基于NIO的Reactor模型

Java NIO是同步非阻塞IO，在Java中没有IO多路复用的模型，Reactor是基于NIO实现的多路复用的一种模式。

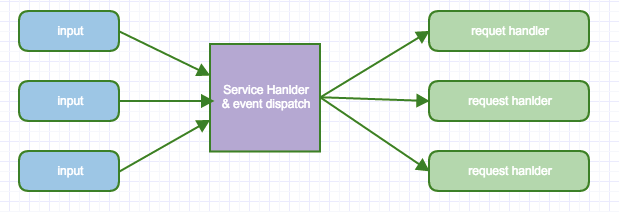
# Reactor概念

在Wiki中，Reactor模式的概念如下：

*The reactor design pattern is an event handling pattern for handling service requests delivered concurrently to a service handler by one or more inputs. The service handler then demultiplexes the incoming requests and dispatches them synchronously to the associated request handlers.*

其中关键点如下：

1. 事件驱动，Event Handling
2. 处理一个或者多个输入源，one or more inputs
3. 通过Service Handler使用多路复用技术将事件分发到相应的Request Handler进行处理



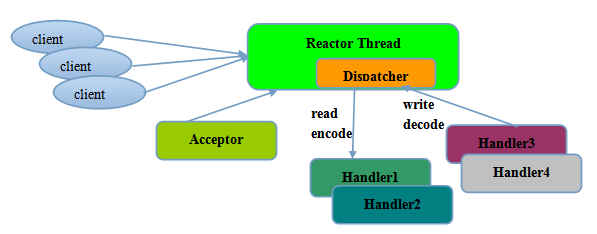
多个事件源到达后Service Handler后将事件多路分解，然后集中进行Dispatch到相应的事件服务应用中。在Reactor中，定义了以下三种角色：

* Reactor，将I/O事件分派给对应的Handler
* Acceptor，处理客户端新连接，并分派请求到处理器链中
* Handler，处理非阻塞读写任务

下面是介绍几种常见的Reactor模型。

# 单Reactor单线程模型

最基本的Reactor模型，如下图所示：



1. Reactor线程负责多路分离套接字，当新连接到来后触发connect事件之后，交由Acceptor进行处理
2. Acceptor主要任务是构建Handler，在获取到Client相关的SocketChannel后，绑定到对应的Handler上
3. SocketChannel有读写事件后，由Handler处理。所有IO事件都绑定到Selector上，由Reactor分发
4. 在Handler中，具有共同的结构：

