JDK7新特性(完整版)

作者: janeky http://janeky.iteye.net

<p>这段时间陆陆续续写了这个JDK7系列的文章，希望对大家有用。从简介到语法，到各个 特性，尽量用简单的例子来说明。更多的技术文章,欢迎访问我的blog http://www.iamcoding.com</p>

目 录

1. jdk7

1.1 JDK7新特性<一>概述 3

1.2 JDK7新特性<二> 语法 7

1.3 JDK7新特性<三> JDBC4.1 13

1.4 JDK7新特性<四> NIO2.0 文件系统 15

1.5 JDK7新特性<五> fork/join 框架 17

1.6 JDK7新特性<六> 监听文件系统的更改 23

1.7 JDK7新特性<七> 遍历文件树 26

1.8 JDK7新特性<八>异步io/AIO 29

1.1 JDK7新特性<一>概述

发表时间: 2011-05-18

JDK7至今仍未正式发布。从官方的 milestone schedule（里程碑）可知，目前已经准备就绪，处于测试准备阶段，感兴趣的可以从官方下载JDK7开发预览版。笔者根据官方的相关文档，整理了这个系列的文章。

准备

JDK7下载 http://download.java.net/jdk7/

API文档 http://download.java.net/jdk7/docs/api/

新特性

1. 虚拟机

支持动态语言

严格的类文件检查

2. 语言

语法方面的更新（请参考http://janeky.iteye.com/blog/1047799）

core 类加载器（class-loader）的架构进行了升级改进

提供关闭URLClassLoader的方法

并发框架和容器的更新（请参考http://janeky.iteye.com/blog/1047805）

3. 网络

提供更多的new I/O API （请参考http://janeky.iteye.com/blog/1047804）

filesystem支持zip/jar归档

SCTP(Stream Control Transmission Protocol)

SDP(Socket Direct Protocol)

使用Windows Vista 的IPv6 stack

TLS 1.2

4. 安全相关

Elliptic-curve cryptography (ECC)

5. 国际化

Unicode6.0

Local 增强

区别 user local 和 user-interface local

6. jdbc

JDBC4.1 (请参考http://janeky.iteye.com/blog/1047800)

7. client

Java 2D 提供 XRender pipeline

为 6u10 图形特性提供新的平台api

Swing 支持光圈效果 （ Nimbus look-and-feel ）

Swing JLayer 组件

8. web

更新 XML stack

9. mgmt

增强 JMX Agent 和 MBeans

（注：这篇文章发表时，JDK7未正式公布，可能有误差，具体以官方正式版为准）

参考资料

Jdk7官网 http://openjdk.java.net/projects/jdk7/

更多的jdk7文章，欢迎访问http://janeky.iteye.com/category/157060

1.2 JDK7新特性<二> 语法

发表时间: 2011-05-18

JDK7对Java语法有少量更新，重点是在易用性和便捷性的改进。

1.二进制字面量

JDK7开始，终于可以用二进制来表示整数（byte,short,int和long）。使用二进制字面量的好处是，可以是代码更容易被理解。语法非常简单，只要在二进制数值前面加 0b或者0B

```
byte nByte = (byte)0b0001;
short nShort = (short)0B0010;
int nInt = 0b0011;
long nLong = 0b0100L;
```

2.数字字面量可以出现下划线

对于一些比较大的数字，我们定义起来总是不方面，经常缺少或者增加位数。JDK7为我们提供了一种解决方案，下划线可以出现在数字字面量。

```
int a = 10_0000_0000;
long b = 0xffff_ffff_ffff_ffffl;
byte c = 0b0001_1000;
```

注意：你只能将下划线置于数字之间，以下使用方法是错误的，

1.数字的开头或者结尾

2.小数点的前后

3. ‘F’ 或者 ‘f’ 的后缀

4.只能用数字的位置

```
int err1 = _11,err2=11_;
float err3=3._4,err4=3_.4;
long err5=0x888_f;
```

