架构实战营模块8 - 第1课

单机高性能网络模型

李运华

前阿里资深技术专家（P9）

教学目标

极客时间 训练营

1. 掌握单机高性能计算的网络模型

知道哪个轮子好就够了，千万别自己造轮子！

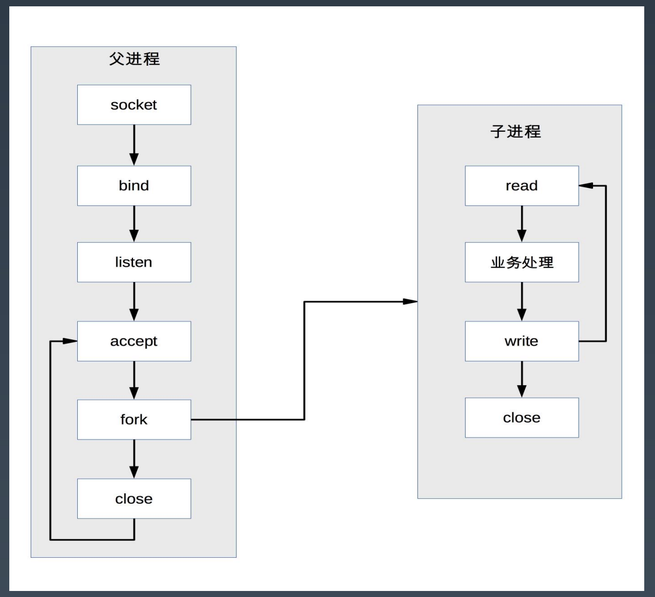
极客时间训练营

目录

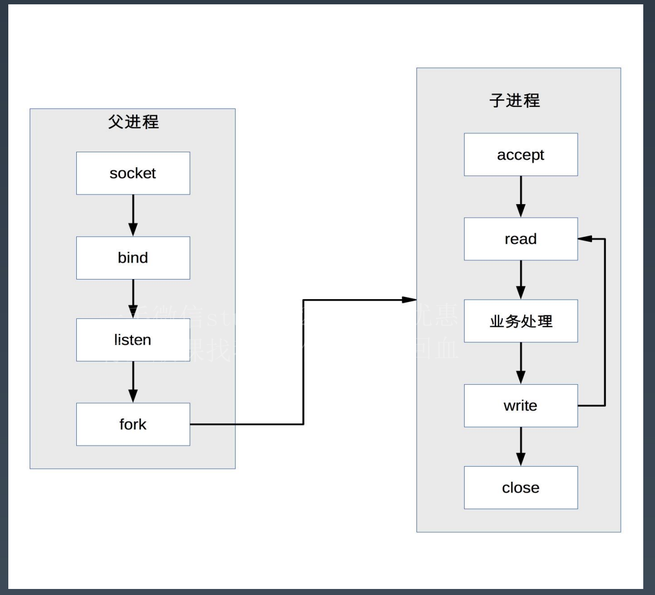
1. 传统网络模型
2. Reactor 网络模型
3. Proactor 网络模型
4. 网络模型对比

1. 传统网络模型

PPC 和 prefork 示意



PPC：Process per connection



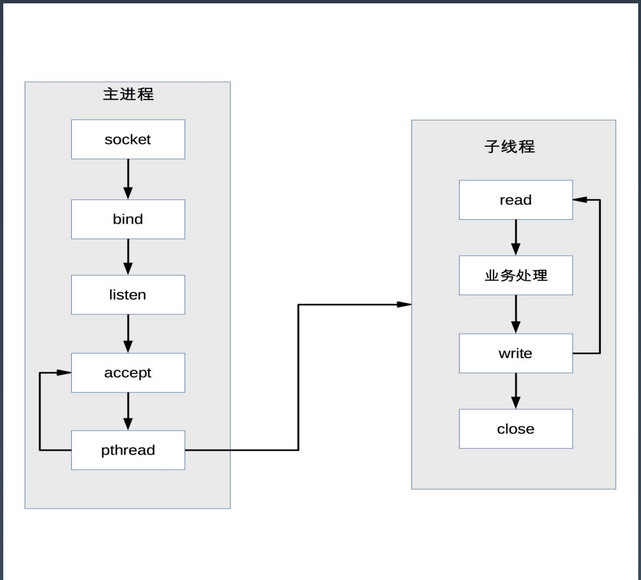
prefork：processes are forked before connection