* Read Reqeust
* Decode Request
* Process Service
* Encode Reply
* Send Reply

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，下面是单线程Reactor及Accepor的示例程序

*public class Reactor implements Runnable {*

*ServerSocketChannel serverChannel;*

*Selector selector;*

*private Reactor() throws Exception {*

*serverChannel = ServerSocketChannel.open();*

*serverChannel.bind(new InetSocketAddress(9898));*

*serverChannel.configureBlocking(false);*

*selector = Selector.open(); //定义Selector，负责Reactor的事件分发*

*System.out.println("Selector Open Success");*

*SelectionKey sk = serverChannel.register(selector, SelectionKey.OP\_ACCEPT);*

*sk.attach(new Acceptor()); //Attach Acceptor，*

*}*

*@Override*

*public void run() {*

*try {*

*while (!Thread.interrupted()) {*

*selector.select();*

*Set<SelectionKey> selected = selector.selectedKeys();*

*Iterator it = selected.iterator();*

*while (it.hasNext()) {*

*dispatch((SelectionKey) it.next()); //分发客户端连接*

*it.remove();*

*}*

*}*

*} catch (IOException ex) {*

*}}*

*void dispatch(SelectionKey selectKey ) { //事件分发逻辑*

*Runnable runnable = (Runnable) selectKey.attachment();*

*if(runnable!= null) {*

*runnable.run();*

*}}*

*class Acceptor implements Runnable {*

*@Override*

*public void run(){*

*try {*

*SocketChannel sc = serverChannel.accept();*

*if(sc != null) {*

*new Handler(sc, selector).run(); //Handler执行逻辑*

*}*

*} catch (Exception e) {*

*e.printStackTrace();*

*}*

*}*

*}*

*class Handler {*

*SocketChannel sc = null;*

*Selector selector = null;*

*Handler(SocketChannel sc, Selector selector) {*

*this.sc = sc;*

*this.selector = selector;*

*}*

*public void run() throws Exception {*

*//这里简化ByteBuffer的decode,encode及reply逻辑*

*ByteBuffer buf = ByteBuffer.allocate(1024);*

*int len = 0;*

*System.out.println("Read Msg from Channel!");*

*while((len = sc.read(buf)) > 0) {*

*buf.flip();*

*byte[] bytes = new byte[1024];*

*buf.get(bytes,0, len);*

*System.out.println(new String(bytes,0, len));*

*}*

*}*

*}*

*public static void main(String[] args) throws Exception {*

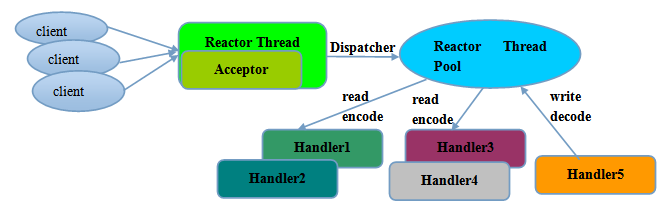
*new Reactor().run();*

*}*

*}*

# 单Reactor多线程模型

但是单线程模型不能充分利用多核资源，因此在处理业务逻辑中，获取IO读写事件之后，交由线程池来处理，这样可以减少主Reactor的性能，从而更专注做事件转发，从而提升整个应用的吞吐，如下图所示：



将上例中的Handler修改如下：

*class MultiThreadHandler implements Runnable {*

*SocketChannel sc = null;*

*Selector selector = null;*

*ExecutorService executorService = null;*

*MultiThreadHandler(SocketChannel sc, Selector selector,ExecutorService executor) {*

*this.sc = sc;*

*this.selector = selector;*

*this.executorService = executor;*

*}*

*@Override*

*public void run() {*

*executorService.submit(() -> process());*

*}*

*public void process(){*

*ByteBuffer buf = ByteBuffer.allocate(1024);*

*int len = 0;*

*System.out.println("Read Msg from Channel!");*

*try {*

*while ((len = sc.read(buf)) > 0) {*

*buf.flip();*

*byte[] bytes = new byte[1024];*

*buf.get(bytes, 0, len);*

*System.out.println(new String(bytes, 0, len));*

*}*

*}catch (Exception ex) {*

*ex.printStackTrace();*

*}*

*}*

*}*

对Reactor进行改造，将Handler替换成MultiHandler，如下:

*class Acceptor implements Runnable {*

*@Override*

*public void run(){*

*try {*

*SocketChannel sc = serverChannel.accept();*

*if(sc != null) {*

*new MultiThreadHandler(sc,selector,executorService).run();*

*}*

*} catch (Exception e) {*

*e.printStackTrace();*

*}*

*}*

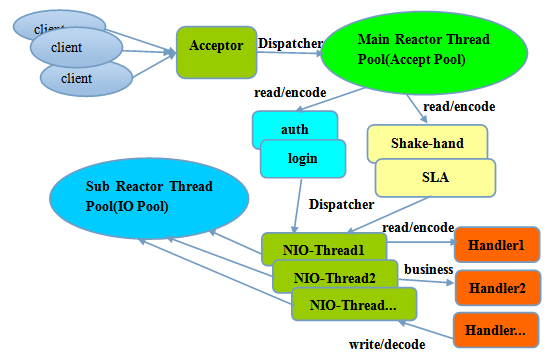
*}*

# 多Reactor多线程模型

与第二种模型相比，将Reactor分成两个部分：

* Main Reactor负责监听Server Socket，用于处理新连接的建立，将建立的SocketChannel指定注册给Sub Reactor
* Sub Reactor维护自己的Selector，基于Main Reactor注册的SocketChannel多路分离IO读写事件，读写网络数据并进行业务处理。业务的处理通过Worker线程池来完成