3.switch 语句可以用字符串了

这个功能千呼万唤，终于出来了

```
private static void switchString(String str){
        switch(str){
            case "one":
                System.err.println("1");
                break;
            case "two":
                System.out.println("2");
                break;
            default :
                System.out.println("err");
        }
}
```

4.泛型实例的创建可以通过类型推断来简化

以后你创建一个泛型实例，不需要再详细说明类型，只需用<>,编译器会自动帮你匹配

```
//例如
Map<String, List<String>> myMap = new HashMap<String, List<String>>();
```

```
//可以简化为
Map<String, List<String>> myMap = new HashMap<>();
```

5.在可变参数方法中传递非具体化参数（Non-Reifiable Formal Parameters）,改进编译警告和错误

有些参数类型，例如ArrayList<Number> 和 List<String>,是非具体化的（non-reifiable）.在编译阶段，编译器会擦除该类型信息。

Heap pollution 指一个变量被指向另外一个不是相同类型的变量。例如

```
List l = new ArrayList<Number>();
      List<String> ls = l;       // unchecked warning
      l.add(0, new Integer(42)); // another unchecked warning
      String s = ls.get(0);      // ClassCastException is thrown
```

回到我们的主题，在jdk7中，当你定义下面的函数时

```
public static <T> void addToList (List<T> listArg, T... elements) {
    for (T x : elements) {
      listArg.add(x);
    }
  }
```

你会得到一个warning

```
warning: [varargs] Possible heap pollution from parameterized vararg type
```

在jdk7之前，当你调用一个含有非具体化参数的可变参数方法，你必须自行保证不会发生“heap pollution”。这有一个问题，如果调用者对方法不熟悉，他根本无法判断。JDK7对此做了改进，在该方法被定义时久发出警告

要消除警告，可以有三种方式

1.加 annotation @SafeVarargs

2.加 annotation @SuppressWarnings({"unchecked", "varargs"})

3.使用编译器参数 –Xlint:varargs;

6.try-with-resources 语句

jdk7提供了try-with-resources,可以自动关闭相关的资源（只要该资源实现了AutoCloseable接口，jdk7为绝大部分资源对象都实现了这个接口）

```
static String readFirstLineFromFile(String path) throws IOException {
  try (BufferedReader br = new BufferedReader(new FileReader(path))) {
    return br.readLine();
  }
}
```

try 语句块中还可以同时处理多个资源,可以跟普通的try语句一样catch异常，有finally语句块

```
try (
      java.util.zip.ZipFile zf = new java.util.zip.ZipFile(zipFileName);
      java.io.BufferedWriter writer = java.nio.file.Files.newBufferedWriter(outputFilePath, cha
    ) {
}catch(...){
}finally{
}
```

7.Catch多个Exception，rethrow exception 改进了类型检测

很多时候，我们捕获了多个异常，却做了相同的事情，比如记日志，包装成新的异常，然后rethrow。这时，代码就不那么优雅了，例如

```
catch (IOException ex) {
     logger.log(ex);
     throw ex;
catch (SQLException ex) {
     logger.log(ex);
     throw ex;
}
```

Jdk7允许捕获多个异常

```
catch (IOException|SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

注意，catch后面的异常参数是final的，不能重新再复制

Rethrow Exception更具包容性的类型检测

当你重新抛出多个异常时，不再需要详细定义异常类型了，编译器已经知道你具体抛出的是哪个异常了。你只需在方法定义的时候声明需要抛出的异常即可

```
public void call() throws ReflectiveOperationException, IOException {
    try {
      callWithReflection(arg);
    } catch (final Exception e) {
      logger.trace("Exception in reflection", e);
      throw e;
    }
}
```

参考资料

Jdk7官网 http://openjdk.java.net/projects/jdk7/

（注：这篇文章发表时，JDK7未正式公布，可能有误差，具体以官方正式版为准）

更多的jdk7文章，欢迎访问

http://janeky.iteye.com/category/157060

1.3 JDK7新特性<三> JDBC4.1

发表时间: 2011-05-18

JDBC4.1更新了两个新特性

1. Connection，ResultSet 和 Statement 都实现了Closeable 接口，所有在 try-with-resources 语句中调用，就可以自动关闭相关资源了

