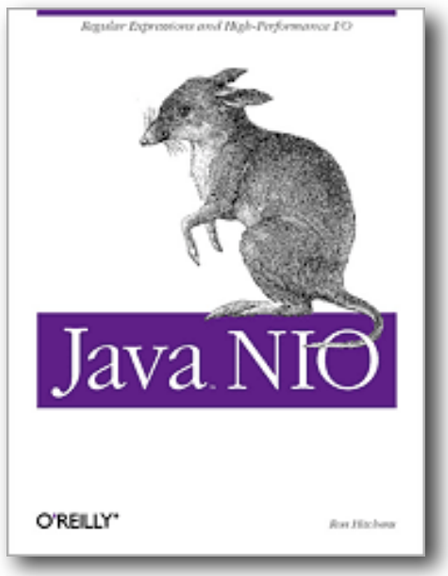
What's New in NIO.2

Ron Hitchens
Senior Engineer
Mark Logic Corporation
San Mateo, CA
ron.hitchens@marklogic.com
ron@ronsoft.com



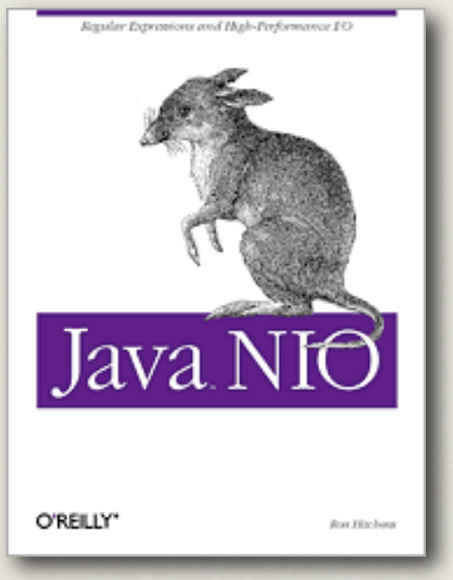
SD WEST 2008
MARCH 3-7, 2008, SANTA CLARA, CA

What's New in NIO.2

Ron Hitchens
Mark Logic Corporation

March 5, 2008

ron.hitchens@marklogic.com
ron@ronsoft.com



http://javanio.info/

What's This?

A sneak peek at JSR-203 (NIO.2)

