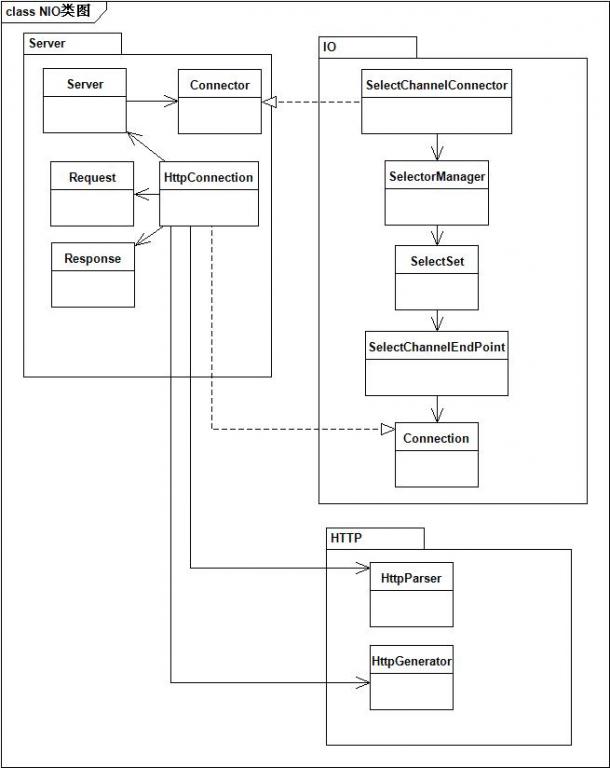
**原创  从Jetty、Tomcat和Mina中提炼NIO构架网络服务器的经典模式（一）**[收藏](javascript:d=document;t=d.selection?(d.selection.type!='None'?d.selection.createRange().text:''):(d.getSelection?d.getSelection():'');void(saveit=window.open('http://wz.csdn.net/storeit.aspx?t='+escape(d.title)+'&u='+escape(d.location.href)+'&c='+escape(t),'saveit','scrollbars=no,width=590,height=300,left=75,top=20,status=no,resizable=yes'));saveit.focus();)****

如何正确使用NIO来构架网络服务器一直是最近思考的一个问题，于是乎分析了一下Jetty、Tomcat和Mina有关NIO的源码，发现大伙都基于类似的方式，我感觉这应该算是NIO构架网络服务器的经典模式，并基于这种模式写了个小小网络服务器，压力测试了一下，效果还不错。废话不多说，先看看三者是如何使用NIO的。

