Asynchronous & Event-Driven Programming

For The Enterprise Developer

bob brown, transentia pty. ltd.
www.transentia.com.au

Danger, Will Robinson!

My sensors detect a large techno-nerd-fest coming this way…*lots* of code…

Buying A Book

Synchronous
You go to your local book store, wait impatiently in a queue while the cashier gets her makeup ready, then pay for the book and take it home.

Asynchronous
You order the book on Amazon, and then get on with other things in your life. At some indeterminate point in the future, when the book is ready for you, the postie raises a knocking event on your door so that the book can be delivered to you.

History

- original idea of J2EE as hosting appliance
- “thou shalt not thread”

```
public class BadBean implements javax.ejb.SessionBean {

   public void ejbCreate() {}
   public void ejbActivate() {}
   public void ejbPassivate() {}
   public void setSessionContext(javax.ejb.SessionContext ctx) {}
   public void unsetSessionContext() {}
   public void ejbRemove() {}

   public String doImportantStuff() {
      Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
      boolean flag = true;
      while (flag = true) {
        flag = /* do something nice */
        }
   }
}
```

- An enterprise bean must not use thread synchronization primitives to synchronize execution of multiple instances.

- The enterprise bean must not attempt to manage threads. The enterprise bean must not attempt to start, stop, suspend, or resume a thread, or to change a thread’s priority or name. The enterprise bean must not attempt to manage thread groups.

- of course, everyone just *had* to find a reason to use threading…
- led to all sorts of *horrid* ‘architectures’
  - I still have nightmares…

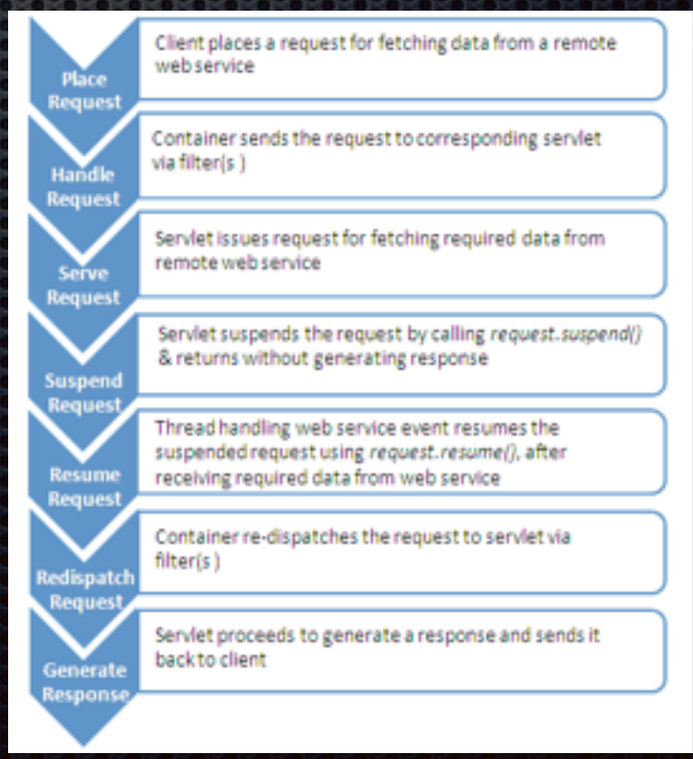
Old-Skool Threading

- thread-per-connection
  - very early* model for handling web traffic
  - scalable strictly within limitation of a thread pool
- thread-per-request
  - thread is allocated only when it is detected that web I/O is in progress
  - given pool of threads can serve much greater amount of traffic
  - but a thread is still allocated for the duration of the request handling
    - thread throttled by having to wait for other resources