【优点】
实现简单

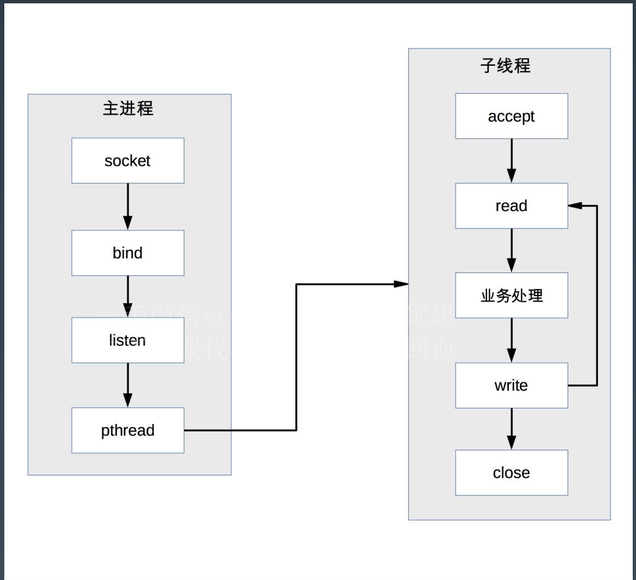
【缺点】
1. PPC：fork 代价高，性能低；
2. 父子进程通信要用 IPC；
3. OS 的上下文切换会限制并发连接数，一般几百。

【案例】
1. 世界上第一个 web 服务器 CERN httpd 采用 PPC 模式。参考链接
2. Apache MPM prefork 模式，默认256个连接。参考链接

TPC 和 prethread 示意



TPC：Thread per connection



prethread：thread are created before connection

【优点】
1. 实现简单；
2. 无需 IPC，线程间通信即可；
3. 无需 fork，线程创建代价低。

【缺点】
1. 线程互斥和共享比 PPC/prefork 要复杂；
2. 某个线程故障可能导致整个进程退出；
3. OS 的上下文切换会限制并发连接数，一般几百，但比 PPC/prefork 要多。

【案例】
1. Apache 服务器 MPM worker 模式就是 prethread 模式的变种（多进程 + prethread），默认支持16 × 25 = 400 个并发处理线程。参考链接

Apache 为什么要将 prethread 改为多进程 + prethread?

2. Reactor 网络模型

Reactor 介绍

Reactor：基于多路复用的事件响应网络编程模型。

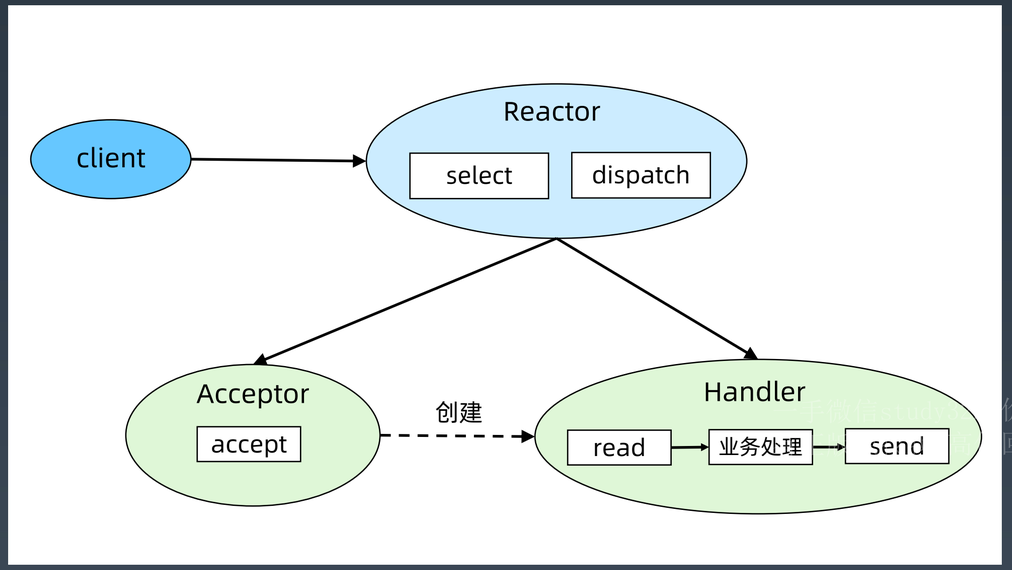
【多路复用】
多个连接复用同一个阻塞对象，例如 Java 的 Selector、epoll 的 epoll_fd（epoll_create 函数创建）。