**Jetty Connector的实现**

先看看有关类图：



其中：

SelectChannelConnector负责组装各组件

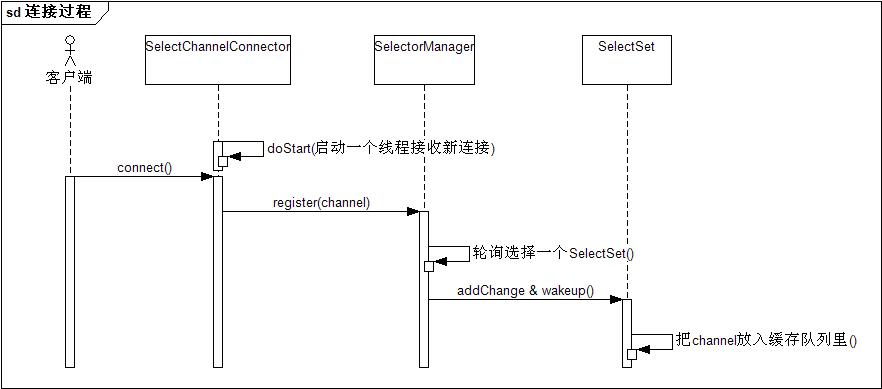
SelectSet负责侦听客户端请求

SelectChannelEndPoint负责IO的读和写

HttpConnection负责逻辑处理

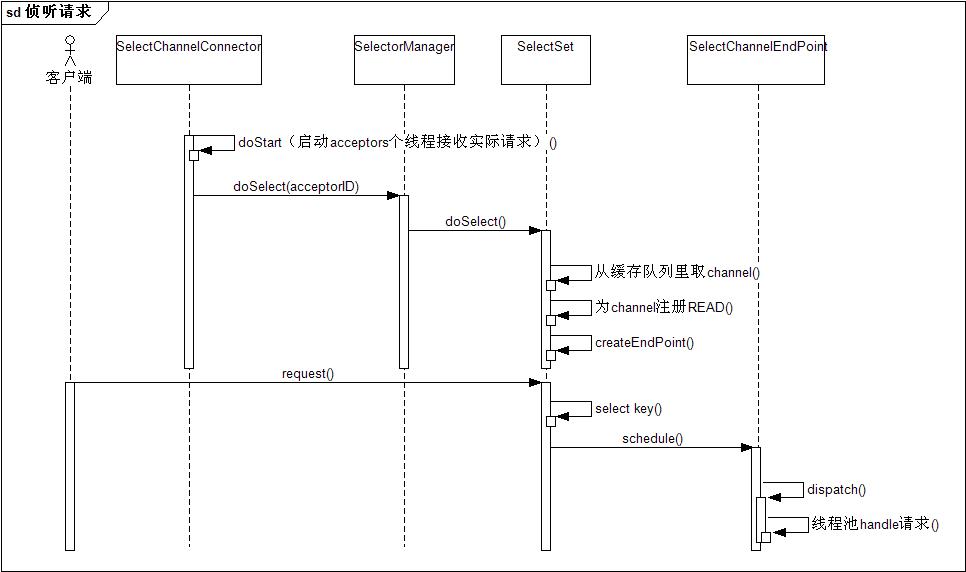
在整个服务端处理请求的过程可以分为三个阶段，时序图如下所示：

**阶段一：监听并建立连接**



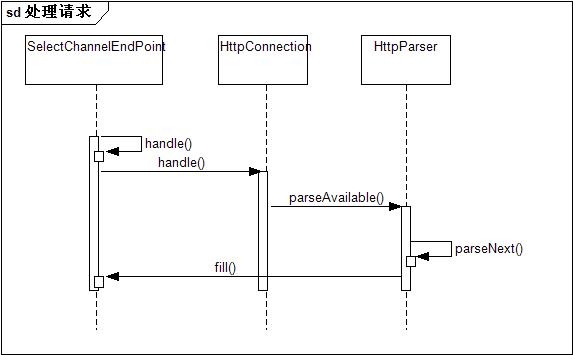
这一过程主要是启动一个线程负责accept新连接，监听到后分配给相应的SelectSet，分配的策略就是轮询。

**阶段二：监听客户端的请求**



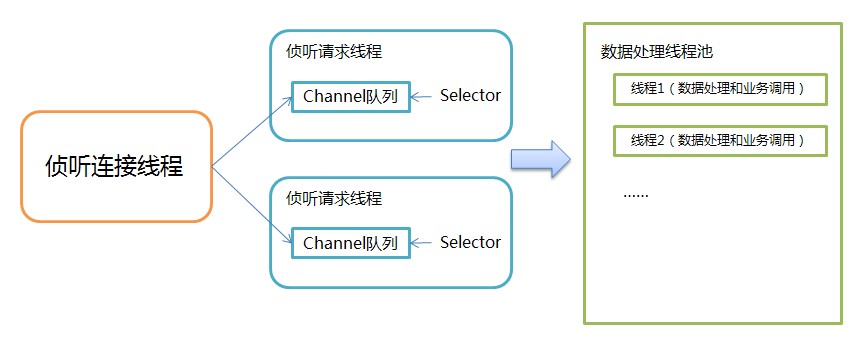
这一过程主要是启动多个线程（线程数一般为服务器CPU的个数），让SelectSet监听所管辖的channel队列，每个SelectSet维护一个Selector，这个Selector监听队列里所有的channel，一旦有读事件，从线程池里拿线程去做处理请求