* obsolete since before the turn of the century…

Kool Threading

- servlet allocated to a thread to handle an incoming request
- if the servlet becomes blocked on a resource, it is suspended; no response is generated and the thread is freed
- when a resource becomes ready, the/another thread is picked to resume the servlet which can finally create the response



https://today.java.net/pub/a/today/2008/10/14/introduction-to-servlet-3.html

The Good, The Bad And The Ugly

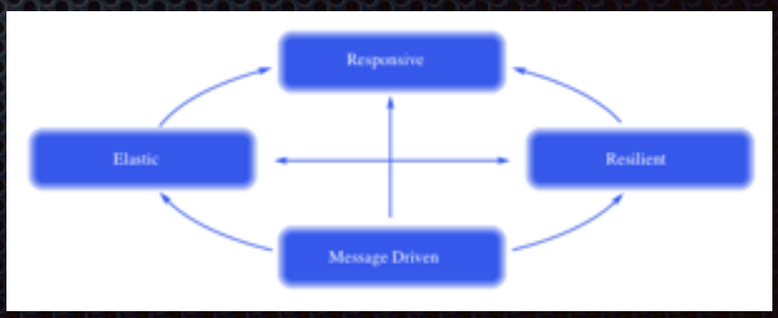
- good
  - *potentially* greater degree of parallelism/improved resource usage for a given system: “more bang for the buck”
  - decoupled dependencies can lead to more tolerant systems
  - better UIs possible via server ‘push’ eventing
- bad
  - flavour of the month; subject to the “i’ve got a hammer” effect
  - can be more tricky to be asynchronous; contention, synchronisation become ‘things’ to think about
    - humans are *really bad* at this
  - frequently need to reinvent the synchronous wheel
  - expectation setting: not *faster*, rather *more performant*
- ugly
  - “callback hell”
  - getting *something* to work is easy, getting a truly *robust* solution is not and its not just a matter of (insert favourite programming language here)’s syntax

> “…tasking seems to have been neglected in most languages currently in production use for such systems. … Semaphores, events, signals and other similar mechanisms are clearly at too low a level. Monitors, on the other hand, are not always easy to understand and… seem to offer an unfortunate mix of high-level and low-level concepts. It is believed that Ada strikes a good balance…”
>
> — Rationale for Ada ‘83 concurrency

Reactive Manifesto 2.0

- Today's demands are simply not met by yesterday's software architectures.
- Reactive Systems are:
  - Responsive: The system responds in a timely manner if at all possible.
  - Resilient: The system stays responsive in the face of failure.
  - Elastic: The system stays responsive under varying workload.
  - Message Driven: Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling, isolation, location transparency, and provides the means to delegate errors as messages.

http://www.reactivemanifesto.org/



Java 8 Improvements

- parallel streams
- lambdas
- try-with-resources

```
try (Writer w = new StringWriter()) {
    w.write("Hello, world!");
}
catch (IOException ioe) {
    ioe.printStackTrace(System.err);
}
```

```
package x;

import java.util.*;
import java.util.stream.*;

public class Main {
    public static void main(String [] args) {
        assert new ArrayList<String>() {{ add("88"); }}.equals(extractParams("fred", "x=1&fred=88&y=42"));
    }

    private static List<String> extractParams(String key, String s) {
        // ("" + s): protect against s being null
        return Stream.of(("" + s).split("&"))
                .parallel()
                .map(line -> line.split("="))
                .filter(pair -> pair.length == 2)
                .filter(entry -> entry[0].equalsIgnoreCase(key))
                .map(entry -> entry[1])
                .collect(Collectors.toList());
    }
}
```

Futures And Callbacks

```
String s = doSomethingTedious();
```

```
private Future<String> doSomethingTedious() { return new Future<String>(); }

private void f() {
   Future<String> f = doSomethingTedious();
   while(!f.isDone()) { doSomethingFun(); }
   String result = f.get();
}
```

We know it most endearingly as "callback hell" or the "pyramid of doom":

```
doAsync1(function () {
  doAsync2(function () {
    doAsync3(function () {
      doAsync4(function () {
    })
  })
})
```

