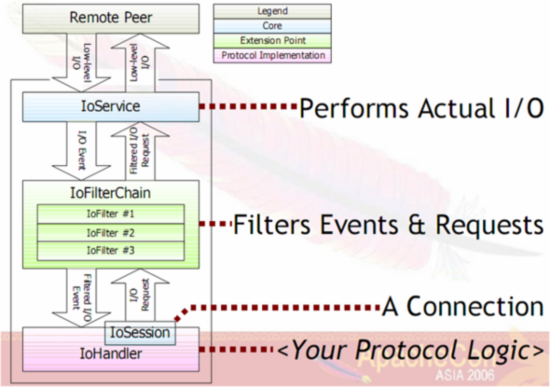
Mina是Apache社区维护的一个开源的高性能IO框架，在业界内久经考验，广为使用。Mina与后来兴起的高性能IO新贵Netty一样，都是韩国人Trustin Lee的大作，二者的设计理念是极为相似的。在作为一个强大的开发工具的同时，这两个框架的优雅设计和不俗的表现，有很多地方是值得学习和借鉴的。本文将从Mina工作原理的角度出发，对其结构进行分析。

**总体结构**

Mina的底层依赖的主要是Java NIO库，上层提供的是基于事件的异步接口。其整体的结构如下：



**IoService**

最底层的是IOService，负责具体的IO相关工作。这一层的典型代表有IOSocketAcceptor和IOSocketChannel，分别对应TCP协议下的服务端和客户端的IOService。IOService的意义在于隐藏底层IO的细节，对上提供统一的基于事件的异步IO接口。每当有数据到达时，IOService会先调用底层IO接口读取数据，封装成IoBuffer，之后以事件的形式通知上层代码，从而将Java NIO的同步IO接口转化成了异步IO。所以从图上看，进来的low-level IO经过IOService层后变成IO Event。

具体的代码可以参考org.apache.mina.core.polling.AbstractPollingIoProcessor的私有内部类Processor。

**IoFilterChain**

Mina的设计理念之一就是业务代码和数据包处理代码分离，业务代码只专注于业务逻辑，其他的逻辑如：数据包的解析，封装，过滤等则交由IoFilterChain来处理。IoFilterChain可以看成是Mina处理流程的扩展点。这样的划分使得结构更加清晰，代码分工更明确。开发者通过往Chain中添加IoFilter，来增强处理流程，而不会影响后面的业务逻辑代码。

**IoHandler**

IoHandler是实现业务逻辑的地方，需要有开发者自己来实现这个接口。IoHandler可以看成是Mina处理流程的终点，每个IoService都需要指定一个IoHandler。

**IoSession**

IoSession是对底层连接的封装，一个IoSession对应于一个底层的IO连接（在Mina中UDP也被抽象成了连接）。通过IoSession，可以获取当前连接相关的上下文信息，以及向远程peer发送数据。发送数据其实也是个异步的过程。发送的操作首先会逆向穿过IoFilterChain，到达IoService。但IoService上并不会直接调用底层IO接口来将数据发送出去，而是会将该次调用封装成一个WriteRequest，放入session的writeRequestQueue中，最后由IoProcessor线程统一调度flush出去。所以发送操作并不会引起上层调用线程的阻塞。

具体代码可以参考org.apache.mina.core.filterchain.DefaultIoFilterChain的内部类HeadFilter的filterWrite方法。

最后附上一个简单的echo server例子来作为本节结束吧。

EchoServer.java

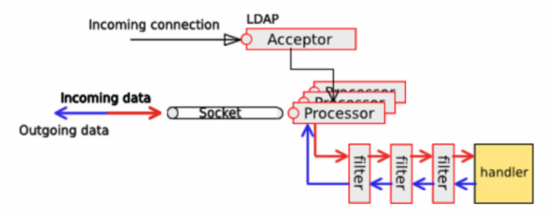
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **public** **class EchoServer {**  **public** **static** **void** main(String[] args) {  **int** PORT = 3333;  NioSocketAcceptor acceptor = **new** NioSocketAcceptor();  acceptor.setHandler(**new** EchoHandler());  **try** {  acceptor.bind(**new** InetSocketAddress(PORT));  System.out.println("Listening on " + PORT);  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

EchoHandler.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **public** **class EchoHandler extends IoHandlerAdapter {**  @Override  **public** **void** messageReceived(IoSession session, Object message) **throws** Exception {  session.write(((IoBuffer)message).duplicate());  }  } |

**工作原理**

前面介绍了Mina总体的层次结构，那么在Mina里面是怎么使用Java NIO和进行线程调度的呢？这是提高IO处理性能的关键所在。Mina的线程调度原理主要如下图所示：



**Acceptor与Connector线程**

在服务器端，bind一个端口后，会创建一个Acceptor线程来负责监听工作。这个线程的工作只有一个，调用Java NIO接口在该端口上select connect事件，获取新建的连接后，封装成IoSession，交由后面的Processor线程处理。在客户端，也有一个类似的，叫Connector的线程与之相对应。这两类线程的数量只有1个，外界无法控制这两类线程的数量。

