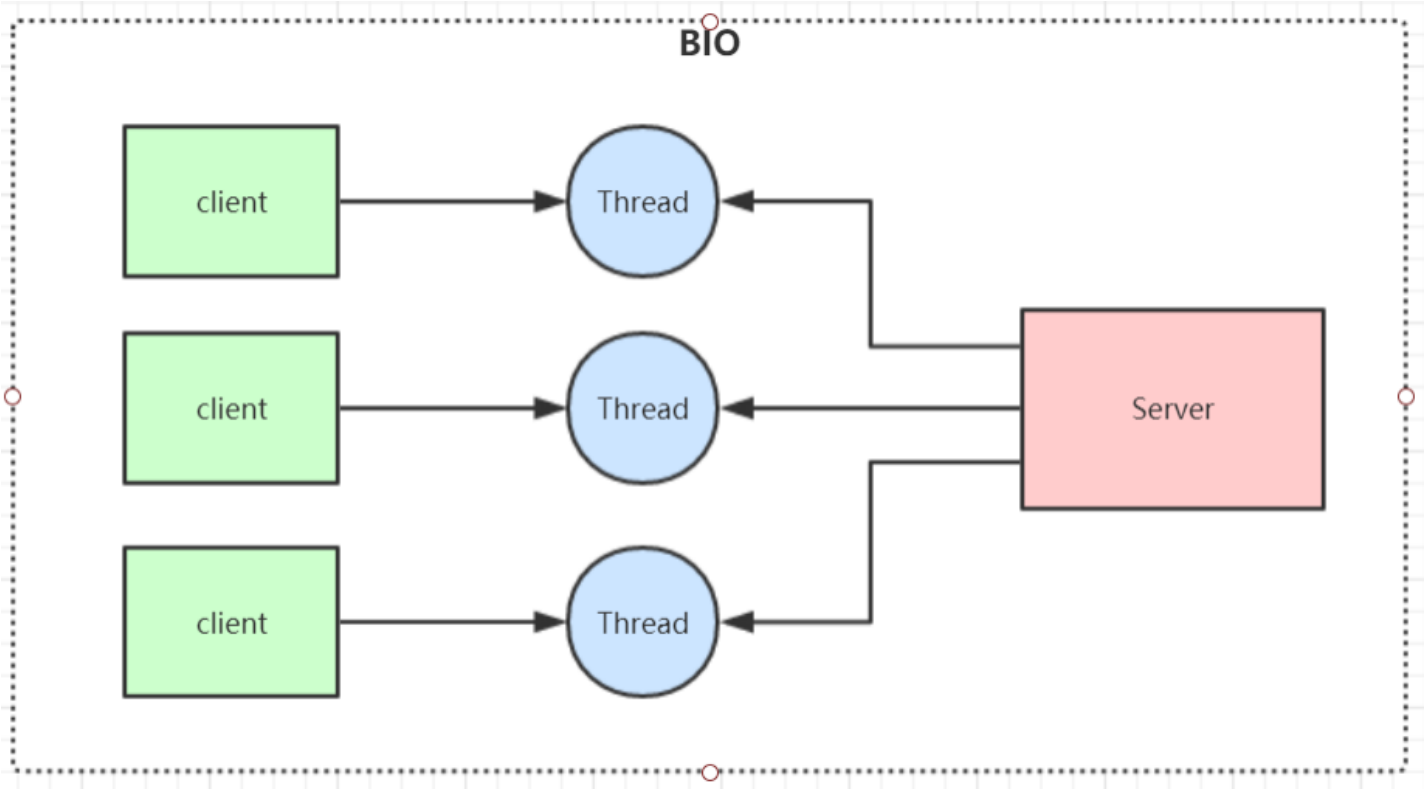
BIO(Blocking IO)

同步阻塞模型，一个客户端连接对应一个处理线程



即：一个线程处理一个客户端请求。

缺点：

1、IO代码里read操作是阻塞操作，如果连接不做数据读写操作会导致线程阻塞，浪费资源

2、如果线程很多，会导致服务器线程太多，压力太大，比如C10K问题

总结：如果说客户端只是建立一个连接而已，不进行任何读写，这个线程可以说是没有干任何事情

NIO(Non Blocking IO)

同步非阻塞，服务器实现模式为一个线程可以处理多个请求(连接)

1.0版本（早期版本）：一个线程无限循环去list（存放着客户端连接）轮训，检查是否有读写请求，如果有则处理，如果没有跳过

这个版本如果连接数特别多的话，会有大量的无效遍历，假如有1000个客户端连接，只有其中100个有读写事件，那么还是会循环1000次到真正这100次的时候才会处理其中的900次循环都是无效的

2.0版本（jdk1.4以上）：客户端发送的连接请求都会注册到多路复用器selector上，多路复用器轮询到连接有IO请求就进行处理，JDK1.4开始引入。

这个版本进行了很大程度的优化，当客户端连接以后，如果有读写事件，则会加入一个队列里，处理事件的线程会阻塞等待这个队列里有新的元素之后处理事件

步骤：

1. 创建ServerSocketChannel服务端
2. 创建多路复用器Selector（每个操作系统创建出来的是不一样的Centos创建的是EPollSelectorProviderImpl，Windows创建的是WindowsSelectorImpl，其实就是Linux的Epoll实例EPollArrayWrapper）
3. ServerSocketChannel将建立连接事件注册到Selector中（register方法往EPollArrayWrapper中添加元素）
4. 处理事件
    1. 如果是建立连接事件，则把客户端的读写请求也注册到Selector中
    2. 如果是读写事件则按业务处理

Redis线程模型

Redis就是典型的基于epoll的NIO线程模型(nginx也是)，epoll实例收集所有事件(连接与读写事件)，由一个服务端线程连续处理所有事

件命令。

Redis底层关于epoll的源码实现在redis的src源码目录的ae_epoll.c文件里，感兴趣可以自行研究。