```
try (Statement stmt = con.createStatement()){
                …
}
```

2. RowSet 1.1：引入RowSetFactory接口和RowSetProvider类，可以创建JDBC driver支持的各种 row sets

```
    RowSetFactory myRowSetFactory = null;
    JdbcRowSet jdbcRs = null;
    ResultSet rs = null;
    Statement stmt = null;

    try {

      myRowSetFactory = RowSetProvider.newFactory();//用缺省的RowSetFactory 实现
      jdbcRs = myRowSetFactory.createJdbcRowSet();

      //创建一个 JdbcRowSet 对象，配置数据库连接属性
      jdbcRs.setUrl("jdbc:myDriver:myAttribute");
      jdbcRs.setUsername(username);
      jdbcRs.setPassword(password);

      jdbcRs.setCommand("select ID from TEST");
```

```
    jdbcRs.execute();
  }
```

RowSetFactory 接口包括了创建不同类型的RowSet的方法

•createCachedRowSet

•createFilteredRowSet

•createJdbcRowSet

•createJoinRowSet

•createWebRowSet

参考资料

Jdk7官网 http://openjdk.java.net/projects/jdk7/

（注：这篇文章发表时，JDK7未正式公布，可能有误差，具体以官方正式版为准）

更多的jdk7文章，欢迎访问http://janeky.iteye.com/category/157060

1.4 JDK7新特性<四> NIO2.0 文件系统

发表时间: 2011-05-18

java.io.File 不够完美吧。Jdk7提供了一套新的文件系统，会让你满意的。

先来聊聊java.io.File的七宗罪吧：)

1.很多方法失败时候都没有抛出异常，很难查找原因

2.方法 rename 在不同平台中运行有问题

3.不能真正支持 symbolic links

4.不能读取文件的更详细属性，比如权限，所有者……

5.访问 文件的 metadata 效率低下

6.很多方法性能不行。例如处理比较大的目录

7.无法递归查找文件树，以及存在循环的symbolic links可能造成问题

本次jdk7更新了很多新的api。方法太多了，我就不一一列举了，感兴趣的可以去查阅api

```
http://download.java.net/jdk7/docs/api/java/nio/file/package-summary.html
```

主要包括：

FileSystem 提供了许多方法来获得当前文件系统的相关信息。

Path 处理路径(文件和目录)，包括

创建path，Paths.get(String s)

获得path的详细信息 getName(),getXX()...

删除path的冗余信息 toRealPath

转换path toAbsolutePath()

合并两个path resolve()

在两个path之间创建相对路径 relativeze()

比较路径 equal() startsWith(),endWith()

Files 支持各种文件操作，包括

移动文件，
复制文件，
删除文件，
更详细的文件属性，包括文件权限，创建者，修改时间……
Walking the File Tree（递归遍历文件树）
Watch a Directory for Change （监听文件更改）

（最后两点，我近期会更新一些相关的范例）

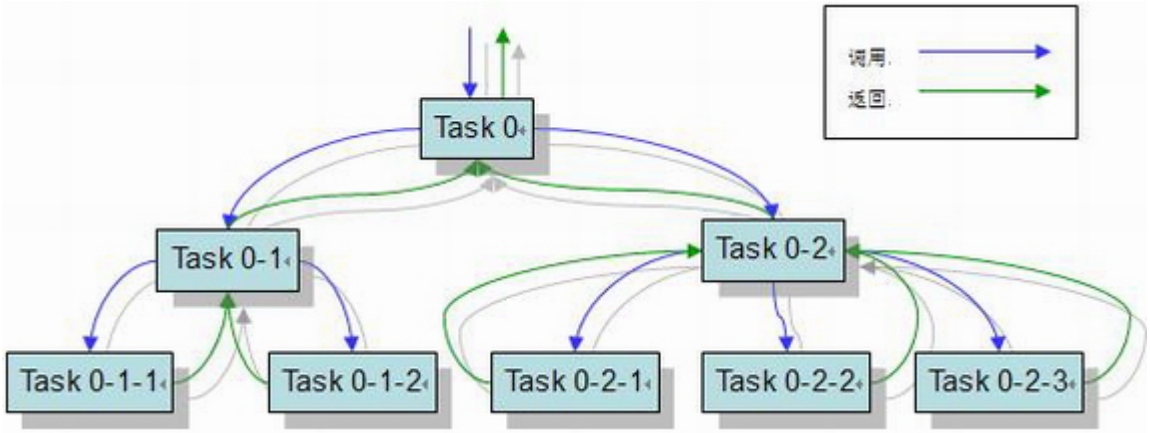
更多的jdk7文章，欢迎访问http://janeky.iteye.com/category/157060

1.5 JDK7新特性<五> fork/join 框架

发表时间: 2011-05-18

对于框架的原理，可以阅读 Doug Lea 的文章“A Java Fork/Join Framework”：了解 Fork/Join 模式的实现机制和执行性能。

原理解析：fork分解，join结合。这个框架的本质是将一个任务分解成多个子任务，每个子任务用单独的线程去处理。这里用到了递归的思想。框架的结构图可以参考



图片来源（http://www.ibm.com/developerworks/cn/java/j-lo-forkjoin/index.html）

使用fork/join 框架很简单，

1.实现子问题的一般求解算法

2.如何分解问题

3.继承 RecursiveAction ，实现compute()方法