TCP实现的代码可以参考org.apache.mina.core.polling.AbstractPollingIoAcceptor的内部类Acceptor和org.apache.mina.core.polling.AbstractPollingIoConnector的内部类Connector。

**Processor线程**

Processor线程主要负责具体的IO读写操作和执行后面的IoFilterChain和IoHandler逻辑。Processor线程的数量N默认是CPU数量+1，可以通过配置参数来控制其数量。前面进来的IoSession会被分配到这N个Processor线程中。默认的SimpleIoProcessorPool的策略是session id绝对值对N取模来分配。

每个Porcessor线程中都维护着一个selector，对它维护的IoSession集合进行select，然后对select的结果进行遍历，逐一处理。像前面提到的，读取数据，以事件的形式通知后面IoFilterChain；以及对写请求队列的flush操作，都是在这类线程中来做的。

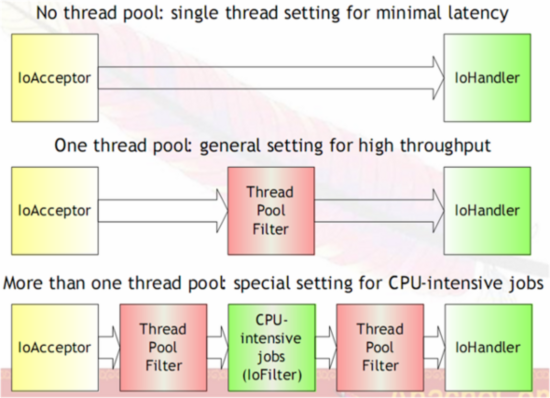
通过将session均分到多个Processor线程里进行处理，可以充分利用多核的处理能力，减轻select操作的压力。默认的Processor的线程数量设置可以满足大部分情况下的需求，但进一步的优化则需要根据实际环境进行测试。

**线程模型**

**线程模型原理**

从单一的Processor线程内部来看，IO请求的处理流程是单线程顺序处理的。前面也提到过，当Process线程select了一批就绪的IO请求后，会在线程内部逐一对这些IO请求进行处理。处理的流程包括IoFilter和IoHandler里的逻辑。当前面的IO请求处理完毕后，才会取下一个IO请求进行处理。也就是说，如果IoFilter或IoHandler中有比较耗时的操作的话（如：读取数据库等），Processor线程将会被阻塞住，后续的请求将得不到处理。这样的情况在高并发的服务器下显然是不能容忍的。于是，Mina通过在处理流程中引入线程池来解决这个问题。

那么线程池应该加在什么地方呢？正如前面所提到过的：IoFilterChain是Mina的扩展点。没错，Mina里是通过IoFilter的形式来为处理流程添加线程池的。Mina的线程模型主要有一下这几种形式：



第一种模型是单线程模型，也是Mina默认线程模型。也就是Processor包办了从底层IO到上层的IoHandler逻辑的所有执行工作。这种模型比较适合于处理逻辑简单，能快速返回的情况。

第二种模型则是在IoFilterChain中加入了Thread Pool Filter。此时的处理流程变为Processor线程读取完数据后，执行IoFilterChain的逻辑。当执行到Thread Pool Filter的时候，该Filter会将后续的处理流程封装到一个Runnable对象中，并交由Filter自身的线程池来执行，而Processor线程则能立即返回来处理下一个IO请求。这样如果后面的IoFilter或IoHandler中有阻塞操作，只会引起Filter线程池里的线程阻塞，而不会阻塞住Processor线程，从而提高了服务器的处理能力。Mina提供了Thread Pool Filter的一个实现：ExecutorFilter。

当然，也没有限制说chain中只能添加一个ExecutorFilter，开发者也可以在chain中加入多个ExecutorFilter来构成第三种情况，但一般情况下可能没有这个必要。

**请求的处理顺序**

在处理流程中加入线程池，可以较好的提高服务器的吞吐量，但也带来了新的问题：请求的处理顺序问题。在单线程的模型下，可以保证IO请求是挨个顺序地处理的。加入线程池之后，同一个IoSession的多个IO请求可能被ExecutorFilter并行的处理，这对于一些对请求处理顺序有要求的程序来说是不希望看到的。比如：数据库服务器处理同一个会话里的prepare，execute，commit请求希望是能按顺序逐一执行的。

Mina里默认的实现是有保证同一个IoSession中IO请求的顺序的。具体的实现是，ExecutorFilter默认采用了Mina提供的OrderedThreadPoolExecutor作为内置线程池。后者并不会立即执行加入进来的Runnable对象，而是会先从Runnable对象里获取关联的IoSession(这里有个down cast成IoEvent的操作)，并将Runnable对象加入到session的任务列表中。OrderedThreadPoolExecutor会按session里任务列表的顺序来处理请求，从而保证了请求的执行顺序。

对于没有顺序要请求的情况，可以为ExecutorFilter指定一个Executor来替换掉默认的OrderedThreadPoolExecutor，让同一个session的多个请求能被并行地处理，来进一步提高吞吐量。