【事件响应】
不同的事件分发给不同的对象处理，Java 的事件有 OP_ACCEPT、OP_CONNECT、OP_READ、OP_WRITE。
参考链接

【优缺点】
1. 实现比传统网络模型要复杂一些；
2. 支持海量连接。

为什么复用同一个阻塞对象就能够支持海量连接？

Reactor 模式1 - 单 Reactor 单进程/线程



【优点】
1. 实现简单，无进程通信，无线程互斥和通信；
2. 无上下文切换，某些场景下性能可以做到很高。

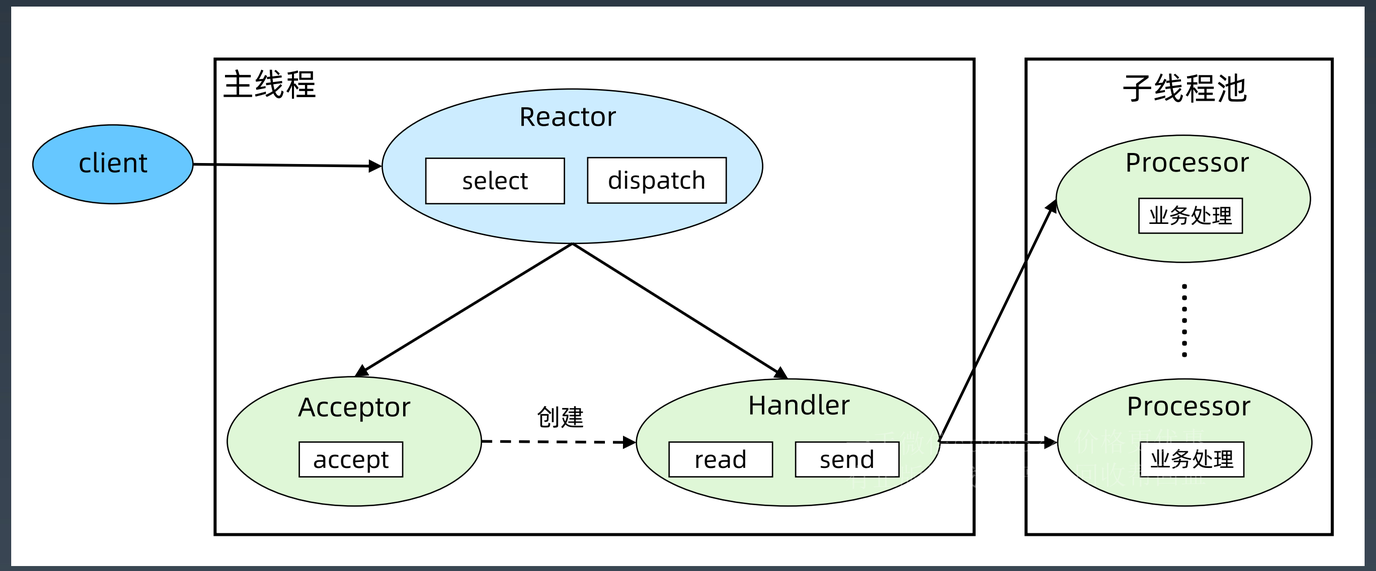
【缺点】
1. 只有一个进程，无法发挥多核 CPU 的性能，只能采取部署多个系统来利用多核 CPU，但这样会带来运维复杂度；
2. Handler 在处理某个连接上的业务时，整个进程无法处理其他连接的事件，可能导致性能瓶颈。