Callback hell is subjective, as heavily nested code can be perfectly fine sometimes. Asynchronous code is *hellish* when it becomes overly complex to manage the flow. A good question to see how much "hell" you are in is: *how much refactoring pain would I endure if* ***doAsync2*** *happened before* ***doAsync1****?* The goal isn't about removing levels of indentation but rather writing modular (and testable!) code that is easy to reason about and resilient.

```
var fs = require('fs')
var path = require('path')

module.exports = function (dir, cb) {
  fs.readdir(dir, function (er, files) { // [1]
    if (er) return cb(er)
    var counter = files.length
    var errored = false
    var stats = []

    files.forEach(function (file, index) {
      fs.stat(path.join(dir,file), function (er, stat) { // [2]
        if (errored) return
        if (er) {
          errored = true
          return cb(er)
        }
        stats[index] = stat // [3]

        if (--counter == 0) { // [4]
          var largest = stats
            .filter(function (stat) { return stat.isFile() }) // [5]
            .reduce(function (prev, next) { // [6]
              if (prev.size > next.size) return prev
              return next
            })
          cb(null, files[stats.indexOf(largest)]) // [7]
        }
      })
    })
  })
}
```

http://strongloop.com/strongblog/node-js-callback-hell-promises-generators/

Concurrency Utilities

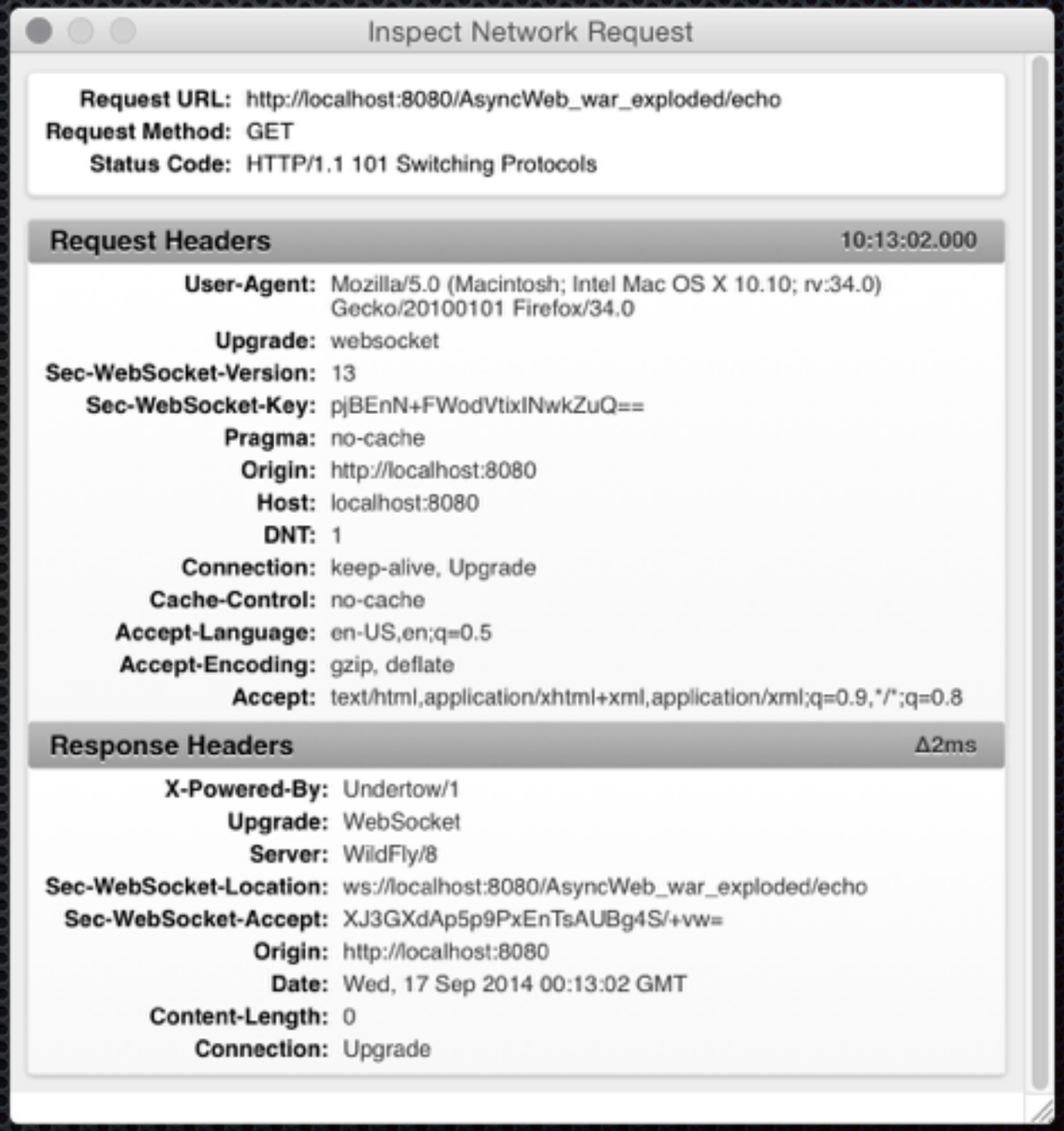
- J2EE: quartz/flux (or proprietary solutions) ruled
- JEE7: Concurrency Utils
  - server managed resources; accessed via JNDI/injection
  - ManagedExecutor
    - create threads that are managed by the container; can access all JEE facilites, subject to DI, etc.
  - ManagedScheduledExecutor
    - delayed/periodic tasks
  - ManagedThreadFactory
    - low-level facility

```
<subsystem xmlns="urn:jboss:domain:ee:2.0">
    <spec-descriptor-property-replacement>false</spec-descriptor-property-replacement>
    <concurrent>
        <context-services>
            <context-service name="default" jndi-name="java:jboss/ee/concurrency/context/default"