Coming soon to a JVM near you

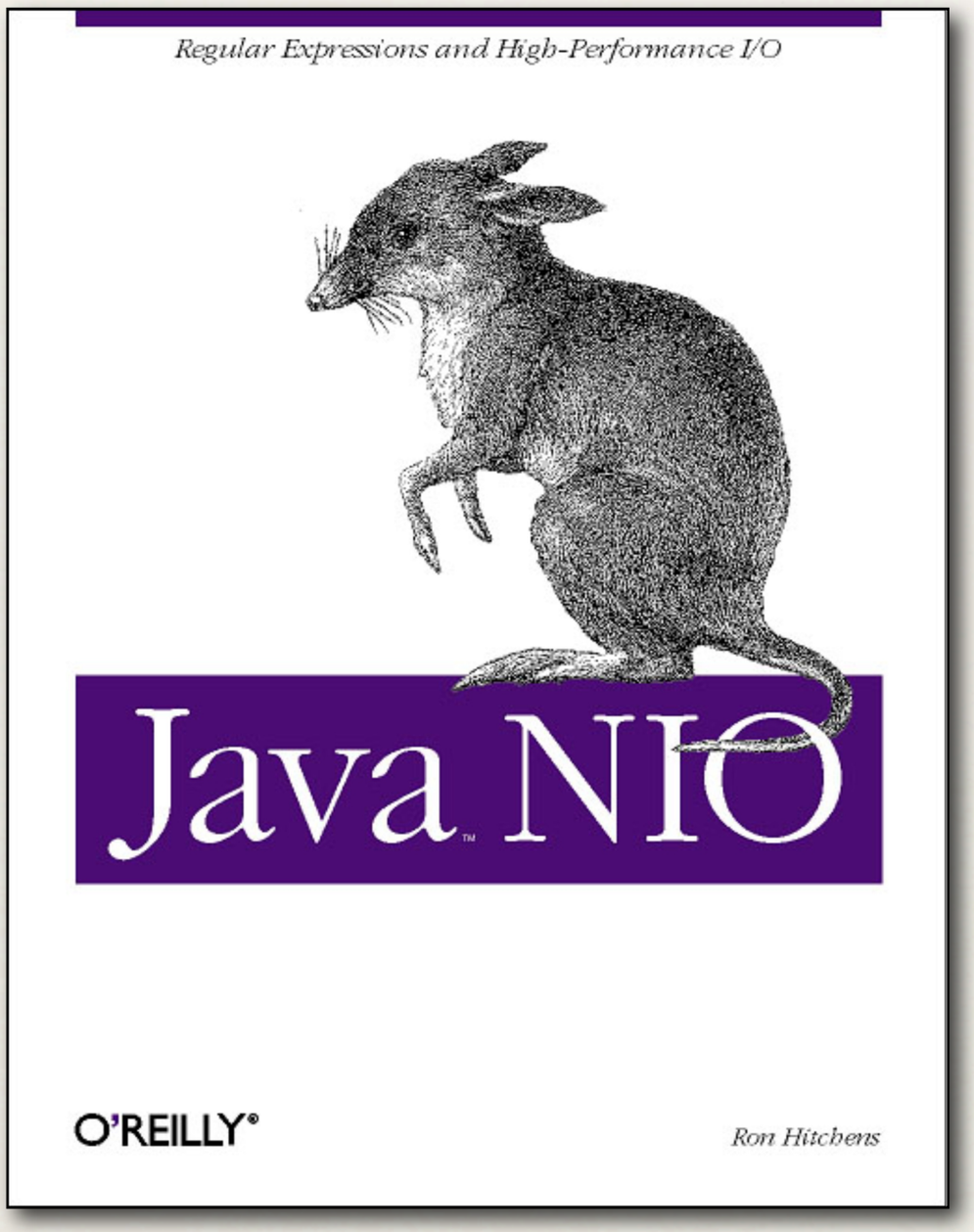
Ron Hitchens

Years spent hacking UNIX® internals

Device drivers, I/O streams, etc.

Tons O' Java since 1997

Java NIO published August 2002



Been at Mark Logic since 2004

Lots of XML, XQuery and Java, not so much NIO lately

Getting Started With XQuery (Pragmatic)

New I/O

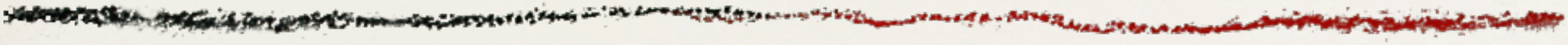
- JSR-51
  - Shipped in JDK 1.4 (2002)
  - Block-oriented I/O with buffers and channels
  - Non-blocking sockets, readiness selection
  - Also Charsets and regular expressions

- JSR-203
  - JSR opened in 2003, primary development since 2006
  - Stuff that didn't make JSR-51, plus new stuff
  - Expected to ship in JDK 1.7 (Java 7)
  - In process to be OS'ed on http://openjdk.java.net/

What's New in NIO.2

Updated
- Buffers
- Sockets
- File I/O

New
- Filesystem API
- Asynchronous I/O



Buffers

Buffers

java.nio

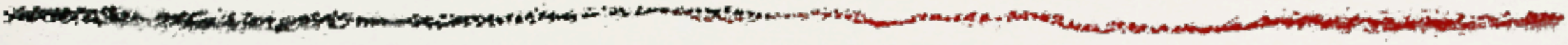
- ByteBuffers supply or receive data transferred by ByteChannels
- BigBuffer
  - BigByteBuffer, BigIntBuffer, BigShortBuffer, etc
  - Long (64 bit) indexes, to handle 2GB+ buffer sizes
    - Especially useful for MappedBigByteBuffer on huge files
- Mappable interface
  - Refactored methods from MappedByteBuffer
  - MappedByteBuffer and MappedBigByteBuffer

MBean Support

java.lang.management

- Monitor buffer pool status with JMX tools
- Primarily of interest to tool builders

```
MBeanServer server = ManagementFactory.getPlatformMBeanServer();
Set<ObjectName> mbeans = server.queryNames (
   new ObjectName ("java.nio:type=BufferPool,*"), null);


for (ObjectName name: mbeans) {
    BufferPoolMXBean pool = ManagementFactory.newPlatformMXBeanProxy (
       server, name.toString(), BufferPoolMXBean.class);
    ...
}
```



Sockets

SocketChannel

java.nio.channels

- SocketChannel API has been fleshed out
  - No longer necessary to use socket() in most cases
  - Check connection state, get remote address, shutdown, etc
- Implements new NetworkChannel interface
  - Methods common to all network channel classes
- Where practical, methods return “this” to facilitate invocation chaining
  - A design pattern used frequently in the original NIO

