根据 Reactor 的数量和处理资源池线程的数量不同，有 3 种典型的实现：

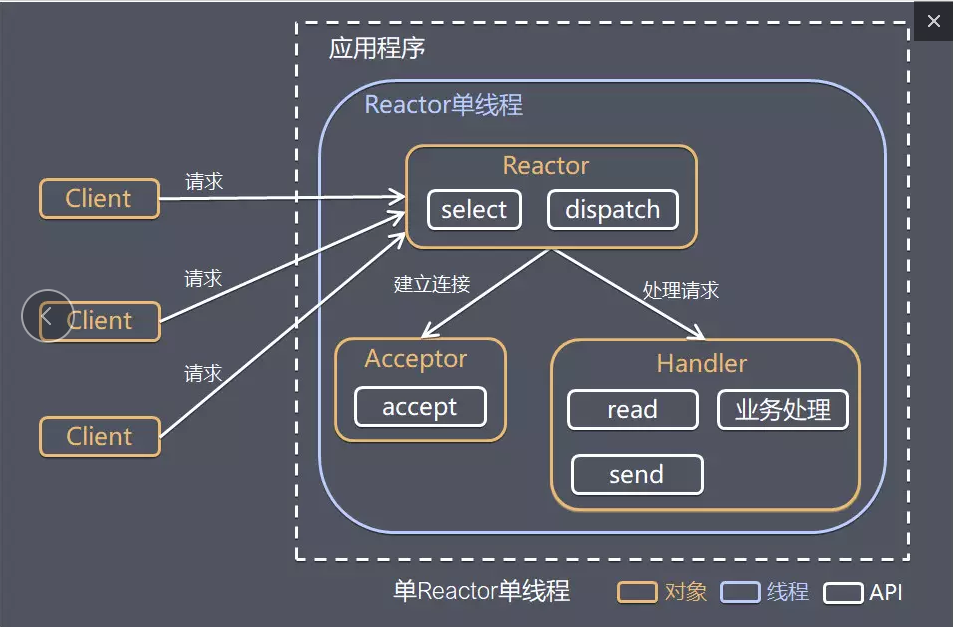
**单 Reactor 单线程**

**单 Reactor 多线程**

**主从 Reactor 多线程**

下面详细介绍这 3 种实现方式。

**单 Reactor 单线程**



其中，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多路连接请求，其他方案示意图类似。

方案说明：

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 对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。

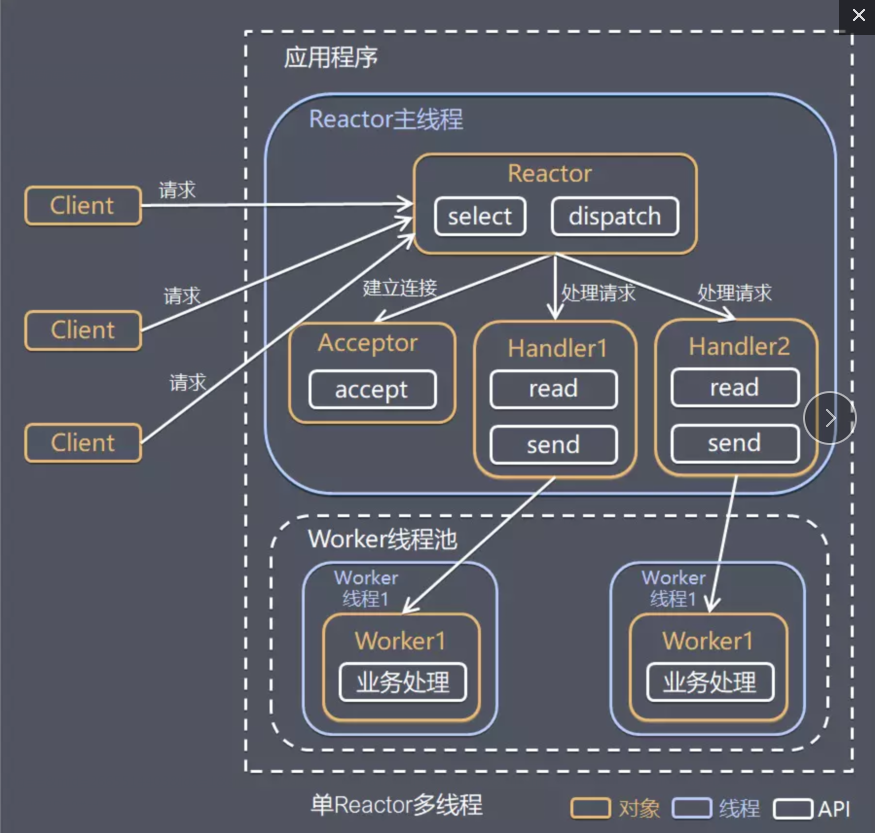
优点：模型简单，没有多线程、进程通信、竞争的问题，全部都在一个线程中完成。

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 在处理某个连接上的业务时，整个进程无法处理其他连接事件，很容易导致性能瓶颈。

