Java 7
Asynchronous I/O (NIO2)

Julien Buret
Séven Le Mesle

Asynchronous I/O API

Comparaison avec les autres API I/O

- Socket
  - Java 1.0
  - **Synchrone**
  - **Bloquant**
- SocketChannel
  - Java 1.4
  - **Synchone**
  - **Non bloquant**
  - Reactor pattern
  - Notification
- AsyncSocketChannel
  - Java 1.7
  - **Asynchrone**
  - **Non bloquant**
  - Proactor pattern

Asynchronous I/O API

Comparaison avec les autres API I/O

FileOutputStream FileInputStream	FileChannel	AsynchronousFileChannel
OutputStream InputStream	SocketChannel	AsynchronousSocketChannel
OutputStream InputStream	ServerSocketChannel	AsynchronousServerSocketChannel
Java 1.0	Java 1.4	Java 1.7

Asynchronous I/O API

Créer un channel

```
AsynchronousChannelGroup channelGroup = AsynchronousChannelGroup.withFixedThreadPool(4,
threadFactory);
AsynchronousServerSocketChannel socketChannel = AsynchronousServerSocketChannel.open
(channelGroup);
socketChannel.bind(new InetSocketAddress(8080));
Future<AsynchronousSocketChannel> resAccept = socketChannel.accept();
Future<Integer> res = resAccept.get().read(ByteBuffer.allocate(512));
```

Asynchronous I/O API
Concept : Future

- Future<Integer> result = channel.read(byteBuffer);
  - Retourne un «Future» avec le nombre d'octet lue (ou écrit)
- result.get();
  - Attend la fin de l'opération I/O
- result.get(5, TimeUnit.SECONDS);
  - Attend avec un timout
- result.isDone();
  - Verifie si l'appel est terminé