【案例】
Redis。

1. Reactor 对象通过 select 监控连接事件，收到事件后通过 dispatch 进行分发；
2. 如果是连接建立的事件，则由 Acceptor 处理，Acceptor 通过 accept 接受连接，并创建一个 Handler 来处理连接后续的各种事件；
3. 如果不是连接建立事件，则 Reactor 会调用连接对应的 Handler（第2步中创建的 Handler）来进行响应，Handler 会完成 read-> 业务处理-> send 的完整业务流程。

Redis 采用这种模式，你能推断出 Redis 的缺点是什么吗？

Reactor 模式2 - 单 Reactor 多线程



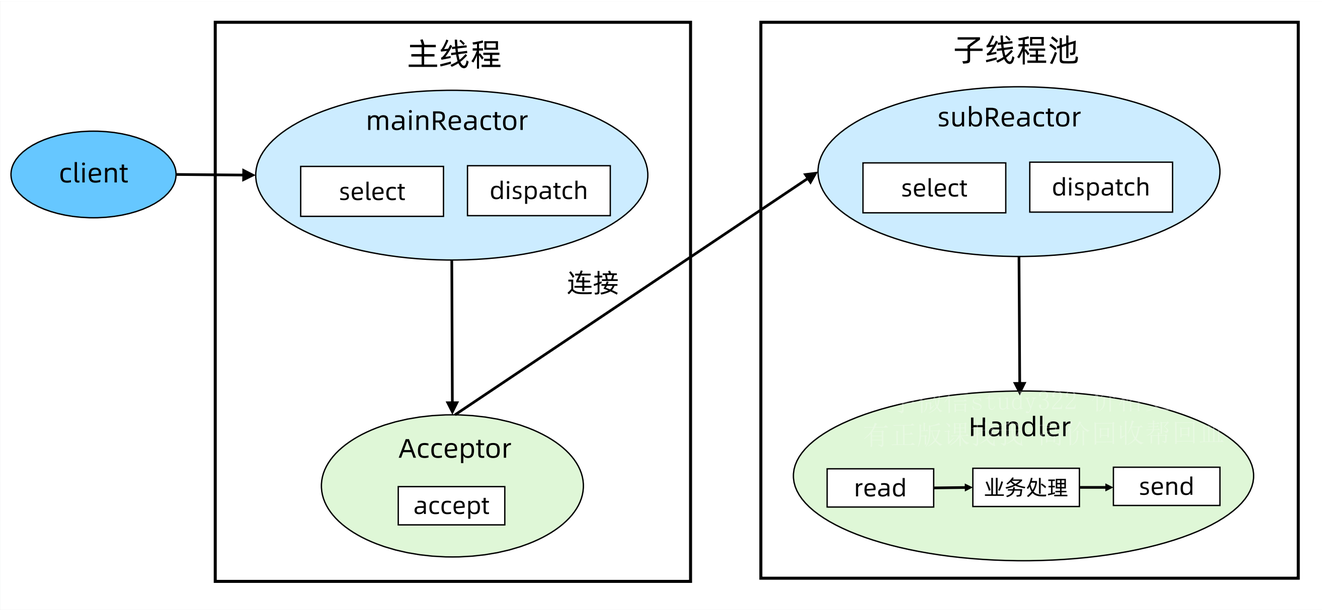
【优点】
1. 充分利用了多核 CPU 的优势，性能高。

【缺点】
1. 多线程数据共享和访问比较复杂；
2. Reactor 承担所有事件的监听和响应，只在主线程中运行，瞬时高并发时会成为性能瓶颈。

1. 主线程中，Reactor 对象通过 select 监控连接事件，收到事件后通过 dispatch 进行分发；
2. 如果是连接建立的事件，则由 Acceptor 处理，Acceptor 通过 accept 接受连接，并创建一个 Handler 来处理连接后续的各种事件；
3. 如果不是连接建立事件，则 Reactor 会调用连接对应的 Handler（第2步中创建的 Handler）来进行响应；
4. Handler 只负责响应事件，不进行业务处理；Handler 通过 read 读取到数据后，会发给 Processor 进行业务处理；
5. Processor 会在独立的子线程中完成真正的业务处理，然后将响应结果发给主进程的 Handler 处理；Handler 收到响应后通过 send 将响应结果返回给 client。

