

我是一段不羁的公告！

记得给芬芳这 3 个项目加油，添加一个 STAR 噢。

<https://github.com/YunaiV/SpringBoot-Labs>

<https://github.com/YunaiV/onemall>

<https://github.com/YunaiV/ruoyi-vue-pro>

• NETTY

精尽 Netty 源码解析 —— EventLoop（一）之 Reactor 模型

1. 概述

从本文开始，我们来分享 Netty 非常重要的一个组件 EventLoop。在看 EventLoop 的具体实现之前，我们先来对 Reactor 模型做个简单的了解。

为什么要了解 Reactor 模型呢？因为 EventLoop 是 Netty 基于 Reactor 模型的思想进行实现。所以理解 Reactor 模型，对于我们理解 EventLoop 会有很大帮助。


我们来看看 Reactor 模型的**核心思想**：

将关注的 I/O 事件注册到多路复用器上，一旦有 I/O 事件触发，将事件分发到事件处理器中，执行就绪 I/O 事件对应的处理函数中。模型中有三个重要的组件：

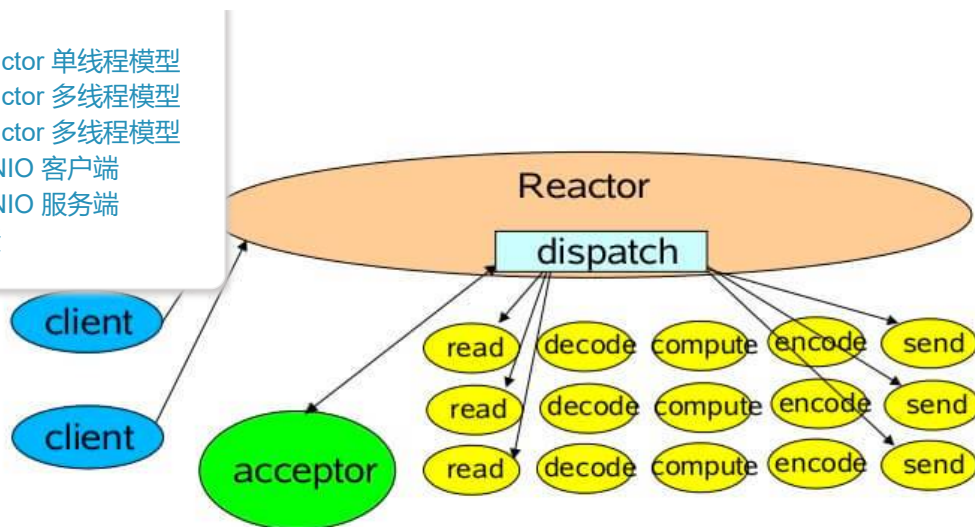
- 多路复用器：由操作系统提供接口，Linux 提供的 I/O 复用接口有 select、poll、epoll。
- 事件分离器：将多路复用器返回的就绪事件分发到事件处理器中。
- 事件处理器：处理就绪事件处理函数。

初步一看，Java NIO 符合 Reactor 模型啊？因为 Reactor 有 3 种模型实现：

1. 单 Reactor 单线程模型
2. 单 Reactor 多线程模型
3. 多 Reactor 多线程模型

 由于老芬芳不擅长相对理论文章的内容编写，所以「2.」、「3.」、「4.」小节的内容，我决定一本正经的引用基友 wier 的《【NIO 系列】——之 Reactor 模型》。

1. 概述
2. 单 Reactor 单线程模型
3. 单 Reactor 多线程模型
4. 多 Reactor 多线程模型
5. Netty NIO 客户端
6. Netty NIO 服务端
666. 彩蛋



老芳芳：示例代码主要表达大体逻辑，比较奔放。所以，胖友理解大体意思就好。

Reactor 示例代码如下：

```
/**
 * 等待事件到来，分发事件处理
 */
class Reactor implements Runnable {

    private Reactor() throws Exception {
        SelectionKey sk = serverSocket.register(selector, SelectionKey.OP_ACCEPT);
        // attach Acceptor 处理新连接
        sk.attach(new Acceptor());
    }

    @Override
    public void run() {
        try {
            while (!Thread.interrupted()) {
                selector.select();
                Set selected = selector.selectedKeys();
                Iterator it = selected.iterator();
                while (it.hasNext()) {
                    it.remove();
                    //分发事件处理
                    dispatch((SelectionKey) (it.next()));
                }
            }
        } catch (IOException ex) {
            // ...
        }
    }
}
```

