链滴

线程模型

作者：ximenxiaolanggou
原文链接：https://ld246.com/article/1594887244486
来源网站：链滴

<h2 id="传统阻塞I-O服务模型">传统阻塞 I/O 服务模型</h2>
<h2 id="工作原理">工作原理</h2>

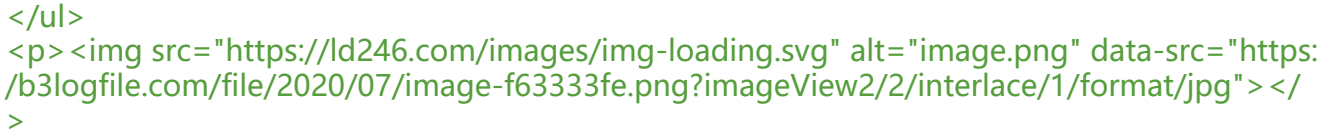
黄色的框表示对象， 蓝色的框表示线程
白色的框表示方法(API)

<h2 id="模型特点">模型特点</h2>

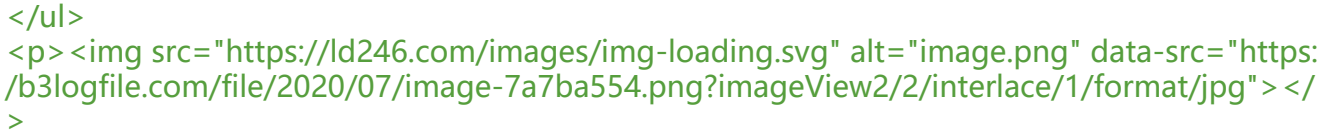
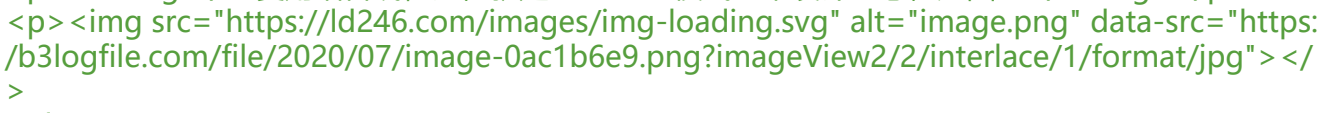
采用阻塞 IO 模式获取输入的数据
每个连接都需要独立的线程完成数据的输入，业务处理，数据返回

<h2 id="问题">问题</h2>

当并发数很大，就会创建大量的线程，占用很大系统资源
连接创建后，如果当前线程暂时没有数据可读，该线程会阻塞在 read 操作，造成线程资源浪费</i>


<h2 id="Reactor模式">Reactor 模式</h2>
<h2 id="针对传统阻塞-I-O-服务模型的-2-个缺点-解决方案">针对传统阻塞 I/O 服务模型的 2 个缺，解决方案</h2>

基于 I/O 复用模型：多个连接共用一个阻塞对象，应用程序只需要在一个阻塞对象等待，无需阻等待所有连接。当某个连接有新的数据可以处理时，操作系统通知应用程序，线程从阻塞状态返回，始进行业务处理 Reactor 对应的叫法: 1. 反应器模式 2. 分发者模式(Dispatcher)3. 通知者模式(notifier
基于线程池复用线程资源：不必再为每个连接创建线程，将连接完成后的业务处理任务分配给线进行处理，一个线程可以处理多个连接的业务。


<h2 id="Reactor模式核心">Reactor 模式核心</h2>
<h3 id="概述">概述</h3>
I/O 复用结合线程池，就是 Reactor 模式基本设计思想，如图：


Reactor 模式，通过一个或多个输入同时传递给服务处理器的模式(基于事件驱动)
服务器端程序处理传入的多个请求,并将它们同步分派到相应的处理线程，因此 Reactor 模式也叫Dispatcher 模式
Reactor 模式使用 IO 复用监听事件, 收到事件后，分发给某个线程(进程), 这点就是网络服务器并发处理关键

<h3 id="核心组成">核心组成</h3>

Reactor：Reactor 在一个单独的线程中运行，负责监听和分发事件，分发给适当的处理程序来对O 事件做出反应。它就像公司的电话接线员，它接听来自客户的电话并将线路转移到适当的联系人；
Handlers：处理程序执行 I/O 事件要完成的实际事件，类似于客户想要与之交谈的公司中的实官员。Reactor 通过调度适当的处理程序来响应 I/O 事件，处理程序执行非阻塞操作。

<h2 id="单Reactor单线程">单 Reactor 单线程</h2>

/b3logfile.com/file/2020/07/image-bcaf3f0b.png?imageView2/2/interlace/1/format/jpg"></
>
<h3 id="方案说明">方案说明</h3>