NetworkChannel

java.nio.channels

- NetworkChannel interface
  - Methods for binding to and returning local address
  - Methods to set and get socket options
    - Option names are typesafe SocketOption Enums
    - Option values are Object

- StandardSocketOption class
  - Contains Enum definitions
  - SO_SNDBUF, TCP_NODELAY, etc
  - Each socket type defines the options it will accept

Socket Example

```
import static java.nio.channels.StandardSocketOption.*;



ServerSocketChannel ssc = ServerSocketChannel.open()
   .setOption(SO_RCVBUF, 256*1024).bind (address, backlog);

SocketChannel sc = SocketChannel.open()
   .setOption(SO_SNDBUF, 128*1024)
   .setOption(SO_KEEPALIVE, true);


boolean nagle = (Boolean)sc.getOption(TCP_NODELAY);

Map<String,Class> options = sc.getOptions();


SocketAddress local = sc.getLocalAddress();

SocketAddress remote = sc.getConnectedAddress();

sc.shutdown(ShutdownType.READ);
```

Note autoboxing

Multicast

java.nio.channels

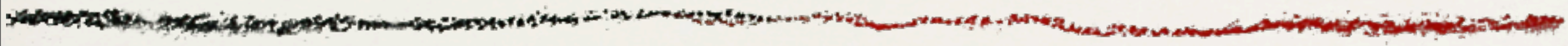
- New MulticastChannel interface
  - Only datagram channels implement it
- Join a multicast group (an InetAddress) on a specific NetworkInterface (in java.net)
  - Joining returns a MembershipKey
- Channel may choose to receive all datagrams, or only those from a specific sender
- Datagrams go to any host listening to group address
- Channel setup requires specific options
- Call drop() on MembershipKey to leave the group
- Implements the latest RFCs / newest features

Multicast Sender

```
InetAddress group = InetAddress.getByName (“multigroup.megacorp.com”);
NetworkInterface interface = NetworkInterface.getByName (“if0”);
int port = 1234;
ByteBuffer buffer = ...;

DatagramChannel channel = DatagramChannel.open (StandardProtocolFamily.INET)
    .bind (new InetSocketAddress (port));

channel.setOption (StandardSocketOption.IP_MULTICAST_IF, interface);

// Datagram will be received by anyone listening to the group address
channel.send (buffer, group);
```

Multicast Listener

```
List<InetAddress> includeList = ...;  List<InetAddress> excludeList = ...;
InetAddress group = InetAddress.getByName (“multigroup.megacorp.com”);
NetworkInterface interface = NetworkInterface.getByName (“if0”);
int port = 1234;

DatagramChannel channel = DatagramChannel.open (StandardProtocolFamily.INET)
     .setOption (StandardSocketOption.SO_REUSEADDR, true)
     .bind (new InetSocketAddress(port));

if (includeList.isEmpty()) {
     // Join the group, then explicitly ignore datagrams from certain addresses
     MembershipKey key = channel.join (group, interface);
     for (InetAddress source: excludeList) {
          key.block (source);
     }
} else {
     // Listen only to datagrams from specific senders in the group
     for (InetAddress source: includeList) {
          channel.join (target.group(), target.interf(), source);
     }
}
```



Files and Filesystems

FileChannel

java.nio.channels

- Several new methods added
  - open() - Open a file directly
  - isReadable(), isWritable()
  - mapBigBuffer() - For mapping humongous files
    - Just like map() but returns a MappedBigByteBuffer
- Implements SeekableByteChannel
  - Adds position, size and truncation methods to ByteChannel
  - Bare minimum file channel functionality
  - Easier to implement for some filesystem types

Using FileChannels

```
import static java.nio.channels.StandardOpenOption.*;

static final int ONE_MILLION = 1000000;
ByteBuffer dollars = ByteBuffer.wrap ("Dollars".getBytes());
Path path = Path.from ("WordDominationPlan.docevil");

// Create a FileChannel using specific flags
FileChannel fileChannel =
   FileChannel.open (path, WRITE, SYNC).position (ONE_MILLION);
fileChannel.write (dollars);
fileChannel.close();

// An alternate, more generic way (no file locks, mapped files, etc)
SeekableByteChannel channel =
   path.newSeekableByteChannel (WRITE, SYNC).position (ONE_MILLION);
channel.write (dollars);
channel.close();
```