                             use-transaction-setup-provider="true"/>
        </context-services>
        <managed-thread-factories>
            <managed-thread-factory name="default" jndi-name="java:jboss/ee/concurrency/factory/default"
                                    context-service="default"/>
        </managed-thread-factories>
        <managed-executor-services>
            <managed-executor-service name="default" jndi-name="java:jboss/ee/concurrency/executor/default"
                                      context-service="default" hung-task-threshold="60000" core-threads="5"
                                      max-threads="25" keepalive-time="5000"/>
        </managed-executor-services>
        <managed-scheduled-executor-services>
            <managed-scheduled-executor-service name="default"
                                                jndi-name="java:jboss/ee/concurrency/scheduler/default"
                                                context-service="default" hung-task-threshold="60000"
                                                core-threads="2" keepalive-time="3000"/>
        </managed-scheduled-executor-services>
    </concurrent>
    <default-bindings context-service="java:jboss/ee/concurrency/context/default"
                      datasource="java:jboss/datasources/ExampleDS"
                      jms-connection-factory="java:jboss/DefaultJMSConnectionFactory"
                      managed-executor-service="java:jboss/ee/concurrency/executor/default"
                      managed-scheduled-executor-service="java:jboss/ee/concurrency/scheduler/default"
                      managed-thread-factory="java:jboss/ee/concurrency/factory/default"/>
</subsystem>
```

```
$ wildfly/bin/jboss-cli.sh --connect
[standalone@localhost:9990 /] /subsystem=ee/managed-thread-factory=myManagedThreadFactory:add(priority=1,context-service=default,jndi-name=java:jboss/ee/concurrency/factory/MyManagedThreadFactory)
[standalone@localhost:9990 /] /subsystem=ee/managed-thread-factory=myManagedThreadFactory:write-attribute(name=priority,value=3)
[standalone@localhost:9990 /] /:reload
```