文章目录

1. 概述
2. 单 Reactor 单线程模型
3. 单 Reactor 多线程模型
4. 多 Reactor 多线程模型
5. Netty NIO 客户端
6. Netty NIO 服务端
666. 彩蛋

```
    / k) {  
        acceptor  
        handler  
        Runnable) (k.attachment());  
        {  
            runnable.run();  
        }  
    }  
}
```

老芳芳：示例的 Handler 的代码实现应该是漏了。胖友脑补一个实现 Runnable 接口的 Handler 类。😈

这是最基础的单 Reactor 单线程模型。

Reactor 线程，负责多路分离套接字。

- 有新连接到来触发 OP_ACCEPT 事件之后，交由 Acceptor 进行处理。
- 有 IO 读写事件之后，交给 Handler 处理。

Acceptor 主要任务是构造 Handler。

- 在获取到 Client 相关的 SocketChannel 之后，绑定到相应的 Handler 上。
- 对应的 SocketChannel 有读写事件之后，基于 Reactor 分发，Handler 就可以处理了。

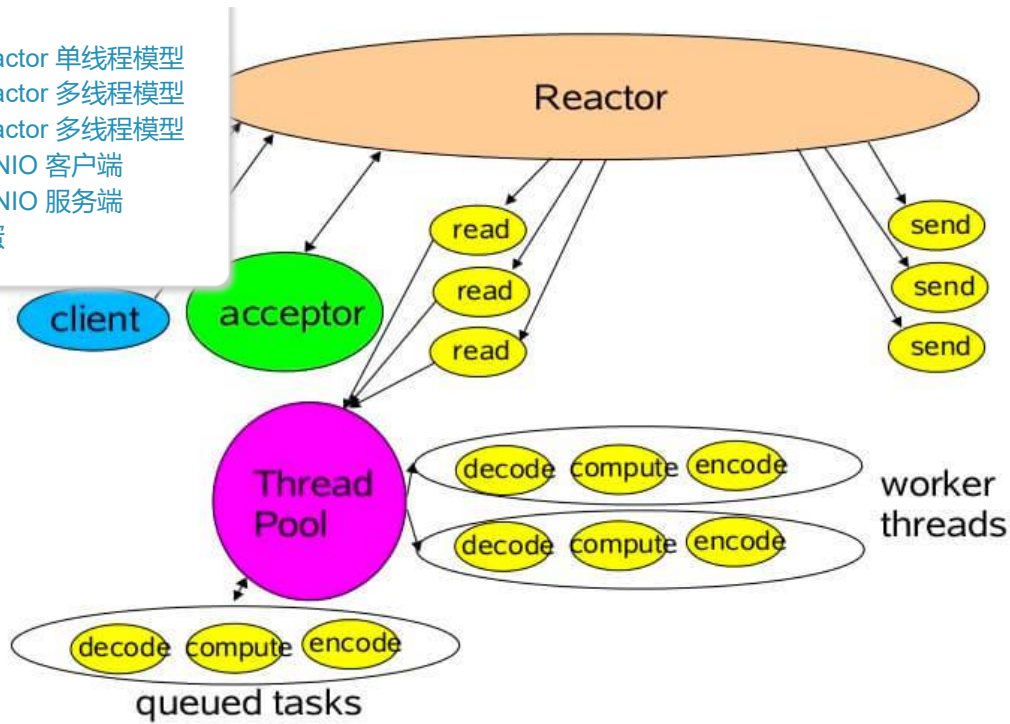
注意，所有的 IO 事件都绑定到 Selector 上，由 Reactor 统一分发。

该模型适用于处理器链中业务处理组件能快速完成的场景。不过，这种单线程模型不能充分利用多核资源，所以实际使用的不多。

3. 单 Reactor 多线程模型

示例图如下：

1. 概述
2. 单 Reactor 单线程模型
3. 单 Reactor 多线程模型
4. 多 Reactor 多线程模型
5. Netty NIO 客户端
6. Netty NIO 服务端
666. 彩蛋



相对于第一种单线程的模式来说，在处理业务逻辑，也就是获取到 IO 的读写事件之后，交由线程池来处理，这样可以减小主 Reactor 的性能开销，从而更专注的做事件分发工作了，从而提升整个应用的吞吐。