**阶段三：处理请求**



这一过程就是每次客户端请求的数据处理过程，值得注意的是为了不让后端的业务处理阻碍Selector监听新的请求，就多线程来分隔开监听请求和处理请求两个阶段。

由此可以大致总结出Jetty有关NIO使用的模式，如下图所示：

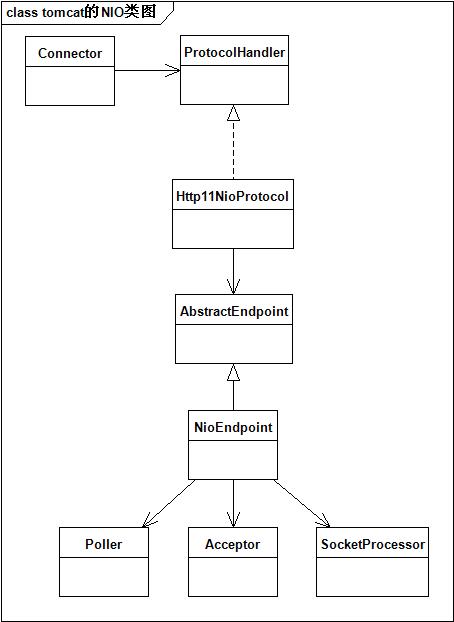


最核心就是把三件不同的事情隔离开，并用不同规模的线程去处理，最大限度地利用NIO的异步和通知特性

# 原创  从Jetty、Tomcat和Mina中提炼NIO构架网络服务器的经典模式（二） [收藏](javascript:d=document;t=d.selection?(d.selection.type!='None'?d.selection.createRange().text:''):(d.getSelection?d.getSelection():'');void(saveit=window.open('http://wz.csdn.net/storeit.aspx?t='+escape(d.title)+'&u='+escape(d.location.href)+'&c='+escape(t),'saveit','scrollbars=no,width=590,height=300,left=75,top=20,status=no,resizable=yes'));saveit.focus();)http://blog.csdn.net/images/tuijian.gif