Asynchronous I/O API

Concept : CompletionHandler

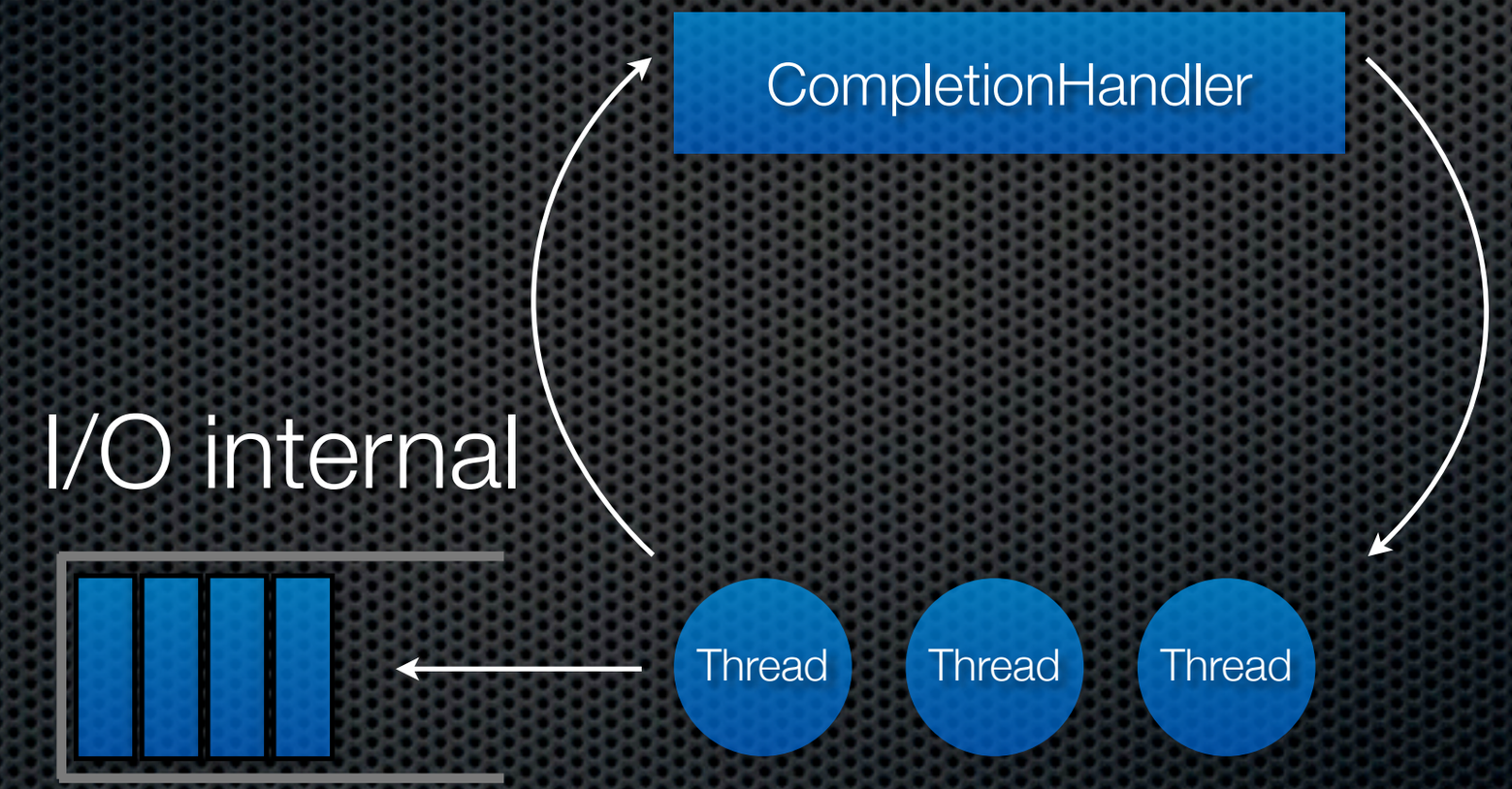
```
channelToClient.read(buffer, counter, new CompletionHandler<Integer,
AtomicInteger>() {

    public void completed(final Integer result, final AtomicInteger counter){
        handlerThreadPool.submit(new Runnable() {
            public void run() {
                readComplete(result, counter, buffer, channelToClient);
            }
        });
    }

    public void failed(Throwable exc, AtomicInteger counter) {
        logger.error("Client {} failed to read", counter, exc);
    }
});
```

Threading model

- AsynchronousChannelGroup
  - Encapsule le pool de thread, la file d'attente et les autres ressources partagées par les sockets
  - 2 implémentations
    - Fixed thread pool
    - Cached thread pool (ou thread pool fournit)

