作者	正文

singleant
等级: 初级会员

性别:
文章: 18
积分: 80
来自: 杭州

发表时间：15 小时前

JDK7 AIO初体验

JDK7已经release一段时间了，有个重要的新特性是AIO。
今天趁闲暇，简单体验了下，简单分享如下：

关于AIO的概念理解

关于AIO的概念，仅谈谈个人的一点理解。可能不到位，请大家指出。
Io的两个重要步骤：发起IO请求，和实际的IO操作。在unix网络编程的定义里异步和非异步概念的区别就是实际的IO操作是否是由操作系统完成。如果是就是异步，如果不是就是同步。
而阻塞和非阻塞的区别在于发起IO请求的时候是否会阻塞，如果会就是阻塞，不会就是非阻塞。

本人理解能力有限，想了个例子来辅助自己理解：
小明想要买一本<深入java虚拟机>的书，以下几个场景可以来理解这几种io模式：

1. 如果小明每天都去书店问售货员说有没有这本书，如果没有就回去继续等待，等下次再过来文。(阻塞)
2. 如果小明告诉售货员想买一本<深入java虚拟机>的书，那么就在家里等着做其他事情去了，如果书到了售货员就通知小明，小明再自己过去取。
3. 如果小明告售货员想买一本<深入java虚拟机>的书，然后告诉售货员到了帮他送到某某地方去，就做其他事情去了。小明就不管了，等书到了，售货员就帮他送到那个地方了。

售货员可以认为是操作系统的一个服务，而小明是一个用户进程。不知道是否有误，如果有误请大家拍砖指出，谢谢。
可以看出2,3的效率明显要比1高。但是1最简单，而2,3需要一些协作。充分证明了团队合作的力量。

JDK7 AIO初体验

AsynchronousChannel：支持异步通道，包括服务端AsynchronousServerSocketChannel和普通AsynchronousSocketChannel等实现。
CompletionHandler：用户处理器。定义了一个用户处理就绪事件的接口，由用户自己实现，异步io的数据就绪后回调该处理器消费或处理数据。
AsynchronousChannelGroup：一个用于资源共享的异步通道集合。处理IO事件和分配给CompletionHandler。(具体这块还没细看代码，后续再分析这块)

以一个简单监听服务端为例，基本过程是：

1. 启动一个服务端通道
2. 定义一个事件处理器，用户事件完成的时候处理，如消费数据。
3. 向系统注册一个感兴趣的事件，如接受数据，并把事件完成的处理器传递给系统。
4. 都已经交待完毕，可以只管继续做自己的事情了，操作系统在完成事件后通过其他的线程会自动调用处理器完成事件处理。