MultiThreadHandler 示例代码如下：

```
/**
 * 多线程处理读写业务逻辑
 */
class MultiThreadHandler implements Runnable {
    public static final int READING = 0, WRITING = 1;
    int state;
    final SocketChannel socket;
    final SelectionKey sk;

    //多线程处理业务逻辑
    ExecutorService executorService = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors);

    public MultiThreadHandler(SocketChannel socket, Selector sl) throws Exception {
        this.state = READING;
        this.socket = socket;
        sk = socket.register(selector, SelectionKey.OP_READ);
        sk.attach(this);
        socket.configureBlocking(false);
    }

    @Override
    public void run() {
        if (state == READING) {
            read();
        } else if (state == WRITING) {
            write();
        }
    }
}
```

- 1. 概述
- 2. 单 Reactor 单线程模型
- 3. 单 Reactor 多线程模型
- 4. 多 Reactor 多线程模型
- 5. Netty NIO 客户端
- 6. Netty NIO 服务端
- 666. 彩蛋

```
        t(() -> process());

        // 写数据
        executorService.submit(() -> process());

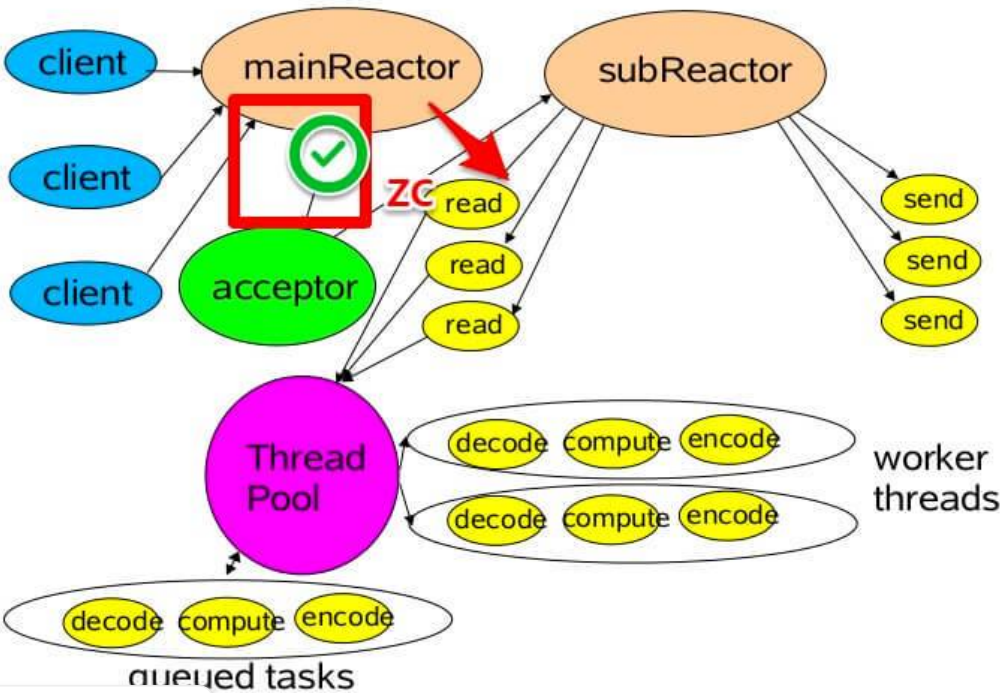
        // 下一步处理读事件
        sk.interestOps(SelectionKey.OP_READ);
        this.state = READING;
    }

    /**
     * task 业务处理
     */
    public void process() {
        //do IO ,task,queue something
    }
}
```

- 在 #read() 和 #write() 方法中，提交 executorService 线程池，进行处理。

4. 多 Reactor 多线程模型

示例图如下：



文章目录

Reactor 分成两部分：

1. 概述
2. 单 Reactor 单线程模型
3. 单 Reactor 多线程模型
4. 多 Reactor 多线程模型
5. Netty NIO 客户端
6. Netty NIO 服务端
666. 彩蛋