Varargs syntax, pass as many options as you need

New Filesystem API

java.nio.file, java.nio.file.util

- Access to files, directories and filesystems
  - Including links
- Metadata (size, owner, create time, etc)
  - Metadata about filesystems too
- Map paths to FileRef instances
  - FileRef provides information about a file or directory
- Watch service
  - Be notified of changes to files or the filesystem
  - Create/modify/delete of files and directories
- SPI to plug in your own filesystem types

Filesystem Class

java.nio.file

- Abstract view of a filesystem
- Factory for Path and WatchService objects
- Obtain a FileSystem object from the FileSystems class
  - FileSystems.getDefault() in most cases
  - Factory methods to obtain FileSystem instances from specific providers, or to open a file as a filesystem (CDROM image, say)
- May be a façade in front of several distinct filesystems
  - Each FileRef has an associated BackingFileSystem
  - BackingFileSystem encapsulates the actual supported capabilities

Filesystem Providers

java.nio.file.spi

- You can write your own FileSystemProvider
  - Provider registration method may depend on JSR-277
- Provider is a factory for FileSystem, FileRef and FileChannel objects
- Need not be tied to a “real” filesystem
  - Zip file, CD-ROM image, ram disk, flash rom, etc
- Multiple/alternate views of same underlying files
  - Hide sensitive files, read-only views, path munging, etc

Files

java.nio.file

- The old java.io.File class is a mess
  - Makes assumptions, poor metadata access, no links, etc
- FileRef (interface)
  - A reference to a specific file or directory
  - Factory for ByteChannels to read/write file content
- Path (implements FileRef)
  - Locates a file by its path
  - Path-related methods to rename, copy, add links, get parent, etc
  - Returned by FileSystem and DirectoryEntry objects
    - Also static helper methods: Path.get(“foo.txt”)

File Operations

java.nio.file

```
Path source = Path.get("foo");
Path target = Path.get("/otherdir/bar");

source.createFile();                                  // create if doesn't exist

source.copyTo (target, REPLACE_EXISTING);             // copy source to target

source.moveTo (target, ATOMIC_MOVE);                  // rename/move atomically

source.createLinkTo (target);                         // link: source -> target

URI uri = source.toURI();                             // URI form of path
```

Directories

java.nio.file

Path globbing, regex patterns also supported

- Collection of DirectoryEntry objects

```
Path qpath = Path.get ("/path/to/queuedir");
DirectoryStream stream = qpath.newDirectoryStream (“*.qmsg”);

try {
    for (DirectoryEntry entry: stream) {
        processQueuedFile (entry);       // DirectoryEntry is a FileRef
        entry.delete();
    }
} finally {
    stream.close();
}


// Closure-friendly alternative syntax
Files.withDirectory (qpath, new DirectoryAction() {
  public void invoke (DirectoryEntry entry) {
      ...
  }
});
```

Files Helper Class

java.nio.file.util

- java.nio.file.util.Files
  - Not to be confused with java.io.File
- Two handy utility methods
  - Probe a file to discover its mime type
    - Uses FileTypeDetector (java.nio.file.spi)
    - You can create and install your own
  - Walk a file tree using the Visitor pattern
    - You provide an instance of the FileVisitor interface
    - Pass it to Files.walkFileTree() with a FileRef object
    - Your visitor is called for each directory and file in the tree
    - Your visitor may terminate or prune the traversal

File Tree Walk

```
import static java.nio.file.util.FileVisitResult.*;

class FileTypeMapper extends AbstractFileVisitor {
   private final Path base;
   private final Map<Path,String> map;

   public FileTypeMapper (Path base, Map<Path,String> map) { ... }

   public FileVisitResult visitFile (FileRef file, FileName[] dirs,
      BasicFileAttributes attrs)
   {
      map.put (fullPath (base, dirs), Files.probeContentType (file));
      return CONTINUE;
   }

   public FileVisitResult preVisitDirectory (...
   // skip some directories by returning PRUNE rather than CONTINUE
}
...

Map<Path,String> mimeMap = ...;    // path => mimetype mappings
Path path = Path.get ("/path/to/some/dir");
Files.walkFileTree (path, true, new FileTypeMapper (path, mimeMap));
```

File Attributes

java.nio.file.attribute

- Generalized metadata API
- Specialized views of file and filesystem attributes
  - May be mutable or read-only
- BasicFileAttributes
  - PosixFileAttributes, DosFileAttributes
- AttributeView
  - FileAttributeView
    - BasicFileAttributeView
      - PosixFileAttributeView, DosFileAttributeView
    - ExtendedAttributeView, AclFileAttributeView
  - FileSystemAttributeView
    - DiskSpaceAttributeView