为什么这里没有“单 Reactor 多进程”的模式？

Reactor 模式3 - 多 Reactor 多进程/线程



【优点】
1. 充分利用了多核 CPU 的优势，性能高；
2. 实现简单，父子进程(线程)交互简单，subReactor 子进程(线程)间无互斥共享或通信。

【缺点】
没有明显的缺点。

【案例】
Memcached、Netty、Nginx 等，注意：实现细节都有一些差异，例如 Memcached 用了事件队列、Nginx 是子进程 accept。

1. 父进程中 mainReactor 对象通过 select 监控连接建立事件，收到事件后通过 Acceptor 接收，将新的连接分配给某个子进程；
2. 子进程的 subReactor 将 mainReactor 分配的连接加入连接队列进行监听，并创建一个 Handler 用于处理连接的各种事件；
3. 当有新的事件发生时，subReactor 会调用连接对应的 Handler（即第2步中创建的 Handler）来进行响应；
4. Handler 完成 read →业务处理→ send 的完整业务流程。

Redis 6.0 后多线程写同一个 key 怎么处理？

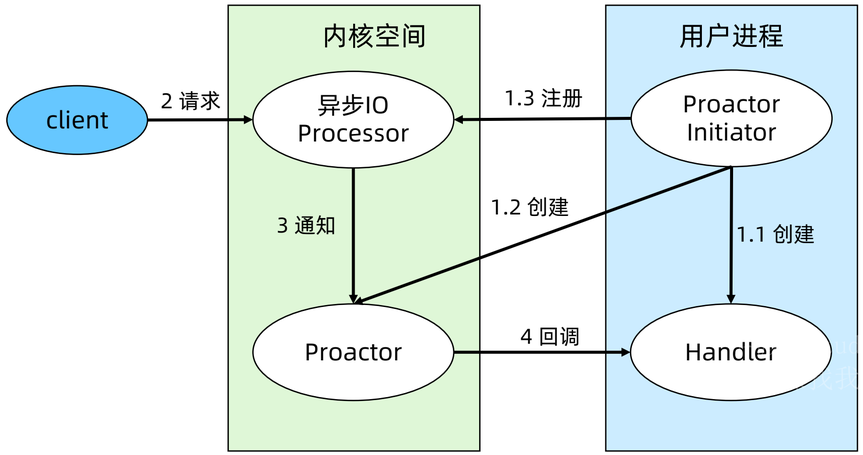
Netty 代码示例 - example/http/helloworld

极客时间 训练营

```
// Configure the server.
EventLoopGroup bossGroup = new NioEventLoopGroup( nThreads: 1);
EventLoopGroup workerGroup = new NioEventLoopGroup();
try {
    ServerBootstrap b = new ServerBootstrap();
    b.option(ChannelOption.SO_BACKLOG,  value: 1024);
    b.group(bossGroup, workerGroup)
     .channel(NioServerSocketChannel.class)
     .handler(new LoggingHandler(LogLevel.INFO))
     .childHandler(new HttpHelloWorldServerInitializer(sslCtx));

    Channel ch = b.bind(PORT).sync().channel();

    System.err.println("Open your web browser and navigate to " +
            (SSL? "https" : "http") + "://127.0.0.1:" + PORT + '/');

    ch.closeFuture().sync();
} finally {
    bossGroup.shutdownGracefully();
    workerGroup.shutdownGracefully();
}
```