SocketChannel，用来处理客户端新连接的建立，并将建立的客户端的 SocketChannel 指

针，基于 mainReactor 建立的客户端的 SocketChannel 多路分离 IO 读写事件，读写网
另外扔给 worker 线程池来完成。

代码如下：

```
/**
 * 多work 连接事件Acceptor,处理连接事件
 */
class MultiWorkThreadAcceptor implements Runnable {

    // cpu线程数相同多work线程
    int workCount = Runtime.getRuntime().availableProcessors();
    SubReactor[] workThreadHandlers = new SubReactor[workCount];
    volatile int nextHandler = 0;

    public MultiWorkThreadAcceptor() {
        this.init();
    }

    public void init() {
        nextHandler = 0;
        for (int i = 0; i < workThreadHandlers.length; i++) {
            try {
                workThreadHandlers[i] = new SubReactor();
            } catch (Exception e) {
            }
        }
    }

    @Override
    public void run() {
        try {
            SocketChannel c = serverSocket.accept();
            if (c != null) { // 注册读写
                synchronized (c) {
                    // 顺序获取SubReactor，然后注册channel
                    SubReactor work = workThreadHandlers[nextHandler];
                    work.registerChannel(c);
                    nextHandler++;
                    if (nextHandler >= workThreadHandlers.length) {
                        nextHandler = 0;
                    }
                }
            }
        } catch (Exception e) {
        }
    }
}
```

SubReactor 示例代码如下：

文章目录

1. 概述
2. 单 Reactor 单线程模型
3. 单 Reactor 多线程模型
4. 多 Reactor 多线程模型
5. Netty NIO 客户端
6. Netty NIO 服务端
666. 彩蛋

```
Runnable {
```

```
int workCount = Runtime.getRuntime().availableProcessors();
ExecutorService executorService = Executors.newFixedThreadPool(workCount);

public SubReactor() throws Exception {
    // 每个SubReactor 一个selector
    this.mySelector = SelectorProvider.provider().openSelector();
}

/**
 * 注册channel
 *
 * @param sc
 * @throws Exception
 */
public void registerChannel(SocketChannel sc) throws Exception {
    sc.register(mySelector, SelectionKey.OP_READ | SelectionKey.OP_CONNECT);
}

@Override
public void run() {
    while (true) {
        try {
            //每个SubReactor 自己做事件分派处理读写事件
            selector.select();
            Set<SelectionKey> keys = selector.selectedKeys();
            Iterator<SelectionKey> iterator = keys.iterator();
            while (iterator.hasNext()) {
                SelectionKey key = iterator.next();
                iterator.remove();
                if (key.isReadable()) {
                    read();
                } else if (key.isWritable()) {
                    write();
                }
            }
        } catch (Exception e) {

        }
    }
}

private void read() {
    //任务异步处理
```

```
c(() -> process());
```

文章目录

1. 概述
2. 单 Reactor 单线程模型
3. 单 Reactor 多线程模型
4. 多 Reactor 多线程模型
5. Netty NIO 客户端
6. Netty NIO 服务端
666. 彩蛋

```
        t(() -> process());
    }

    /** task 业务处理 */
    public void process() {
        //do IO ,task,queue something
    }
}
```

从代码中，我们可以看到：

1. mainReactor 主要用来处理网络 IO 连接建立操作，通常，mainReactor 只需要一个，因为它一个线程就可以处理。
2. subReactor 主要和建立起来的客户端的 SocketChannel 做数据交互和事件业务处理操作。通常，subReactor 的个数和 CPU 个数相等，每个 subReactor 独占一个线程来处理。

此种模式中，每个模块的工作更加专一，耦合度更低，性能和稳定性也大大的提升，支持的可并发客户端数量可达到上百万级别。

老芳芳：一般来说，是达到数十万级别。

关于此种模式的应用，目前有很多优秀的框架已经在应用，比如 Mina 和 Netty 等等。**上述中去掉线程池的第三种形式的变种，也是 Netty NIO 的默认模式。**

5. Netty NIO 客户端

我们来看看 Netty NIO 客户端的示例代码中，和 EventLoop 相关的代码：

```
// 创建一个 EventLoopGroup 对象
EventLoopGroup group = new NioEventLoopGroup();
// 创建 Bootstrap 对象
Bootstrap b = new Bootstrap();
// 设置使用的 EventLoopGroup
b.group(group);
```