```
Result solve(Problem problem) {
  if (problem is small)
```

```
                directly solve problem
        else {
                split problem into independent parts
                fork new subtasks to solve each part
                join all subtasks
                compose result from subresults
        }
}
```

这里我通过一个改进的二分查找来讲解fork/join的使用。（后面才发现，选用这个案例是非常失败的，因为二分查找的时间是logn,而创建线程的开销更大，这样并不能体现多线程二分查找的优势，所以这个代码不具有实用性，只是为了说明如何使用框架：）

代码如下：

BinarySearchProblem.java

```
package testjdk7;

import java.util.Arrays;
/**
 * @author kencs@foxmail.com
 */
public class BinarySearchProblem {
    private final int[] numbers;
    private final int start;
    private final int end;
    public final int size;
```

```
    public BinarySearchProblem(int[] numbers,int start,int end){
        this.numbers = numbers;
        this.start = start;
        this.end = end;
        this.size = end -start;
    }

    public int searchSequentially(int numberToSearch){
       //偷懒，不自己写二分查找了
       return Arrays.binarySearch(numbers, start, end, numberToSearch);
    }

    public BinarySearchProblem subProblem(int subStart,int subEnd){
        return new BinarySearchProblem(numbers,start+subStart,start+subEnd);
    }
}
```

BiSearchWithForkJoin.java

```
package testjdk7;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;

/**
 * @author kencs@foxmail.com
 */
public class BiSearchWithForkJoin extends RecursiveAction {
    private final int threshold;
    private final BinarySearchProblem problem;
    public int result;
    private final int numberToSearch;

    public BiSearchWithForkJoin(BinarySearchProblem problem,int threshold,int numberToSearch){
```

```
        this.problem = problem;
        this.threshold = threshold;
        this.numberToSearch = numberToSearch;
    }

    @Override
    protected void compute() {
       if(problem.size < threshold){ //小于阀值，就直接用普通的二分查找
           result = problem.searchSequentially(numberToSearch);
       }else{
           //分解子任务
           int midPoint = problem.size/2;
           BiSearchWithForkJoin left = new BiSearchWithForkJoin(problem.subProblem(0, midPoint)
           BiSearchWithForkJoin right = new BiSearchWithForkJoin(problem.subProblem(midPoint+1,
           invokeAll(left,right);
           result = Math.max(left.result, right.result);
       }
    }