3. Proactor 网络模型

Proactor 模式



1. Proactor Initiator 负责创建 Proactor 和 Handler，并将 Proactor 和 Handler 都通过 Asynchronous Operation Processor 注册到内核；
2. Asynchronous Operation Processor 负责处理注册请求，并完成 I/O 操作；
3. Asynchronous Operation Processor 完成 I/O 操作后通知 Proactor；
4. Proactor 根据不同的事件类型回调不同的 Handler 进行业务处理；
5. Handler 完成业务处理，Handler 也可以注册新的 Handler 到内核进程。

【优点】
1. 理论上性能要比 Reactor 更高一些，但实测性能差异不大。

【案例】
Windows IOCP。

【缺点】
1. 操作系统实现复杂，Linux 目前对 Proactor 模式支持并不成熟；
2. 程序调试复杂。

4. 网络模型对比

三类网络模型对比

传统网络模型 → Reactor → Proactor

1.上下文切换限制了连接数量；
2. 阻塞同步模型。

1. 支持海量连接；
2. 多路复用(同步)模型。

1. 支持海量连接；
2. 异步模型。

	复杂度	连接数量	应用场景
PPC	低	常量连接，几百	内部系统、中间件
prefork	低	常量连接，几百	内部系统、中间件
TPC	低	常量连接，几百	内部系统、中间件
prethread	低	常量连接，几百	内部系统、中间件
Reactor	中，程序复杂	海量连接，上万	互联网、物联网
Proactor	高，OS 内核复杂	海量连接，上万	互联网、物联网