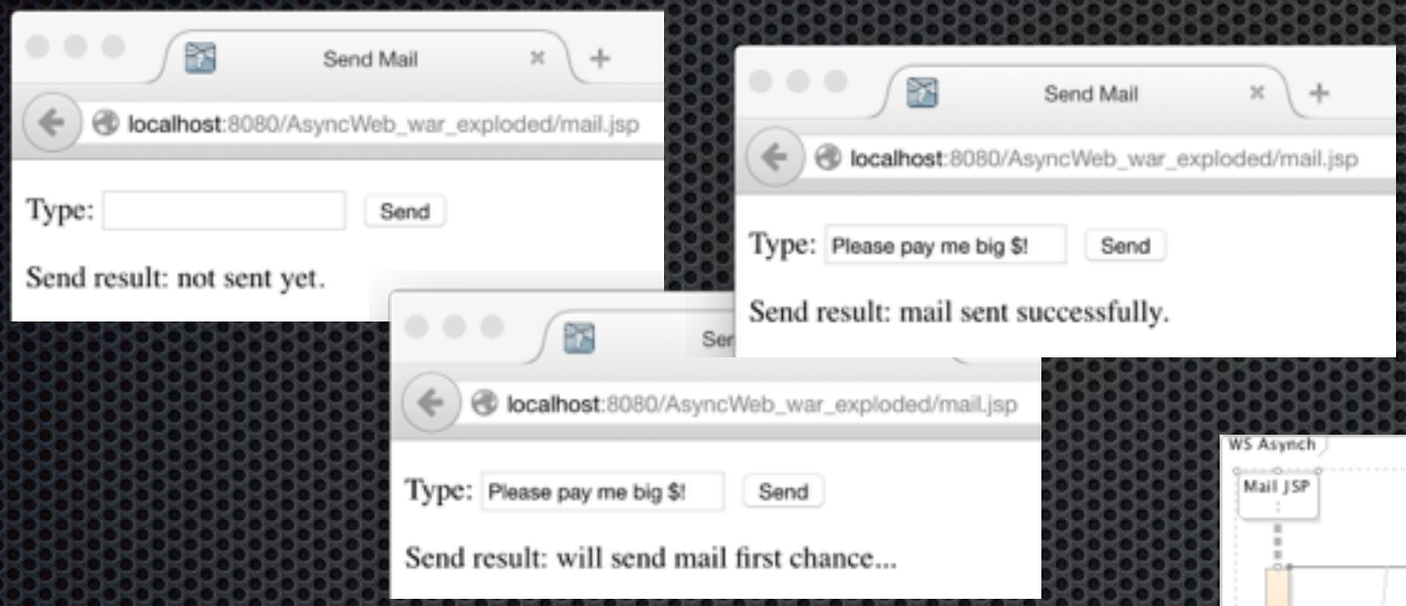
WebSockets

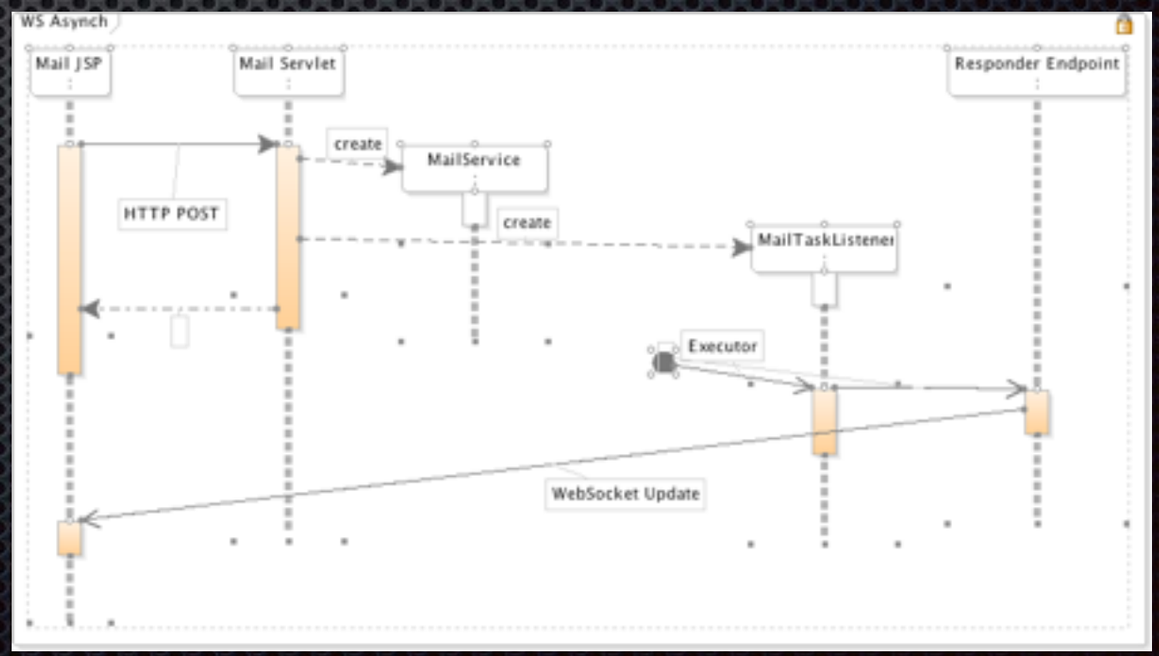
- undirectional channel allows the server to asynchronously push data to clients
- can associate codecs for custom datatypes
- Javascript componentry
- part of HTML5, so supported by most modern browsers