    //构造数据
    private static final int[] data = new int[1000_0000];
    static{
        for(int i = 0;i<1000_0000;i++){
            data[i] = i;
        }
    }
    public static void main(String[] args){
       BinarySearchProblem problem = new BinarySearchProblem(data,0,data.length);
       int threshold = 100;
       int nThreads = 10;
       //查找100_0000所在的下标
       BiSearchWithForkJoin  bswfj = new BiSearchWithForkJoin(problem,threshold,100_0000);
       ForkJoinPool fjPool = new ForkJoinPool(nThreads);
       fjPool.invoke(bswfj);
       System.out.printf("Result is:%d%n",bswfj.result);
    }
```

```
}
```

RecursiveTask 还可以带返回值，这里给出一段代码作为参考（斐波那契函数）

（来自http://www.ibm.com/developerworks/cn/java/j-lo-forkjoin/index.html）

```
class Fibonacci extends RecursiveTask<Integer> {
    final int n;

    Fibonacci(int n) {
        this.n = n;
    }

    private int compute(int small) {
        final int[] results = { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 };
        return results[small];
    }

    public Integer compute() {
        if (n <= 10) {
            return compute(n);
        }
        Fibonacci f1 = new Fibonacci(n - 1);
        Fibonacci f2 = new Fibonacci(n - 2);
        System.out.println("fork new thread for " + (n - 1));
        f1.fork();
        System.out.println("fork new thread for " + (n - 2));
        f2.fork();
        return f1.join() + f2.join();
    }
}
```

用途

只要问题能够分解成类似子问题的，都可以使用这个框架。对于大批量的数据尤其合适

参考资料

Jdk7官网 http://openjdk.java.net/projects/jdk7/

（注：这篇文章发表时，JDK7未正式公布，可能有误差，具体以官方正式版为准）

更多的jdk7文章，欢迎访问http://janeky.iteye.com/category/157060

1.6 JDK7新特性<六> 监听文件系统的更改

发表时间: 2011-05-19

我们用IDE（例如Eclipse）编程，外部更改了代码文件，IDE马上提升“文件有更改”。Jdk7的NIO2.0也提供了这个功能，用于监听文件系统的更改。它采用类似观察者的模式，注册相关的文件更改事件（新建，删除……），当事件发生的，通知相关的监听者。

java.nio.file.*包提供了一个文件更改通知API,叫做Watch Service API.

实现流程如下

1.为文件系统创建一个WatchService 实例 watcher

2.为你想监听的目录注册 watcher。注册时，要注明监听那些事件。

3.在无限循环里面等待事件的触发。当一个事件发生时，key发出信号，并且加入到watcher的queue

4.从watcher的queue查找到key，你可以从中获取到文件名等相关信息

5.遍历key的各种事件

6.重置 key，重新等待事件

7.关闭服务

```
import java.io.IOException;
import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.WatchEvent;
```

```
import java.nio.file.WatchKey;
import java.nio.file.WatchService;
import static java.nio.file.StandardWatchEventKind.*;

/**
 * @author kencs@foxmail.com
 */
public class TestWatcherService {

    private WatchService watcher;

    public TestWatcherService(Path path)throws IOException{
        watcher = FileSystems.getDefault().newWatchService();
        path.register(watcher, ENTRY_CREATE,ENTRY_DELETE,ENTRY_MODIFY);
    }

    public void handleEvents() throws InterruptedException{
        while(true){
            WatchKey key = watcher.take();
            for(WatchEvent<?> event : key.pollEvents()){
                WatchEvent.Kind kind = event.kind();

                if(kind == OVERFLOW){//事件可能lost or discarded
                    continue;
                }

                WatchEvent<Path> e = (WatchEvent<Path>)event;
                Path fileName = e.context();

                System.out.printf("Event %s has happened,which fileName is %s%n"
                        ,kind.name(),fileName);
            }
            if(!key.reset()){
                break;
            }
        }
    }
```