可靠性问题，线程意外跑飞，或者进入死循环，会导致整个系统通信模块不可用，不能接收和处理外部消息，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，业务处理的时间复杂度 O(1)。

单 Reactor 多线程



方案说明：

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 对象处理连接完成后续的各种事件。

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 只负责响应事件，不做具体业务处理，通过 Read 读取数据后，会分发给后面的 Worker 线程池进行业务处理。

Worker 线程池会分配独立的线程完成真正的业务处理，如何将响应结果发给 Handler 进行处理。

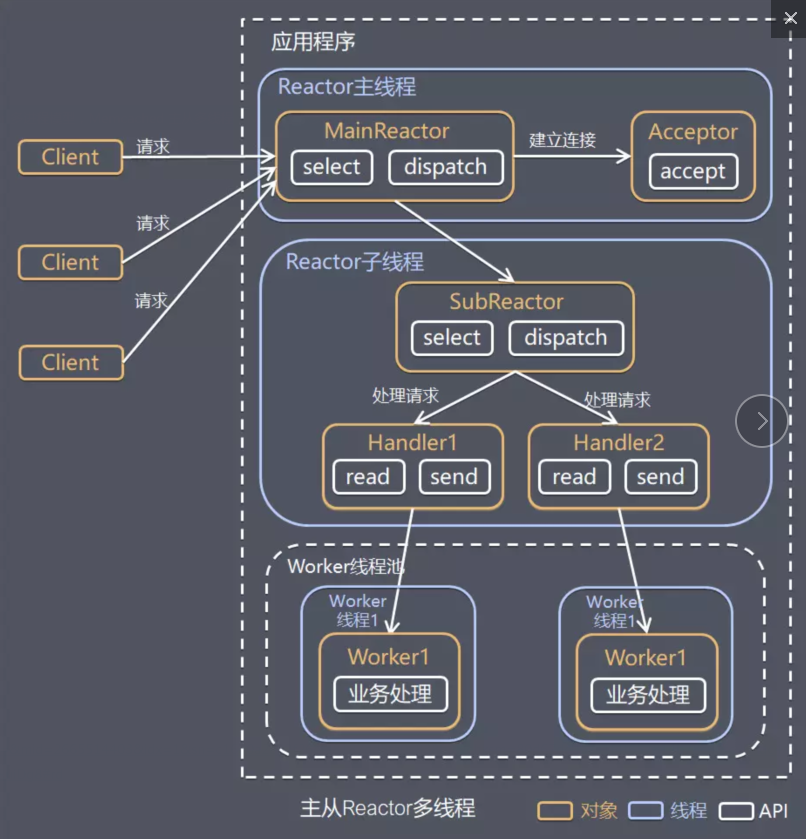
Handler 收到响应结果后通过 Send 将响应结果返回给 Client。

优点：可以充分利用多核 CPU 的处理能力。

缺点：多线程数据共享和访问比较复杂；Reactor 承担所有事件的监听和响应，在单线程中运行，高并发场景下容易成为性能瓶颈。

主从 Reactor 多线程

针对单 Reactor 多线程模型中，Reactor 在单线程中运行，高并发场景下容易成为性能瓶颈，可以让 Reactor 在多线程中运行。



方案说明：

Reactor 主线程 MainReactor 对象通过 Select 监控建立连接事件，收到事件后通过 Acceptor 接收，处理建立连接事件。

Acceptor 处理建立连接事件后，MainReactor 将连接分配 Reactor 子线程给 SubReactor 进行处理。

SubReactor 将连接加入连接队列进行监听，并创建一个 Handler 用于处理各种连接事件。

当有新的事件发生时，SubReactor 会调用连接对应的 Handler 进行响应。

Handler 通过 Read 读取数据后，会分发给后面的 Worker 线程池进行业务处理。

Worker 线程池会分配独立的线程完成真正的业务处理，如何将响应结果发给 Handler 进行处理。

Handler 收到响应结果后通过 Send 将响应结果返回给 Client。

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，子线程完成后续的业务处理。

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，子线程无需返回数据。

这种模型在许多项目中广泛使用，包括 Nginx 主从 Reactor 多进程模型，Memcached 主从多线程，Netty 主从多线程模型的支持。