其流程图如下所示:



1. 定义SubReactor

*class SubReactor implements Runnable {*

*final Selector mySelector;*

*int workCount = Runtime.getRuntime().availableProcessors();*

*ExecutorService executorService = Executors.newFixedThreadPool(workCount);*

*public SubReactor() throws Exception {*

*this.mySelector = SelectorProvider.provider().openSelector();*

*}*

*//将客户端SocketChannel注册到本SubReactor的Selector*

*public void registerChannel(SocketChannel sc) throws Exception {*

*sc.register(mySelector, SelectionKey.OP\_READ | SelectionKey.OP\_CONNECT);*

*}*

*// 该线程执行，遍历MySelector，处理SocketChannel的读写事件*

*@Override*

*public void run() {*

*while(true) {*

*try {*

*selector.select();*

*Set<SelectionKey> keys = selector.selectedKeys();*

*Iterator<SelectionKey> iterator = keys.iterator();*

*while(iterator.hasNext()) {*

*SelectionKey key = iterator.next();*

*iterator.remove();*

*executorService.submit(() ->process());*

*}*

*} catch (Exception e) {*

*}*

*}*

*}*

*public void process( ){ //这里可请求转发给Handler进行处理*

*new MultiHandler().run(); //其可使用线程池来处理*

*}*

*}*

1. 定义Acceptor

*class MultiworkThreadAcceptor implements Runnable {*

*ServerSocketChannel serverSocket = null;*

*int workCount = Runtime.getRuntime().availableProcessors();*

*SubReactor[] workThreadHandlers = new SubReactor[workCount];*

*volatile int nextHandler = 0;*

*public MultiworkThreadAcceptor(ServerSocketChannel serverSocket) {*

*this.serverSocket = serverSocket;*

*this.init();*

*}*

*public void init( ) {*

*nextHandler = 0;*

*for(int i = 0;i < workThreadHandlers.length;i++) {*

*try {*

*workThreadHandlers[i] = new SubReactor();*

*} catch (Exception ex) {*

*}*

*}*

*}*

*@Override*

*public void run() {*

*try {*

*SocketChannel c = serverSocket.accept();*

*if (c != null) {// 注册读写*

*synchronized (c) { //当有客户端连接时，将其注册到某个SubReactor中*

*SubReactor work = workThreadHandlers[nextHandler];*

*work.registerChannel(c);*

*nextHandler++;*

*if (nextHandler >= workThreadHandlers.length) {*

*nextHandler = 0;*

*}*

*}*

*}*

*} catch (Exception e) {*

*}*

*}*

*}*

1. 修改主程序，通过MultiworkThreadAcceptor接收SocketChannel

*sk.attach(new MultiworkThreadAcceptor(serverChannel));*

MultiMulitReactor执行如下：

*@Override*

*public void run() {*

*try {*

*while (!Thread.interrupted()) {*

*selector.select();*

*Set<SelectionKey> selected = selector.selectedKeys();*

*Iterator it = selected.iterator();*

*while (it.hasNext()) {*

*dispatch((SelectionKey) it.next());*

*it.remove();*

*}*

*}*

*} catch (IOException ex) {}*

*void dispatch(SelectionKey selectKey ) {*

*Runnable runnable = (Runnable) selectKey.attachment();*

*//这里调用MultiworkThreadAcceptor的run方法，*

*//将客户端SocketChannel添加给某个SubReactor*

*if(runnable!= null) {*

*runnable.run();*

*}*

在这种模型中，MainReactor主要用来处理网络IO连接建立操作，通常一个线程就可以处理。而SubReactor主要和建立起来的Socket做数据交互和事件业务处理操作。通过这种方式使得每个模块的工作更加专一，耦合度更低。

Netty和Mina框架都使用这种模型。

http://www.fanyeong.com/2016/10/29/netty%E7%AE%80%E6%98%8E%E6%95%99%E7%A8%8B/#comments

https://my.oschina.net/u/1859679/blog/1844109