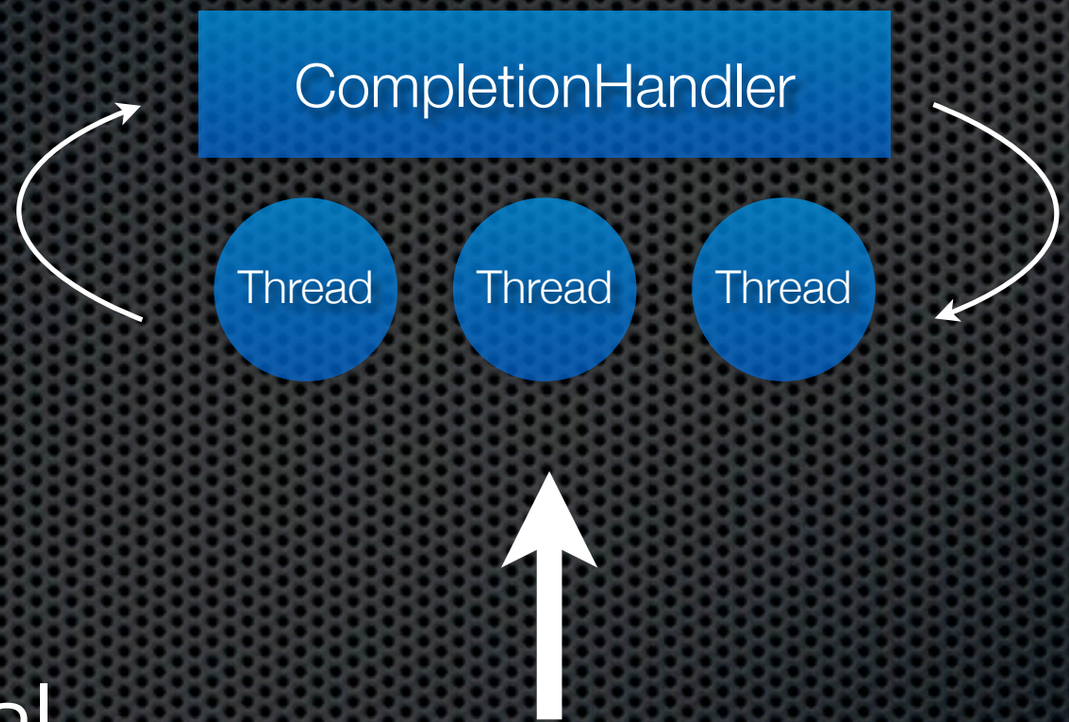
Threading model

Fixed thread pool



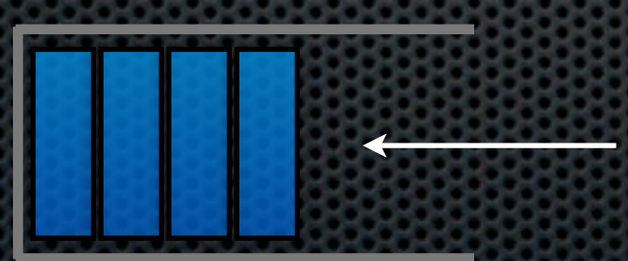
Threading model

Cached thread pool



Cached Thread pool

I/O internal



Internal Thread pool

1 ou sun.nio.ch.internalThreadPoolSize

Threading model

- FixedThreadPool
  - Thread partagé entre I/O et callback
  - Eviter la contention dans les callback
- CachedThreadPool ou ThreadPool fournit
  - Un pool de thread pour l'I/O et un autre pour les callback
  - Attention au context switch

Bench

- Sur une VM 2vCPU (client et serveur)
- 4000 req/s (50Ko/req et 50Ko/resp)
- 200 Mo/s (Reçu et émis)

Multicast non bloquant

```
NetworkInterface eth0Itf = NetworkInterface.getByName("eth0");
        InetAddress mcGroup = InetAddress.getByName("225.0.0.100");

    DatagramChannel dc = DatagramChannel.open(StandardProtocolFamily.INET)
            .setOption(StandardSocketOptions.SO_REUSEADDR, true)
            .bind(new InetSocketAddress(5000))
            .setOption(StandardSocketOptions.IP_MULTICAST_IF, eth0Itf);

    MembershipKey key = dc.join(mcGroup, eth0Itf);

    Selector selector = Selector.open();
    dc.register(selector, dc.validOps());
```

Multicast non bloquant

```
while (true) {
       selector.select();
      Iterator it = selector.selectedKeys().iterator();

      while (it.hasNext()) {
          SelectionKey selKey = (SelectionKey) it.next();
          it.remove();
          if (selKey.isValid() && selKey.isReadable()) {
              DatagramChannel sChannel = (DatagramChannel) selKey.channel();
          }
          if (selKey.isValid() && selKey.isWritable()) {
              DatagramChannel sChannel = (DatagramChannel) selKey.channel();
          }
          }
}
```

Le reste

- Mise à jour des API SocketChannel
  - bind(SocketAdress local)
  - setOption(SocketOption name, T Value)
- Support de SCTP (Protocole de transport réseau)
- IPV6 sous Windows
- Support de SDP sous Solaris (protocole RMDA utilisé par Infiniband)

File extension

- Traitement Asynchrone
- Droits POSIX / ACL NFS / DOS / ...
- Liens symboliques et vérous
- Système de fichier et partitions
- Supervision d’activité sur répertoires et fichiers
- Parcours de répertoire
- Méthodes utilitaires

java.nio.Path

- = java.io.File
- file.toPath() - path.toFile()
- chemin système du fichier

```
Path p = Paths.get(“/home/user/.myApp“);

Iterator<Path> it = p.iterator(); // elements

p.relativize(Paths.get(“/home/john“)); // “../john“

p.normalize(); // /home/john/../user = /home/user

p.startsWith(“/home“);
```

AsynchronousFileChannel

- Traitement asynchrone non bloquant
  - Chaque read/write donne la position dans le fichier
  - Accède a plusieurs parties du fichier en parallèle
- Threadsafe
- Supporte les Locks
- Options définies sur open(...)