- 对于 Netty NIO 客户端来说，仅创建一个 EventLoopGroup。
- 一个 EventLoop 可以对应一个 Reactor。因为 EventLoopGroup 是 EventLoop 的分组，所以对等理解，EventLoopGroup 是一种 Reactor 的分组。
- 一个 Bootstrap 的启动，只能发起对一个远程的地址。所以只会使用一个 NIO Selector，也就是说仅使用一个 Reactor。即使，我们在声明使用一个 EventLoopGroup，该 EventLoopGroup 也只会分配一个 EventLoop 对 IO 事件进行处理。

因为 Netty NIO 模型主要使用于服务端开发中，如果套用在 Netty NIO 客户端中，到底使用了哪一种模式呢？

文章目录

Netty NIO 客户端，那么可以认为是【单 Reactor 单线程模型】。

1. 概述
2. 单 Reactor 单线程模型
3. 单 Reactor 多线程模型
4. 多 Reactor 多线程模型
5. Netty NIO 客户端
6. Netty NIO 服务端
666. 彩蛋

ty NIO 客户端，那么可以认为是【单 Reactor 多线程模型】。

使用【多 Reactor 多线程模型】呢？😡 创建多个 Netty NIO 客户端，连接同一个服务，以认为符合多 Reactor 多线程模型了。

三。

例如 Dubbo 或 Motan 这两个 RPC 框架，支持通过配置，同一个 Consumer 对同一个客户端连接。

6. Netty NIO 服务端

我们来看看 Netty NIO 服务端的示例代码中，和 EventLoop 相关的代码：

```
// 创建两个 EventLoopGroup 对象
EventLoopGroup bossGroup = new NioEventLoopGroup(1); // 创建 boss 线程组 用于服务端接受客户端的连接
EventLoopGroup workerGroup = new NioEventLoopGroup(); // 创建 worker 线程组 用于进行 SocketChannel 的数据读写
// 创建 ServerBootstrap 对象
ServerBootstrap b = new ServerBootstrap();
// 设置使用的 EventLoopGroup
b.group(bossGroup, workerGroup);
```

- 对于 Netty NIO 服务端来说，创建两个 EventLoopGroup 。
 - bossGroup 对应 Reactor 模式的 mainReactor，用于服务端接受客户端的连接。比较特殊的是，传入了方法参数 `nThreads = 1`，表示只使用一个 EventLoop，即只使用一个 Reactor。这个也符合我们上面提到的，“通常，mainReactor 只需要一个，因为它一个线程就可以处理”。
 - workerGroup 对应 Reactor 模式的 subReactor，用于进行 SocketChannel 的数据读写。对于 EventLoopGroup，如果未传递方法参数 `nThreads`，表示使用 CPU 个数 Reactor。这个也符合我们上面提到的，“通常，subReactor 的个数和 CPU 个数相等，每个 subReactor 独占一个线程来处理”。
- 因为使用两个 EventLoopGroup，所以符合【多 Reactor 多线程模型】的多 Reactor 的要求。实际在使用时，workerGroup 在读完数据时，具体的业务逻辑处理，我们会提交到**专门的业务逻辑线程池**，例如在 Dubbo 或 Motan 这两个 RPC 框架中。这样一来，就完全符合【多 Reactor 多线程模型】。
- 那么可能有胖友可能和我有一样的疑问，bossGroup 如果配置多个线程，是否可以使用**多个 mainReactor** 呢？我们来分析一波，一个 Netty NIO 服务端**同一时间**，只能 bind 一个端口，那么只能使用一个 Selector 处理客户端连接事件。又因为，Selector 操作是非线程安全的，所以无法在多个 EventLoop（多个线程）中，同时操作。所以这样就导致，即使 bossGroup 配置多个线程，实际能够使用的也就是一个线程。
- 那么如果一定一定一定要多个 mainReactor 呢？创建多个 Netty NIO 服务端，并绑定多个端口。

666. 彩蛋

如果 Reactor 模式讲解的不够清晰，或者想要更加深入的理解，推荐阅读如下文章：

- wier 《【NIO 系列】—— 之 Reactor 模型》
- 永顺 《Netty 源码分析之 三 我就是大名鼎鼎的 EventLoop(一)》 里面有几个图不错。
- Essviv 《Reactor 模型》 里面的代码示例不错。
- xieshuang 《异步网络模型》 内容很高端，一看就是高玩。

另外，还有一个经典的 Proactor 模型，因为 Netty 并未实现，所以笔者就省略了。如果感兴趣的胖友，可以自行 Google 了解下。