**https://blog.csdn.net/robinjwong/article/details/41792623**

**Selector(选择器)是Java NIO中能够检测一到多个NIO通道，并能够知晓通道是否为诸如读写事件做好准备的组件。这样，一个单独的线程可以管理多个channel，从而管理多个网络连接**。

NIO模式的基本原理描述如下：

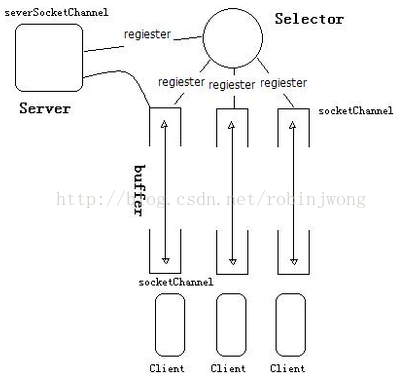
服务端打开一个通道（ServerSocketChannel），并向通道中注册一个选择器（Selector），这个选择器是与一些感兴趣的操作的标识（SelectionKey，即通过这个标识可以定位到具体的操作，从而进行响应的处理）相关联的，然后基于选择器（Selector）轮询通道（ServerSocketChannel）上注册的事件，并进行相应的处理。

客户端在请求与服务端通信时，也可以向服务器端一样注册（比服务端少了一个SelectionKey.OP\_ACCEPT操作集合），并通过轮询来处理指定的事件，而不必阻塞。

要点一：不推荐直接写channel，而是通过缓存和attachment传入要写的数据，改变interestOps()来写数据；

要点二：**每个channel只对应一个SelectionKey**，所以，只能改变interestOps()，不能register()和cancel()。

NIO的通讯过程：



使用Selector

仅用单个线程来处理多个Channels的好处是，只需要更少的新城来处理通道。事实上，可以只用一个线程处理所有的通道。

一旦调用了select()方法，并且返回值表明有一个或更多个通道就绪了，然后可以通过调用selector的selectorKeys()方法，访问”已选择键集“中的就绪通道，

Set selectedKeys = selector.selectedKeys();

可以遍历这个已选择的集合来访问就绪的通道：

Set selectedKeys = selector.selectedKeys();

Iterator keyIterator = selectedKeys.iterator();

while(keyIterator.hasNext()){

    SelectionKey key = keyIterator.next();

    if (key.isAcceptable()){      // a connection was accepted by a ServerSocketChannel

      }else

    if (key.isConnectable()){     // a connection was eatablished with a remote server

      }else

    if (key.isReadable()){        // a channel is ready for reading

      }else

    if (key.isWritable()){        // a channel is ready for writing

      }

      keyIterator.remove();  // **需要手工删除**

}