下面再来看看Tomcat是如何使用NIO来构架Connector这块的。

先看看Tomcat Connector这块的类图：



其中：

NioEndpoint负责组装各部件

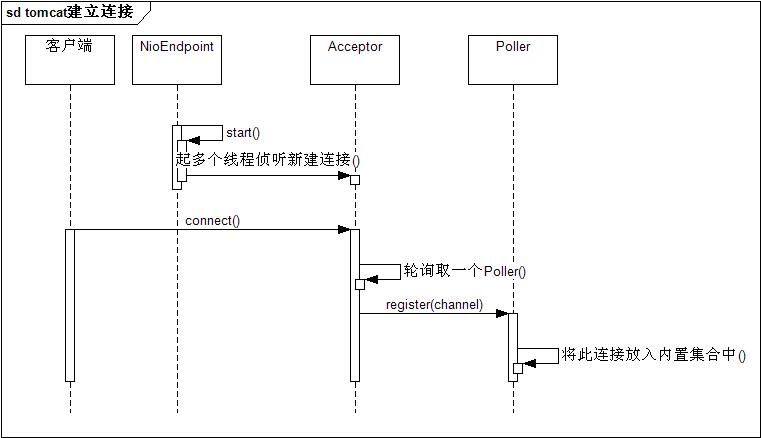
Acceptor负责监听新连接，并把连接交给Poller

Poller负责监听所管辖的channel队列，并把请求交给SocketProcessor处理

SocketProcessor负责数据处理，并把请求传递给后端业务处理模块

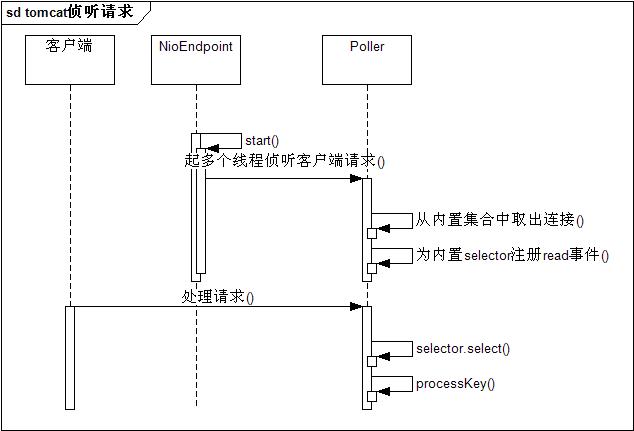
在整个服务端处理请求的过程可以分为三个阶段，时序图如下所示：

**阶段一：监听并建立连接**

****

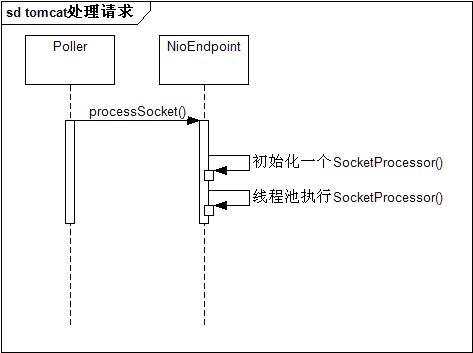
这一阶段主要是Acceptor监听新连接，并轮询取一个Poller ，把连接交付给Poller

**阶段二：** **监听客户端的请求**

****

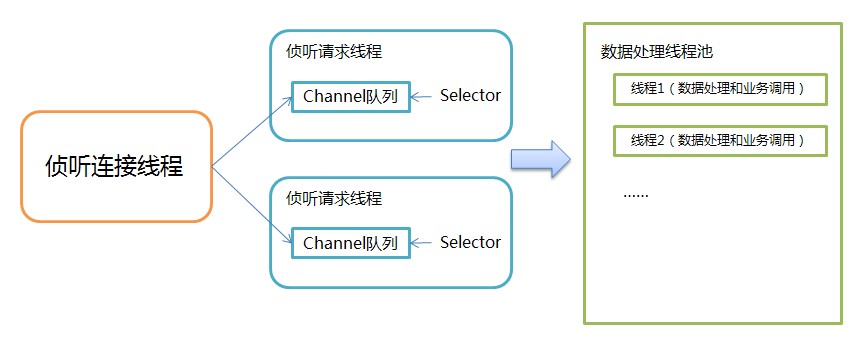
这一过程主要是让每个Poller监听所管辖的channel队列，select到新请求后交付给SocketProcessor处理

**阶段三：处理请求**

****

这一过程就是从多线程执行SocketProcessor，做数据和业务处理

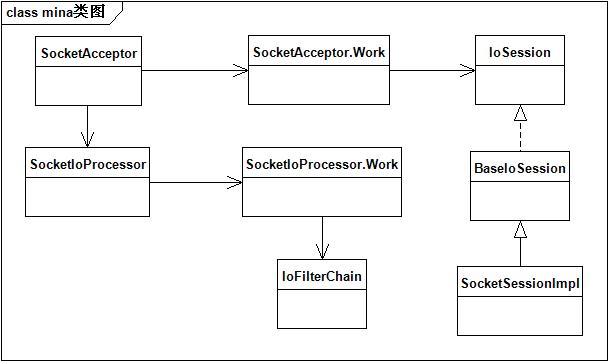
于是乎我们发现抛开具体代码细节，Tomcat和Jetty在NIO的使用方面是非常一致的，采用的模式依然是下图：