以下用一个例子来简单实现，一个服务端和客户端。服务端监听客户端的消息，并打印出来。

AIOServer.java

Java代码

```java
package io.aio;

import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousServerSocketChannel;
import java.nio.channels.AsynchronousSocketChannel;
import java.nio.channels.CompletionHandler;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

/**
 *
 * @author noname
 */
public class AIOServer {
        public final static int PORT = 9888;
        private AsynchronousServerSocketChannel server;

        public AIOServer() throws IOException {
                server = AsynchronousServerSocketChannel.open().bind(
                                new InetSocketAddress(PORT));
        }

        public void startWithFuture() throws InterruptedException,
                        ExecutionException, TimeoutException {
                System.out.println("Server listen on " + PORT);
                Future<AsynchronousSocketChannel> future = server.accept();
                AsynchronousSocketChannel socket = future.get();
                ByteBuffer readBuf = ByteBuffer.allocate(1024);
                readBuf.clear();
                socket.read(readBuf).get(100, TimeUnit.SECONDS);
                readBuf.flip();
                System.out.printf("received message:" + new String(readBuf.array()));
                System.out.println(Thread.currentThread().getName());

        }

        public void startWithCompletionHandler() throws InterruptedException,
                        ExecutionException, TimeoutException {
                System.out.println("Server listen on " + PORT);
                //注册事件和事件完成后的处理器
                server.accept(null,
                                new CompletionHandler<AsynchronousSocketChannel, Object>() {
                                        final ByteBuffer buffer = ByteBuffer.allocate(1024);

                                        public void completed(AsynchronousSocketChannel result,
                                                        Object attachment) {
                                                System.out.println(Thread.currentThread().getName());
                                                System.out.println("start");
                                                try {
                                                        buffer.clear();
```

```
                                                result.read(buffer).get(100, TimeUnit.SECONDS);

                                                buffer.flip();
                                                System.out.println("received message: "
                                                                + new String(buffer.array()));

                                        } catch (InterruptedException | ExecutionException e) {

                                                System.out.println(e.toString());
                                        } catch (TimeoutException e) {
                                                e.printStackTrace();
                                        } finally {

                                                try {
                                                        result.close();
                                                        server.accept(null, this);
                                                } catch (Exception e) {
                                                        System.out.println(e.toString());
                                                }
                                        }

                                                System.out.println("end");
                                }

                                @Override
                                public void failed(Throwable exc, Object attachment) {
                                        System.out.println("failed: " + exc);
                                }
                        });
                // 主线程继续自己的行为
                while (true) {
                        System.out.println("main thread");
                        Thread.sleep(1000);
                }

        }

        public static void main(String args[]) throws Exception {
                new AIOServer().startWithCompletionHandler();
        }
}
```

AIOClient.java

Java代码

```
package io.aio;

import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousSocketChannel;

public class AIOClient {

        public static void main(String... args) throws Exception {
                AsynchronousSocketChannel client = AsynchronousSocketChannel.open();
                client.connect(new InetSocketAddress("localhost", 9888));
                client.write(ByteBuffer.wrap("test".getBytes())).get();
        }
}
```

服务端写了两种处理实现方式，startWithCompletionHandler是通过Handler来处理，startWithFuture是通过Future方式来处

理。startWithCompletionHandler方法里可以看到调用accepte()完成异步注册后，线程就可以继续自己的处理了，完全不被这个io所中断。
从以上来看AIO的代码简单了很多，至少比NIO的代码实现简单很多。

本文理解暂处于体验阶段，深入学习还待后面，欢迎一起交流。

参考资料

http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html

http://blog.csdn.net/roger_77/article/details/1555170

http://download.oracle.com/javase/tutorial/essential/io/file.html

返回顶楼 主页 资料 短信 留言 关注

beihan007
等级: 初级会员

性别:
文章: 14
积分: 30
来自: 成都
我现在离线

发表时间：13 小时前

顶LZ。AIO还没有研究过。看了以后有了一定的了解，有时间去研究下。

返回顶楼 主页 资料 短信 留言 关注 回帖地址 0 0 请登录后投票

jentrees
等级: 初级会员

性别:
文章: 9
积分: 70

发表时间：9 分钟前

有钻研精神，赞一个！

来自: 杭州

我现在在线

返回顶楼 | 主页 | 资料 | 短信 | 留言 | 关注 | 回帖地址 | 0 | 0 请登录后投票

wupuyuan

等级: 初级会员

性别:

文章: 41

积分: 30

来自: 南京

我现在在线

发表时间: 7 分钟前

不错，谢谢分享，其实我一直觉得之前的NIO 和现在的AIO 实现的原理类似，有两个线程池，一个是负责接收，一个是负责处理，不知道对不对，希望和LZ讨论

返回顶楼 | 主页 | 资料 | 短信 | 留言 | 关注 | 回帖地址 | 0 | 0 请登录后投票

发表回复

论坛首页 → Java编程和Java企业应用版 → Java综合

跳转论坛: Java编程和Java企业应用

- 上海: 为为网诚聘JAVA软件开发工程师:
- 上海: 绿岸网络科技诚聘Java高级软件工程师
- 北京: 美团诚聘研发工程师（Java）
- 北京: 亿玛在线诚聘高级JAVA工程师
- 湖北: 广通信达杭州研发中心诚聘java工程师
- 浙江: 阿里巴巴诚聘Project Leader/Manager (Softwa