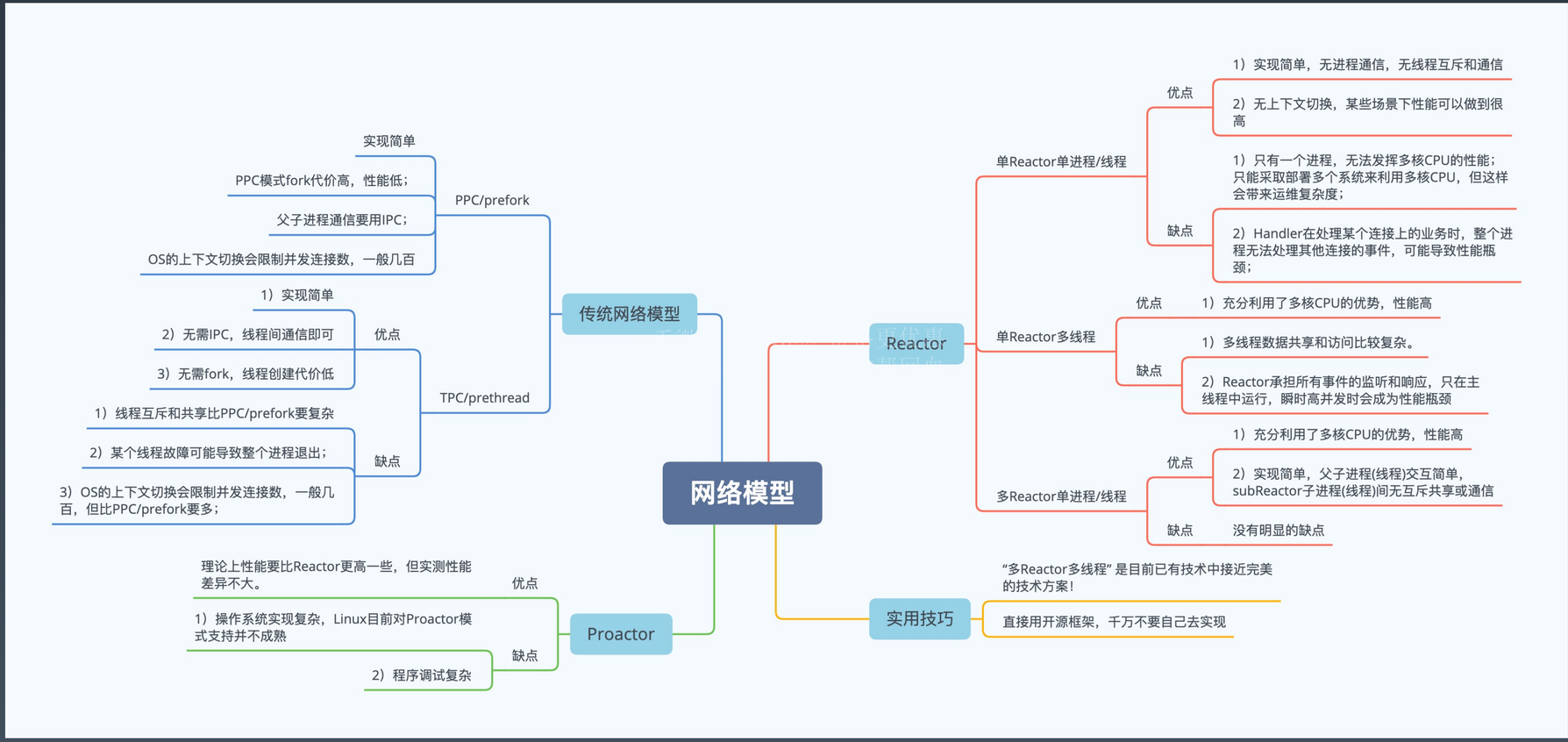
单个连接的处理性能，不同模式差异会很大么？

三类网络模型实战技巧

“多Reactor多线程” 是目前已有技术中接近完美的技术方案！
1. 所有场景；2. 所有平台；3. 性能和 Proactor 接近。

直接用开源框架，千万不要自己去实现，例如 Netty、libevent (memcached 网络框架)、libuv (node.js 底层网络框架)。

本节思维导图


网络模型
传统网络模型
PPC/prefork
实现简单
PPC模式fork代价高，性能低；
父子进程通信要用IPC；
OS的上下文切换会限制并发连接数，一般几百
TPC/prethread
优点
1）实现简单
2）无需IPC，线程间通信即可
3）无需fork，线程创建代价低
缺点
1）线程互斥和共享比PPC/prefork要复杂
2）某个线程故障可能导致整个进程退出；
3）OS的上下文切换会限制并发连接数，一般几百，但比PPC/prefork要多；
Proactor
优点
理论上性能要比Reactor更高一些，但实测性能差异不大。
缺点
1）操作系统实现复杂，Linux目前对Proactor模式支持并不成熟
2）程序调试复杂
Reactor
单Reactor单进程/线程
优点
1）实现简单，无进程通信，无线程互斥和通信
2）无上下文切换，某些场景下性能可以做到很高
缺点
1）只有一个进程，无法发挥多核CPU的性能；只能采取部署多个系统来利用多核CPU，但这样会带来运维复杂度；
2）Handler在处理某个连接上的业务时，整个进程无法处理其他连接的事件，可能导致性能瓶颈；
单Reactor多线程
优点
1）充分利用了多核CPU的优势，性能高
缺点
1）多线程数据共享和访问比较复杂。
2）Reactor承担所有事件的监听和响应，只在主线程中运行，瞬时高并发时会成为性能瓶颈
多Reactor单进程/线程
优点
1）充分利用了多核CPU的优势，性能高
2）实现简单，父子进程(线程)交互简单，subReactor子进程(线程)间无互斥共享或通信
缺点
没有明显的缺点
实用技巧
“多Reactor多线程”是目前已有技术中接近完美的技术方案！
直接用开源框架，千万不要自己去实现

随堂测验

极客时间 训练营

【判断题】

1. PPC/prefork 等传统网络模型不支持海量连接。
2. 单 Reactor 单进程模式因为没有上下文切换，性能会很高。
3. 单 Reactor 多进程模式一样可行。
4. 多 Reactor 多线程是接近完美的网络模型，而 Proactor 是性能最高的网络模型。
5. 如果技术实力足够，可以自己开发网络模型，这样会更可控一些。

【思考题】

如果开发消息队列，可以选用哪些网络模型？

Q&A

茶歇时间

八卦，趣闻，内幕……

THANKS