****

# 原创  从Jetty、Tomcat和Mina中提炼NIO构架网络服务器的经典模式（三） [收藏](javascript:d=document;t=d.selection?(d.selection.type!='None'?d.selection.createRange().text:''):(d.getSelection?d.getSelection():'');void(saveit=window.open('http://wz.csdn.net/storeit.aspx?t='+escape(d.title)+'&u='+escape(d.location.href)+'&c='+escape(t),'saveit','scrollbars=no,width=590,height=300,left=75,top=20,status=no,resizable=yes'));saveit.focus();)

最后我们再看看NIO方面最著名的框架Mina，抛开Mina有关session和处理链条等方面的设计，单单挑出前端网络层处理来看，也采用的是与Jetty和Tomcat类似的模式，只不过它做了些简化，它没有隔开请求侦听和请求处理两个阶段，因此，宏观上看它只分为两个阶段。

先看看它的类图：



其中：

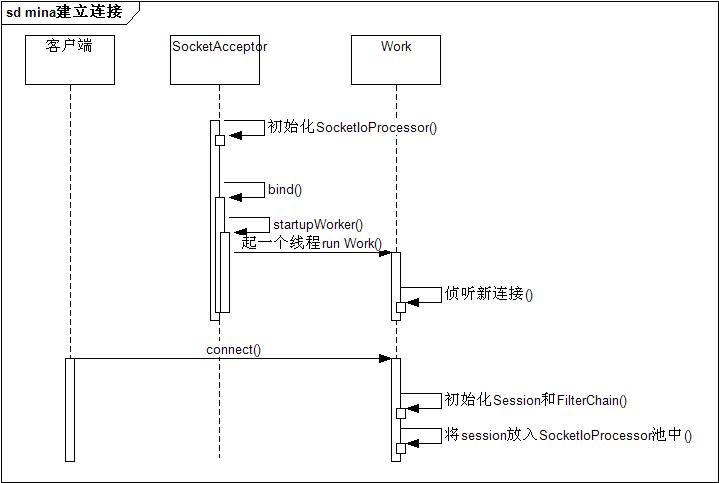
SocketAcceptor起线程调用SocketAcceptor.Work负责新连接侦听，并交给SocketIoProcessor处理

SocketIoProcessor起线程调用SocketIoProcessor.Work负责侦听所管辖的channel队列， select到新请求后交给IoFilterChain处理

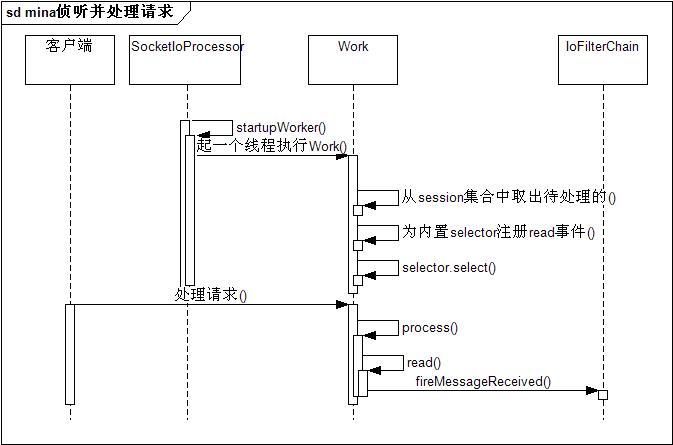
IoFilterChain组装了mina的处理链条

在整个服务端处理请求的过程可以分为两个阶段，时序图如下所示：

**阶段一：监听并建立连接**

****

**阶段二：** **监听并处理客户端的请求**

****

总结来看Jetty、tomcat和Mina，我们也大概清楚了该如何基于NIO来构架网络服务器，通过这个提炼出来的模式，我写了个很简单的NIO Server，在保持连接的情况下，可以很轻松的保持6万连接（由于有65535连接限制），并能在负载只有3左右的情况下（4核），承担3到4万的TPS请求（当然做的事情很简单，仅仅是把buffer转化为自定义协议的包，然后再把包转为buffer写到客户端）。因此简单地实践一下可以证明这个模式的有效性，不妨再看看这个图，希望对大伙以后写server有用：