```
    public static void main(String args[]) throws IOException, InterruptedException{
        if(args.length!=1){
            System.out.println("请设置要监听的文件目录作为参数");
            System.exit(-1);
        }
        new TestWatcherService(Paths.get(args[0])).handleEvents();
    }
}
```

接下来，见证奇迹的时刻

1.随便新建一个文件夹 例如 c:\\test

2.运行程序 java TestWatcherService c:\\test

3.在该文件夹下新建一个文件本件 “新建文本文档.txt”

4.将上述文件改名为 “abc.txt”

5.打开文件，输入点什么吧，再保存。

6.Over！看看命令行输出的信息吧

```
Event ENTRY_CREATE has happened,which fileName is 新建文本文档.txt
Event ENTRY_DELETE has happened,which fileName is 新建文本文档.txt
Event ENTRY_CREATE has happened,which fileName is abc.txt
Event ENTRY_MODIFY has happened,which fileName is abc.txt
Event ENTRY_MODIFY has happened,which fileName is abc.txt
```

更多的jdk7文章，欢迎访问http://janeky.iteye.com/category/157060

1.7 JDK7新特性<七> 遍历文件树

发表时间: 2011-05-20

有时需要递归遍历一个文件树，比如查找一个文件夹内符合条件的文件，查找某一天创建的文件……。jdk7 nio包提供一个新的接口 FileVisitor。它提供了遍历文件树的各种操作。

preVisitDirectory - 一个路径被访问时调用

PostVisitDirectory - 一个路径的所有节点被访问后调用。如果有错误发生，exception会传递给这个方法

visitFile - 文件被访问时被调用。该文件的文件属性被传递给这个方法

visitFileFailed - 当文件不能被访问时，此方法被调用。Exception被传递给这个方法。

如果你比较懒，不想实现所有方法。你可以选择继承 SimpleFileVisitor。它帮你实现了上述方法，你只需Override 你感兴趣的方法。

下面给个例子，简单地遍历一个文件夹，打印出所有信息

```
import java.io.IOException;
import java.nio.file.FileVisitResult;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.SimpleFileVisitor;
import java.nio.file.attribute.BasicFileAttributes;
```

```
/**
 * @author kencs@foxmail.com
 */
public class FileVisitorTest extends SimpleFileVisitor<Path> {

    private void find(Path path){
        System.out.printf("访问-%s:%s%n",(Files.isDirectory(path)?"目录":"文件"),path.getFileNam
    }
    @Override
    public FileVisitResult visitFile(Path file,BasicFileAttributes attrs){
        find(file);
        return FileVisitResult.CONTINUE;
    }

    @Override
    public FileVisitResult preVisitDirectory(Path dir,BasicFileAttributes attrs){
        find(dir);
        return FileVisitResult.CONTINUE;
    }

    @Override
    public FileVisitResult visitFileFailed(Path file,IOException e){
        System.out.println(e);
        return FileVisitResult.CONTINUE;
    }

    public static void main(String[] args) throws IOException{
        if(args.length!=1){
            System.out.println("请输入一个文件路径作为参数");
            System.exit(-1);
        }
        Files.walkFileTree(Paths.get( args[0]), new FileVisitorTest());
    }
}
```

随便选择一个路径作为参数

java FileVisitorTest
"C:\\Program Files\\Java\\jre7\\bin"

```
访问-目录:bin
访问-文件:awt.dll
访问-文件:axbridge.dll
访问-目录:client
访问-文件:classes.jsa
访问-文件:jvm.dll
访问-文件:Xusage.txt
访问-文件:dcpr.dll
访问-文件:deploy.dll
访问-文件:deployJava1.dll
访问-文件:dt_shmem.dll
访问-文件:dt_socket.dll
......
```

注意 FileVisitResult有四种

CONTINUE –继续
TERMINATE –终止，这次遍历结束了
SKIP_SUBTREE –子树（当前路径的子目录）不再遍历了
SKIP_SIBLINGS –兄弟节点（同级别目录）不再访问了。

可以通过这些返回值来控制遍历文件树的流程

更多的jdk7文章，欢迎访问http://janeky.iteye.com/category/157060

1.8 JDK7新特性<八>异步io/AIO

发表时间: 2011-06-09

概述

JDK7引入了Asynchronous I/O。I/O编程中，常用到两种模式：Reactor 和 Proactor。Reactor就是Java的NIO。当有事件触发时，我们得到通知，进行相应的处理。Proactor就是我们今天要讲的 AIO了。AIO进行I/O操作，都是异步处理，当事件完成时，我们会得到通知。

JDK7的 AIO包括网络和文件操作。两者大同小异，本文通过一个完整的客户端/服务器Sample来详细说明aio的网络操作。

AIO提供了两种异步操作的监听机制。第一种通过返回一个Future对象来事件，调用其get（ ）会等到操作完成。第二种类似于回调函数。在进行异步操作时，传递一个CompletionHandler，当异步操作结束时，会调用CompletionHandler.complete 接口

范例

这个范例功能比较简单，就是客户端向服务端发送一个“test”命令，然后结束。

服务端程序Sever.java

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