Example

- pushing mail status notifications to connected clients





Example...

```
<%@ page contentType="text/html;charset=UTF-8" language="java" import="java.util.UUID" %>
<!DOCTYPE html>
<html>
<head>
  <title>Send Mail</title>
  <script src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
  <%! String clientID = UUID.randomUUID().toString(); %>
</head>
<body>
  <p>Type: <input type="text" name="body" id="body"> 
  <button type="button" id="send">Send</button></p>
  <p>Send result: <span id="result">not sent yet.</span></p>
</body>
<script>
  $(function () {
    new WebSocket("ws://" + location.host + "${pageContext.request.contextPath}/responder/<%= clientID %>").onmessage = function (evt) {
      $("#result").text("mail " + (evt.data == "true" ? "sent" : "NOT sent") + " successfully.");
    };
    $("#send").click(function() {
      $.post("mail", {clientID: "<%= clientID %>", body: $("#body").val()}, function (data) {
        $("#result").html(data);
      })
    });
  });
</script>
</html>
```

Example...

```
package bob;

import javax.websocket.*;
import javax.websocket.server.*;
import java.util.concurrent.ConcurrentHashMap;

@ServerEndpoint(value = "/responder/{clientID}")
public class Responder {
    private static final ConcurrentHashMap<String, Session> sessions = new ConcurrentHashMap<>();

    @OnOpen
    public void onOpen(@PathParam("clientID") String clientID, Session session) {
        sessions.put(clientID, session);
    }

    @OnClose
    public void onClose(@PathParam("clientID") String clientID, Session session) {
        sessions.remove(clientID);
    }

    public static void send(String clientID, String message) {
        final Session clientSession = sessions.get(clientID);
        if (clientSession != null)
            try {
                clientSession.getBasicRemote().sendText(message);
            } catch (Exception e) {
                e.printStackTrace();
            }
    }
}
```

Example...

```
package bob;

import javax.annotation.Resource;
import javax.enterprise.concurrent.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.*;
import java.util.concurrent.*;

@WebServlet("/mail")
public class Mail extends HttpServlet {
    @Resource
    private ManagedExecutorService executorService;

    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        String body = req.getParameter("body");
        String clientID = req.getParameter("clientID");
        Callable<Boolean> mt = ManagedExecutors.managedTask(new MailService(body), new MailTaskListener(clientID));
        @SuppressWarnings("unused") Future<Boolean> unused = executorService.submit(mt);
        resp.setStatus(HttpServletResponse.SC_OK);
        try (Writer w = resp.getWriter()) {
            w.write("will send mail first chance...");
        }
    }
}
```