Attributes and Attribute Views

```
// alternative: helper method to get attrs directly
PosixFileAttributes attrs =
   Attributes.readPosixFileAttribtues (file, true);
```

```
FileRef file = ...

PosixFileAttributeView view =
   file.newFileAttributeView (PosixFileAttributeView.class);

if (view == null) {
        throw ...      // file doesn't support POSIX view
}

PosixFileAttributes attrs = view.readAttributes();

System.out.format("%s %s %s %d\n",
   PosixFilePermission.toString (attrs.getPermissions()),
   attrs.getOwner().getName(), attrs.getGroup().getName(),
   attrs.getSize());

// set read-only by owner
try {
   view.updatePermissions (PosixFilePermission.OWNER_READ);
} catch (Exception e) { ... }
```

Notification Watch Service

java.nio.file

- FileSystem is the factory for WatchService objects
  - FileSystem.newWatchService()
- Register Watchable objects with a WatchService
  - FileSystem, FileRef and Path
  - Can register for create, modify, and/or delete
- A WatchKey is returned, similar to SelectionKey
  - But WatchService and Selector do not interoperate
- Instances of WatchEvent queue up on the key
- No guarantee that events will be delivered reliably
  - Depends on Filesystem implementation

Watchable Objects

java.nio.file

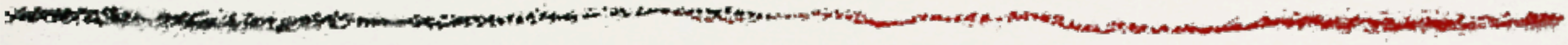
- FileRef and Path are Watchable
  - Watch for changes to files
  - Watch for files added to or deleted from directories

- FileSystem is Watchable
  - Mount and unmount
  - Media insert and eject
  - With your own custom FileSystem, you could create custom events such as low disk space, queue full, etc

WatchService Queue Processor

java.nio.file

```
Path qdir = Path.get ("/path/to/queuedir");
WatchService watcher = qpath.getFileSystem().newWatchService();

qdir.register (watcher, StandardWatchEventType.ENTRY_CREATE);

while (true) {
   WatchKey key = watcher.take();          // sleeps if no current event

   for (WatchEvent event: key.pollEvents()) {
      FileName name = (FileName)event.getContext();  // name of new file
      Path file = qdir.getChild (name);

      processQueuedFile (file);
      file.remove();
}
```



Asynch I/O

Asynchronous I/O

java.nio.channels

- Modeled on java.util.concurrent package
  - May use “real” asynch I/O where available
  - Doesn’t necessarily dispatch to a different thread
- Four new asynchronous channel types
  - AsynchronousFileChannel, ...SocketChannel, etc
- Methods that could block (read/write/connect/etc) instead immediately return an IoFuture object
  - Extends java.util.concurrent.Future
  - Check status of cancel a pending operation

Asynchronous Channels

java.nio.channels

- They do not extend standard Channel classes
  - May not be registered with a Selector
- Four kinds
  - AsynchronousFileChannel
  - AsynchronousSocketChannel
  - AsynchronousServerSocketChannel
  - AsynchronousDatagramChannel
- Each type has open() factory methods
- Instances belong to an AsynchronousChannelGroup
  - Contains an Executor that manages threads
  - There is a default, or you can provide your own

IoFuture

java.nio.channels

- Extends from java.util.concurrent.Future
- Poll status or wait for completion
- Async I/O methods accept an optional CompletionHandler
  - Its completed() method is called, with the IoFuture as the argument, when the operation is complete
  - The handler **may** run in a different thread
  - Practice good concurrency hygiene
- Async I/O methods accept an Object attachment
  - Attach your own context
  - Retrieve it from the IoFuture

Poll Slow Device

java.nio.channels

```
AsynchronousDatagramChannel slowDevice = ...
byte [] msgBuf = new byte [DEVICE_MSG_SIZE];
ByteBuffer buffer = ByteBuffer.wrap (msgBuf);
IoFuture<Integer,Byte[]> result =
   slowDevice.read (buffer, msgBuffer, null);  // buf, attach, handler

while (true) {
   doSomething();
   if (isTimeToQuit()) break;

   if (ioResult.isDone()) {
      try {
         int n = result.getNow();              // get result of the read

         processDeviceMessage (result.attachment(), n);  // msgBuffer
         result = slowDevice.read (buffer, msgBuf, null);// poll again
      } catch (ExecutionException e) {
         // thrown by getNow() if the I/O operation threw an exception
      }
   }
}
// cancel operation if still pending, forcing if necessary
result.cancel (true);  slowDevice.close();
```