```
public class Server {
    private AsynchronousServerSocketChannel server;

    public Server()throws IOException{
        server = AsynchronousServerSocketChannel.open().bind(new InetSocketAddress(8888));
    }

    public void start() throws InterruptedException, ExecutionException, TimeoutException{
        Future<AsynchronousSocketChannel> future = server.accept();
        AsynchronousSocketChannel socket = future.get();

        ByteBuffer readBuf = ByteBuffer.allocate(1024);
        socket.read(readBuf).get(100, TimeUnit.SECONDS);

        System.out.printf("Receiver:%s%n",new String(readBuf.array()));
    }

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

客户端程序（Future版本）

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousSocketChannel;
import java.util.concurrent.ExecutionException;

public class AIOClientWithFuture {
    private final AsynchronousSocketChannel client;

    public AIOClientWithFuture() throws IOException{
        client = AsynchronousSocketChannel.open();
```

```
    }

    public void sendMsg() throws InterruptedException, ExecutionException{
        client.connect(new InetSocketAddress("localhost",8888));
        client.write(ByteBuffer.wrap("test".getBytes())).get();
    }
    public static void main(String...args) throws Exception{
        AIOClientWithFuture client = new AIOClientWithFuture();
        client.sendMsg();
    }
}
```

客户端程序（CompleteHandler版本）

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

public class AIOClientWithHandler {
    private final AsynchronousSocketChannel client ;

    public AIOClientWithHandler() throws Exception{
       client = AsynchronousSocketChannel.open();
    }

    public void start()throws Exception{
        client.connect(new InetSocketAddress("127.0.0.1",8888),null,new CompletionHandler<Void,
            @Override
            public void completed(Void result, Void attachment) {
                try {
                    client.write(ByteBuffer.wrap("test".getBytes())).get();
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
```

```
            }

            @Override
            public void failed(Throwable exc, Void attachment) {
                exc.printStackTrace();
            }
        });
    }

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

相关类说明

AsynchronousSocketChannel 跟 SocketChannel操作类似，只不过改成异步接口了

AsynchronousServerSocketChannel跟ServerSocketChannel操作类似，只不过改成异步接口了

CompletionHandler 接口包括两个方法

```
void completed(V result, A attachment);

void failed(Throwable exc, A attachment);
```

总结

本文只是对jdk7的aio使用做了一个简单的说明。至于其性能提升多少，如何改进现有的网络应用程序，还在摸索中。这里推荐一个比较成熟的网络框架Project Grizzly:http://grizzly.dev.java.net。据说已经用aio重新实现了，有兴趣的同学可以去研究一下源码。

参考资料

（以下有些资料使用的jdk7版本太低，很多接口已经更改了，慎入!：）

http://www.ibm.com/developerworks/java/library/j-nio2-1/index.html?

http://www.iteye.com/topic/446298

http://www.iteye.com/topic/472333

本文是jdk7系列的终结了，感谢大家的支持！更多的内容可以访问我的blog

http://www.iamcoding.com

更多的技术文章 欢迎访问我的 blog <a href="http://www.iamcoding.com">http://www.iamcodin

JDK7新特性(完整版)

作者: janeky
http://janeky.iteye.net