Example...

```
package bob;

import javax.enterprise.concurrent.*;
import java.util.concurrent.*;

public class MailTaskListener implements ManagedTaskListener {
    private final String clientID;

    public MailTaskListener(final String clientID) { this.clientID = clientID; }

    @Override
    public void taskDone(Future<?> future, ManagedExecutorService managedExecutorService, Object mailService, Throwable throwable) {
        try {
            Responder.send(clientID, "" + future.get());
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void taskAborted(Future<?> future, ManagedExecutorService managedExecutorService, Object o, Throwable throwable) { }

    @Override
    public void taskStarting(Future<?> future, ManagedExecutorService managedExecutorService, Object o) { }

    @Override
    public void taskSubmitted(Future<?> future, ManagedExecutorService managedExecutorService, Object o) { }
}
```

Example...

```
package bob;

import javax.annotation.Resource;
import javax.mail.*;
import javax.mail.internet.*;
import java.util.concurrent.Callable;

public class MailService implements Callable<Boolean> {
    private final String body;

    @Resource
    private Session session;

    MailService(String body) { this.body = body; }

    @Override
    public Boolean call() {
        Boolean result = Boolean.TRUE;
        try {
            Message message = new MimeMessage(session);
            message.setRecipients(Message.RecipientType.TO, InternetAddress.parse("bob@transentia.com.au"));
            message.setSubject("Mail from JEE7");
            message.setText(body);

            Transport.send(message);
        } catch (MessagingException e) {
            e.printStackTrace();
            result = Boolean.FALSE;
        }
        return result;
    }
}
```

Servlet 3.0+

```
@WebServlet(asyncSupported=true ...)
@WebFilter(asyncSupported=true ...)
```

- async
  - startAsync() returns an AsyncContext object that caches the request/response on a queue, and the original request thread is recycled
  - a ServletContextListener is notified whenever waited-for resources become available
  - after a request is processed call complete() to commit
- both servlets and filters
  - can be tricky to make async filters, request dispatching and non-blocking I/O work together

Example

```
package bob;

import javax.servlet.*
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.*;

@WebServlet(urlPatterns = { "/simple" }, asyncSupported = true)
public class SimpleAsync extends HttpServlet {

    @Override
    protected void doGet(final HttpServletRequest request, final HttpServletResponse response) throws ServletException, IOException {
        System.out.println("doGet started");
        final AsyncContext context = AsyncHelper.getAsyncContext(request, response);
        context.start(() -> {
            doSomething(context);
            context.complete();
        });
        System.out.println("doGet finished");
    }

    private void doSomething(AsyncContext context) {
        System.out.println("doSomething started");

        HttpServletResponse response = (HttpServletResponse) context.getResponse();
        try (Writer w = response.getWriter()) {
            // Thread.currentThread().sleep(5000);
            response.setContentType("text/plain");
            w.write("Welcome to the world of tomorrow!");
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("doSomething finished");
    }
}
```

```
2014-09-26 17:52:38,648 INFO  [stdout] (default task-13) doGet started
2014-09-26 17:52:38,649 INFO  [stdout] (default task-13) doGet finished
2014-09-26 17:52:38,649 INFO  [stdout] (default task-14) doSomething started
2014-09-26 17:52:43,653 INFO  [stdout] (default task-14) doSomething finished
```

Example...

```
package bob;

import javax.servlet.*;
import java.io.IOException;

public class AsyncHelper {
    public static AsyncContext getAsyncContext(ServletRequest request, ServletResponse response) {
        return getAsyncContext(request, response, new NullAsyncListener());
    }

    public static AsyncContext getAsyncContext(ServletRequest request, ServletResponse response, AsyncListener asyncListener) {
        final AsyncContext asyncContext;
        if (request.isAsyncStarted()) {
            asyncContext = request.getAsyncContext();
        } else {
            asyncContext = request.startAsync(request, response);
            asyncContext.addListener(asyncListener, request, response);
            asyncContext.setTimeout(120000);
        }
        return asyncContext;
    }
}

class NullAsyncListener implements  AsyncListener {
    @Override public void onComplete(AsyncEvent arg0) throws IOException { }
    @Override public void onError(AsyncEvent arg0) throws IOException { }
    @Override public void onStartAsync(AsyncEvent arg0) throws IOException { }
    @Override public void onTimeout(AsyncEvent arg0) throws IOException { }
}
```