Fill Buffer, Process, Repeat

java.nio.channels

```
AsynchronousSocketChannel input = ...
AsynchronousSocketChannel output = ...
ByteBuffer buffer = ByteBuffer.allocate (MESSAGE_SIZE);

input.read (buffer, null, new CompletionHandler<Integer,Void> {
   public void completed (IoFuture<Integer,Void> result) {
      int n = result.getNow();                    // get result of the read
      if (n < 0) { channel.close(); ...; }  // end-of-stream

      if (buffer.hasRemaining()) {
         input.read (buffer, null, this);    // read until full
      } else {
         buffer.flip();                           // buffer is full
         processData (buffer, output);        // process buffer, writes
         buffer.clear();
         input.read (buffer, null, this);    // starts the next cycle
   }
});
// the above will run asynchronously, indefinitely

doSomethingElse();
...
```

Async Behind The Scenes

java.nio.channels

- Solaris
  - Uses the event port framework
- Windows
  - Uses overlapped I/O
- Linux
  - Uses epoll
- Others
  - Platform-specific services as appropriate
  - May simulate with pooled threads

Summary

- Buffers, Sockets and File I/O
  - Mostly small additions for completeness
  - Multicast is the big new feature
- New Filesystem API
  - Tons of new ways to deal with files, directories, filesystems and metadata
  - Change notification service
- Asynchronous I/O
  - Initiate and respond to I/O operations concurrently
  - Builds on the patterns of java.util.concurrent
  - Timely addition for an increasingly multi-core world

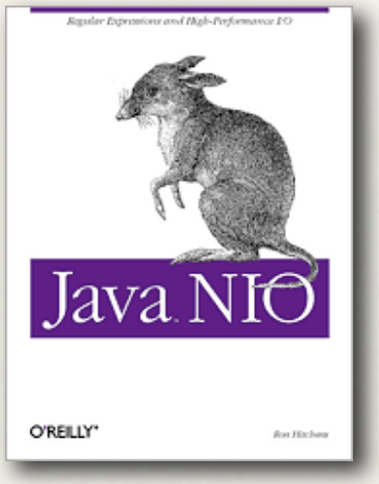
So Where Is NIO.2 Already?

- The work is done, it's ready to go as part of JDK 1.7
- In the final stages of being open-sourced
  - Keep an eye on http://openjdk.java.net
  - Should be available for download within a few weeks
- The NIO.2 team wants to build a community
  - Join the effort, you can still make a difference

For More Information

Web Sites

- http://openjdk.java.net
- http://javanio.info



Books

- *Java NIO*, Ron Hitchens (O'Reilly)
  - (Sorry, no NIO.2 info in there, but still worth buying)
- *Java Concurrency In Practice,* Brian Goetz, et al (AW)

Special Thanks

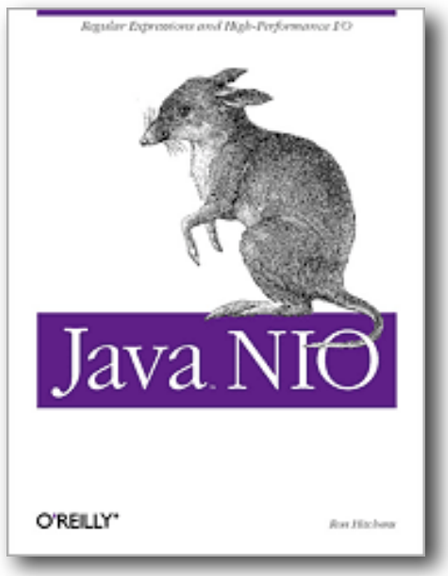
- Alan Bateman (Sun), Spec Lead for NIO.2 (JSR-203)

Q&A

Ron Hitchens
ron@ronsoft.com
http://javanio.info/

What's New in NIO.2

Ron Hitchens
Senior Engineer
Mark Logic Corporation
San Mateo, CA
ron.hitchens@marklogic.com
ron@ronsoft.com



SD WEST 2008
MARCH 3-7, 2008, SANTA CLARA, CA