Select 是前面 I/O 复用模型介绍的标准网络编程 API，可以实现应用程序通过一个阻塞对象监听
路连接请求
Reactor 对象通过 Select 监控客户端请求事件，收到事件后通过 Dispatch 进行分发
如果是建立连接请求事件，则由 Acceptor 通过 Accept 处理连接请求，然后创建一个 Handler
象处理连接完成后的后续业务处理
如果不是建立连接事件，则 Reactor 会分发调用连接对应的 Handler 来响应
Handler 会完成 Read→ 业务处理 →Send 的完整业务流程

<h3 id="总结">总结</h3>
服务器端用一个线程通过多路复用搞定所有的 IO 操作（包括连接，读、写等），编码简单，清
明了，但是如果客户端连接数量较多，将无法支撑，前面的 NIO 案例就属于这种模型。

优点：模型简单，没有多线程、进程通信、竞争的问题，全部都在一个线程中完成
缺点：性能问题，只有一个线程，无法完全发挥多核 CPU 的性能。Handler 在处理某个连接上
业务时，整个进程无法处理其他连接事件，很容易导致性能瓶颈
缺点：可靠性问题，线程意外终止，或者进入死循环，会导致整个系统通信模块不可用，不能接
和处理外部消息，造成节点故障
使用场景：客户端的数量有限，业务处理非常快速，比如 Redis 在业务处理的时间复杂度
O(1) 的情况

<h2 id="单Reactor多线程">单 Reactor 多线程</h2>

/b3logfile.com/file/2020/07/image-e80ff5a2.png?imageView2/2/interlace/1/format/jpg"></
>
<h3 id="方案说明-">方案说明</h3>

Reactor 对象通过 select 监控客户端请求事件, 收到事件后，通过 dispatch 进行分发
如果建立连接请求, 则右 Acceptor 通过 accept 处理连接请求, 然后创建一个 Handler 对象处理
成连接后的各种事件
如果不是连接请求，则由 reactor 分发调用连接对应的 handler 来处理
handler 只负责响应事件，不做具体的业务处理, 通过 read 读取数据后，会分发给后面的 worker
线程池的某个线程处理业务
worker 线程池会分配独立线程完成真正的业务，并将结果返回给 handler
handler 收到响应后，通过 send 将结果返回给 client

<h3 id="总结-">总结</h3>

优点：可以充分的利用多核 cpu 的处理能力
缺点：多线程数据共享和访问比较复杂，reactor 处理所有的事件的监听和响应，在单线程运行
在高并发场景容易出现性能瓶颈

<h2 id="主从Reactor多线程">主从 Reactor 多线程</h2>

/b3logfile.com/file/2020/07/image-05dd5e14.png?imageView2/2/interlace/1/format/jpg"><
p>
<h3 id="方案说明--">方案说明</h3>

Reactor 主线程 MainReactor 对象通过 select 监听连接事件, 收到事件后，通过 Acceptor 处
连接事件
当 Acceptor 处理连接事件后，MainReactor 将连接分配给 SubReactor
subreactor 将连接加入到连接队列进行监听,并创建 handler 进行各种事件处理
当有新事件发生时，subreactor 就会调用对应的 handler 处理
handler 通过 read 读取数据，分发给后面的 worker 线程处理
worker 线程池分配独立的 worker 线程进行业务处理，并返回结果
handler 收到响应的结果后，再通过 send 将结果返回给 client
Reactor 主线程可以对应多个 Reactor 子线程, 即 MainRecator 可以关联多个 SubReactor

<h3 id="总结--">总结</h3>

优点：父线程与子线程的数据交互简单职责明确，父线程只需要接收新连接，子线程完成后续的
务处理
优点：父线程与子线程的数据交互简单，Reactor 主线程只需要把新连接传给子线程，子线程无
返回数据
缺点：编程复杂度较高

<h3 id="Scalable-IO-in-Java-对-Multiple-Reactors-的原理图解">Scalable IO in Java 对 Multiple
Reactors 的原理图解</h3>
<p><img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https:
/b3logfile.com/file/2020/07/image-1b264b5f.png?imageView2/2/interlace/1/format/jpg"></
>
<h2 id="Reactor模式小结">Reactor 模式小结</h2>
<h3 id="3-种模式用生活案例来理解">3 种模式用生活案例来理解</h3>

单 Reactor 单线程，前台接待员和服务员是同一个人，全程为顾客服
单 Reactor 多线程，1 个前台接待员，多个服务员，接待员只负责接待
主从 Reactor 多线程，多个前台接待员，多个服务生

<h3 id="Reactor-模式具有如下的优点">Reactor 模式具有如下的优点</h3>

响应快，不必为单个同步时间所阻塞，虽然 Reactor 本身依然是同步的
可以最大程度的避免复杂的多线程及同步问题，并且避免了多线程/进程的切换开销
扩展性好，可以方便的通过增加 Reactor 实例个数来充分利用 CPU 资源
复用性好，Reactor 模型本身与具体事件处理逻辑无关，具有很高的复用性