AsynchronousFileChannel

```
AsynchronousFileChannel afc = AsynchronousFileChannel.open(path,
StandardOpenOption.CREATE);

FileLock  lock = afc.lock().get();

afc.write(buffer, 0l).get();
 // Buffer wrote to file

lock.close();
afc.close();
```

Files operations

- Files fournit des utilitaires (guava / commons-io)
  - copy / move / delete /...
  - read* / write / newReader* / newWriter*
  - temp dirs / symlinks / walkFileTree
- Et les accès systèmes
  - set or get : owner / attribute / posixFilePermissions
  - FileSystem / FileStore

FileAttributes

- Lecture et modification des atributs du fichier
  - BasicFileAttributes, PosixFileAttributes, Dos, ...

```
Path p = Paths.get(“/home/user/.myApp“);

BasicFileAttributes attr = Files.readAttributes(file,BasicFileAttributes.class);

System.out.println("creationTime: " + attr.creationTime());
System.out.println("lastAccessTime: " + attr.lastAccessTime());
System.out.println("lastModifiedTime: " + attr.lastModifiedTime());

System.out.println("isDirectory: " + attr.isDirectory());
System.out.println("isSymbolicLink: " + attr.isSymbolicLink());
System.out.println("size: " + attr.size());

Set<PosixFilePermission> perms = PosixFilePermissions.fromString("rw-------");
FileAttribute<Set<PosixFilePermission>> attr = PosixFilePermissions.asFileAttribute(perms);
Files.setPosixFilePermissions(file, perms);
```

WatchService

- Comme un Selector pour les Paths
  - WatchKey k = register(path, event, ...);
    - WatchEvent evt = k.pollEvents();
    - Reset key
- Boucle infini et Threading à la charge du développeur

WatchService

```
Path p = Paths.get("/home/user/.myApp");

WatchService watcher = FileSystems.getDefault().newWatchService();
WatchKey key = p.register(watcher, ENTRY_CREATE, ENTRY_DELETE, ENTRY_MODIFY);

for (;;) {
    key = watcher.take();
    for (WatchEvent<?> event: key.pollEvents()) {
        WatchEvent.Kind<?> kind = event.kind();

        WatchEvent<Path> ev = (WatchEvent<Path>)event;
        Path filename = ev.context();
        //....
    }
    boolean valid = key.reset();
    if (!valid) {
        break;
    }
}
```

DirectoryStream

- Autorise le forEach() sur les entrées d'un répertoire
  - Glob pattern ("*.java")
  - RegExp
  - Filtre implémenté

```
DirectoryStream<Path> ds = Files.newDirectoryStream(dir, "*.java");

for(Path p : ds){
   // ...
}
```

FileVisitor

- Files.walkFileTree parcours l'arborescence
  - utilise un FileVisitor
    - Définit des options (FOLLOW_LINKS, ...)
- FileVisitor est invoqué pour chaque entrée
  - Permet de modifier le parcours
  - Fourni des callbacks pre/post visit File / Directory

FileVisitor

```
Path start = ...
Files.walkFileTree(start, new SimpleFileVisitor<Path>() {
    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException
    {
        Files.delete(file);
        return FileVisitResult.CONTINUE;
    }
    @Override
    public FileVisitResult postVisitDirectory(Path dir, IOException e) throws IOException
    {
        if (e == null) {
            Files.delete(dir);
            return FileVisitResult.CONTINUE;
        } else {
            // directory iteration failed
            throw e;
        }
    }
});
```