Apache Mina

The high-performance protocol construction toolkit.

Peter Royal <proyal@apache.org>

Originally presented at ApacheCon Europe 2007 in Amsterdam
Hi, I’m Peter

http://fotap.org/~osi
MINA hacker since
Fall 2005
San Francisco
Apache Member

http://apache.org
What is MINA?
MINA?
Multipurpose Infrastructure Networked Applications

http://mina.apache.org
Built on Java NIO
Non-Blocking
Asynchronous
Event-Driven
(SEDA!)
Multiple Transports

(framework is really agnostic)
UDP

(being re-written for 2.0)
In-VM

(great for testing)
RS-232
(under development)
Smartly Designed
Follows Inversion of Control Pattern

(plays nicely with PicoContainer, Spring, etc)
Separation of Concerns
Rather than this...
Concerns are Separated

Wire Protocol

Application Logic

Concerns are Separated
Stable and Production-Ready

- **v1.0** released Fall 2006
- **v1.1** released April 2007
  - Same API as v1.0 but uses Java 5 Concurrency primitives
- **v2.0** this year
  - API simplification based on lessons learned
Many Users
Apache Directory

http://directory.apache.org

LDAPv3, NTP, DNS, DHCP and Kerberos
AsyncWeb

http://asyncweb.safehaus.org
(joining MINA @ Apache very soon though!)

HTTP/HTTPS
Apache (incubating) Qpid

http://cwiki.apache.org/qpid/

Advanced Messaging Queuing Protocol (AMQP)

(from Wall Street!)
QuickFIX/J

http://www.quickfixj.org/

Financial Information eXchange (FIX)
Openfire

http://www.jivesoftware.com/products/openfire/

XMPP
...and more!

(maybe you, next time!)
Key Concepts
ByteBuffer

- Core NIO construct
- MINA version that wraps and provides **additional convenience methods**
  - auto-expanding, string encoding
- MINA gives control...
  - allocate from the **Heap** or **Stack**
  - optional **Pooling**
    - (in v2, will be non-pooled and heap-only, as it provides the best performance)
Future

- Represents a function call that completes asynchronously
- Provides blocking functions to retrieve the result
- MINA allows callbacks to be invoked upon completion, so invoking thread can “fire and forget”
  - (unlike the Java 5 Future)
Two Versions
IoAcceptor
“act as server”
single thread for new connections
IoConnector

“act as client”
Connection instance is an IoSession
Handles reads and writes
Instance count scales with CPU/Load
Session fixed to an Instance

(under review for v2)
Chain of IoFilter's
Per Connection
Reusable
Hot Deployable
Filter all events

Read / Write / Idle / etc
Application Logic Lives Here
Large Library of IoFilter’s
Protocol Conversion

- Framework to plug in your own codecs to handle conversion to/from a ByteBuffer
- Existing codecs
  - Text-based
  - Java Serialization
Logging
(great for debugging!)
SSL / TLS
Compression
Read Throttling

(we’re working on a general solution for writes)
Thread Models
(a necessary evil)
“single threaded”
One IoProcessor Thread
Scalability sucks
Add more IoProcessor Threads
(at least one per CPU core)
Lowest latency
Scales nicely

(connection latency to be addressed in v2)
“multi threaded”
use ExecutorFilter
IoProcessor threads only do reads & writes

(the intent)
Filters execute on a different thread.

(filter location is key!)
Work for a session is serialized

(queued per session)
Work is delegated to an Executor

(generally a java.util.concurrent.ThreadPoolExecutor)
Size thread pool to “active” session count

(too much in queue == OOM!
Use the Read Throttle filter)
“recommended application pattern”

- Use **ExecutorFilter** as first in chain
  - Unless you need really low latency
- Use **ProtocolCodecFilter**
  - Convert the wire protocol into a Java representation
- Put application logic into an **IoHandler**
- Store state in the **IoSession**
- Minimum of **Java 5**
  - java.util.concurrent rocks!
Example Time!
Everybody loves a Haiku
public class HaikuValidator {
    private static final int[] SYLLABLE_COUNTS = { 5, 7, 5 };

    public void validate(Haiku haiku) throws InvalidHaikuException {
        String[] phrases = haiku.getPhrases();

        for (int i = 0; i < phrases.length; i++) {
            String phrase = phrases[i];
            int count = PhraseUtilities.countSyllablesInPhrase(phrase);

            if (count != SYLLABLE_COUNTS[i]) {
                throw new InvalidHaikuException(i + 1, phrase, count, SYLLABLE_COUNTS[i]);
            }
        }
    }
}
Simple Protocol

- Connect
- Send 3 lines of text
- Receive **HAIKU!** or **NOT A HAIKU:**
  - (plus a little reason why not)
ProtocolCodecFilter + TextLineCodecFactory

Bytes to Java String's. For free!
public class ToHaikuIoFilter extends IoFilterAdapter {

    @SuppressWarnings( { "unchecked" } )
    @Override
    public void messageReceived( NextFilter nextFilter, IoSession session, Object message )
        throws Exception
    {
        List<String> phrases = (List<String>) session.getAttribute( "phrases" );

        if ( null == phrases ) {
            phrases = new ArrayList<String>();
            session.setAttribute( "phrases", phrases );
        }

        phrases.add( (String) message );

        if ( phrases.size() == 3 ) {
            session.removeAttribute( "phrases" );

            super.messageReceived( nextFilter, session, new Haiku( phrases.toArray( new String[3] ) ) );
        }
    }
}
Filter is very testable
(mock objects rock!)
IoHandler is very simple

Validate Haiku, send result
Also very testable
Very easy to hook it all up
Questions?
Thank You!