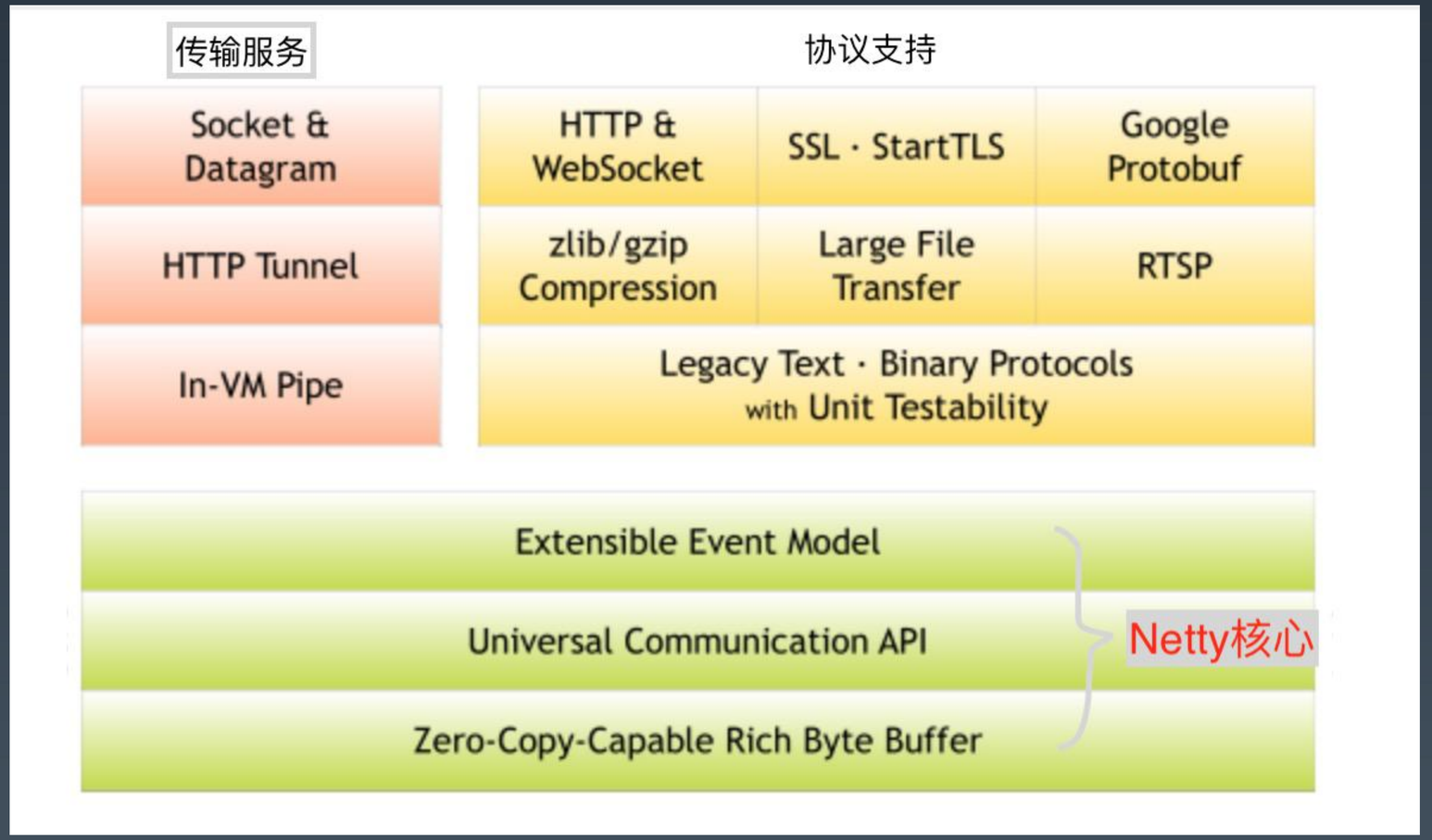
Netty 概览

网络应用开发框架

1. 异步
2. 事件驱动
3. 基于 NIO

适用于:

- 服务端
- 客户端
- TCP/UDP



Netty官网: netty.io

Netty 特性

高性能的协议服务器:

- 高吞吐
- 低延迟
- 低开销
- 零拷贝
- 可扩容

- 松耦合: 网络和业务逻辑分离
- 使用方便、可维护性好

兼容性

JDK 兼容性:

- Netty 3.x: JDK5
- Netty 4.x: JDK6
- ~~Netty 5.x: 已废弃~~

协议兼容性:

- 兼容大部分通用协议
- 支持自定义协议

嵌入式:

- HTTP Server
- HTTPS Server
- WebSocket Server
- TCP Server
- UDP Server
- In VM Pipe

Netty vs. Java EE?

基本概念

- **Channel**
  通道，Java NIO 中的基础概念,代表一个打开的连接,可执行读取/写入 IO 操作。
  Netty 对 Channel 的所有 IO 操作都是非阻塞的。
- **ChannelFuture**
  Java 的 Future 接口，只能查询操作的完成情况, 或者阻塞当前线程等待操作完成。Netty 封装一个 ChannelFuture 接口。
  我们可以将回调方法传给 ChannelFuture，在操作完成时自动执行。
- **Event & Handler**
  Netty 基于事件驱动，事件和处理器可以关联到入站和出站数据流。
- **Encoder & Decoder**
  处理网络 IO 时，需要进行序列化和反序列化, 转换 Java 对象与字节流。
  对入站数据进行解码, 基类是 ByteToMessageDecoder。
  对出站数据进行编码, 基类是 MessageToByteEncoder。
- ChannelPipeline
  数据处理管道就是事件处理器链。
  有顺序、同一 Channel 的出站处理器和入站处理器在同一个列表中。

Event & Handler

入站事件：

- 通道激活和停用
- 读操作事件
- 异常事件
- 用户事件

出站事件：

- 打开连接
- 关闭连接
- 写入数据
- 刷新数据

事件处理程序接口：

- ChannelHandler
- ChannelOutboundHandler
- ChannelInboundHandler

适配器（空实现，需要继承使用）：

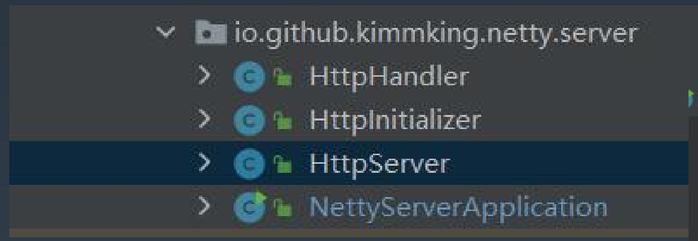
- ChannelInboundHandlerAdapter
- ChannelOutboundHandlerAdapter

Netty 应用组成:

- 网络事件
- 应用程序逻辑事件
- 事件处理程序

5. Netty 使用示例

demo




```java
public void run() throws Exception {
    final SslContext sslCtx;
    if (ssl) {
        SelfSignedCertificate ssc = new SelfSignedCertificate();
        sslCtx = SslContext.newServerContext(ssc.certificate(), ssc.privateKey());
    } else {
        sslCtx = null;
    }

    EventLoopGroup bossGroup = new NioEventLoopGroup( nEventLoops: 3);
    EventLoopGroup workerGroup = new NioEventLoopGroup( nEventLoops: 1000);

    try {
        ServerBootstrap b = new ServerBootstrap();
        b.option(ChannelOption.SO_BACKLOG, value: 128)
                .option(ChannelOption.TCP_NODELAY, value: true)
                .option(ChannelOption.SO_KEEPALIVE, value: true)
                .option(ChannelOption.SO_REUSEADDR, value: true)
                .option(ChannelOption.SO_RCVBUF, value: 32 * 1024)
                .option(ChannelOption.SO_SNDBUF, value: 32 * 1024)
                .option(EpollChannelOption.SO_REUSEPORT, value: true)
                .childOption(ChannelOption.SO_KEEPALIVE, value: true);
                //.option(ChannelOption.ALLOCATOR, PooledByteBufAllocator.DEFAULT);

        b.group(bossGroup, workerGroup).channel(NioServerSocketChannel.class)
                .handler(new LoggingHandler(LogLevel.INFO)).childHandler(new HttpInitializer(sslCtx));

        Channel ch = b.bind(port).sync().channel();
        logger.info("开启netty http服务器，监听地址和端口为 " + (ssl ? "https" : "http") + "://127.0.0.1:" + port + '/');
        ch.closeFuture().sync();
    } finally {
        bossGroup.shutdownGracefully();
        workerGroup.shutdownGracefully();
    }
}
```

NettyServerApplication.java | HttpServer.java | HttpInitializer.java | HttpHandler.java | pom.xml (netty-server)

```java
public class HttpHandler extends ChannelInboundHandlerAdapter {

    private static Logger logger = LoggerFactory.getLogger(HttpHandler.class);

    @Override
    public void channelReadComplete(ChannelHandlerContext ctx) { ctx.flush(); }

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) {
        try {
            //logger.info("channelRead流量接口请求开始，时间为{}", startTime);
            FullHttpRequest fullRequest = (FullHttpRequest) msg;
            String uri = fullRequest.uri();
            //logger.info("接收到的请求url为{}", uri);
            if (uri.contains("/test")) {
                handlerTest(fullRequest, ctx);
            }
        } finally {
            ReferenceCountUtil.release(msg);
        }
    }

    private void handlerTest(FullHttpRequest fullRequest, ChannelHandlerContext ctx) {
        FullHttpResponse response = null;
        try {
            String value = "hello,kimmking";
            response = new DefaultFullHttpResponse(HTTP_1_1, OK, Unpooled.wrappedBuffer(value.getBytes( charsetName: "UTF-8")));
            response.headers().set("Content-Type", "application/json");
            response.headers().setInt( name: "Content-Length", response.content().readableBytes());

        } catch (Exception e) {
            logger.error("处理测试接口出错", e);
            response = new DefaultFullHttpResponse(HTTP_1_1, NO_CONTENT);
        } finally {
            if (fullRequest != null) {
                if (!HttpUtil.isKeepAlive(fullRequest)) {
                    ctx.write(response).addListener(ChannelFutureListener.CLOSE);
                } else {
                    response.headers().set(CONNECTION, KEEP_ALIVE);
                    ctx.write(response);
                }
            }
        }
    }
```

Netty 简单例子

使用 Netty 改写

最开始的例子

然后，压测一下效果如何。

```
D:\test>sb -u http://localhost:8808/test -c 40 -N 30
Starting at 2020/10/24 2:45:05
[Press C to stop the test]
96207   (RPS: 2740.6)
---------------Finished!----------------
Finished at 2020/10/24 2:45:40 (took 00:00:35.3234921)
Status 200:    96232

RPS: 3086 (requests/second)
Max: 363ms
Min: 0ms
Avg: 0.9ms

  50%   below 0ms
  60%   below 0ms
  70%   below 0ms
  80%   below 0ms
  90%   below 0ms
  95%   below 5ms
  98%   below 12ms
  99%   below 18ms
99.9%   below 76ms
```

极客大学

6. 总结回顾与作业实践

第四节课总结回顾

极客大学

Java Socket 编程

IO 处理过程分析

IO 模型与 NIO

Netty 介绍与示例

第四节课作业实践

1、（可选）运行课上的例子，以及 Netty 的例子，分析相关现象。

2、（必做）写一段代码，使用 HttpClient 或 OkHttp 访问 http://localhost:8801，代码提交到Github。

THANKS!