Servlet 3.0+

- non-blocking I/O
  - can check read/write availability in streams before actually doing it
  - SocketStreams augmented with {Read,Write} Listener

Example

- Trivial rate-limited servlet and listener

```
package bob;

import javax.annotation.Resource;
import javax.enterprise.concurrent.ManagedScheduledExecutorService;
import javax.servlet.*
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.*;

@WebServlet(name = "DataRateLimitedServlet", urlPatterns = {"/rls"},  asyncSupported = true)
public class DataRateLimitedServlet extends HttpServlet {

   @Resource
   private ManagedScheduledExecutorService managedScheduledExecutorService;

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
      response.setContentType("text/plain");
      ServletOutputStream out = response.getOutputStream();
      out.setWriteListener(new TrivialRateLimitedDataStream(managedScheduledExecutorService,
                     new FileInputStream("/etc/passwd"), AsyncHelper.getAsyncContext(request, response), out));
   }
}
```

Example...

```
package bob;

import javax.enterprise.concurrent.ManagedScheduledExecutorService;
import javax.servlet.AsyncContext;
import javax.servlet.*;
import java.io.*;
import java.util.concurrent.*;

public class TrivialRateLimitedDataStream implements WriteListener {
    private final int BUFFERSIZE = 2048;
    private final long TIMEOUT = 100L;
    private final InputStream inp;
    private final AsyncContext async;
    private final ServletOutputStream out;
    private final ScheduledFuture<?> scheduledFuture;

    public TrivialRateLimitedDataStream(ManagedScheduledExecutorService managedScheduledExecutorService,
                                        InputStream inp, AsyncContext async, ServletOutputStream out) {
        this.inp = inp;
        this.async = async;
        this.out = out;
        scheduledFuture = managedScheduledExecutorService.scheduleAtFixedRate(() -> {
            try {
                onWritePossible();
            } catch (Exception e) {
                onError(e);
            }
        }, TIMEOUT, TIMEOUT, TimeUnit.MILLISECONDS);
    }
```

```
    @Override
    public void onWritePossible() throws IOException {
        if (out.isReady()) {
            byte[] buffer = new byte[BUFFERSIZE];
            int len = inp.read(buffer);
            if (len < 0) {
                async.complete();
                scheduledFuture.cancel(false);
                return;
            }
            out.write(buffer, 0, len);
        }
    }

    @Override
    public void onError(Throwable t) {
        async.complete();
        scheduledFuture.cancel(true);
    }
}
```

Example...

```
package bob;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;

@WebFilter(servletNames = {"DataRateLimitedServlet"}, asyncSupported = true)
public class TimerFilter implements Filter {
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        AsyncHelper.getAsyncContext(request, response, new AsyncListener() {
            private long start = System.currentTimeMillis();

            @Override public void onComplete(AsyncEvent arg0) throws IOException {
                System.out.println("TimerAsyncListener: " + (System.currentTimeMillis() - start) + "ms");
            }

            @Override public void onStartAsync(AsyncEvent arg0) throws IOException {
                start = System.currentTimeMillis();
            }

            @Override public void onError(AsyncEvent arg0) throws IOException { }
            @Override public void onTimeout(AsyncEvent arg0) throws IOException { }
        });

        chain.doFilter(request, response);
    }

    public void init(FilterConfig config) throws ServletException { }
    public void destroy() { }
}
```

EJBs

- @Asynchronous annotation
  - Session beans
  - control immediately returned to the caller from the EJB container

Example

```
package ejb;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.Writer;
import java.math.BigDecimal;

@WebServlet("/fantasy")
public class Servlet extends HttpServlet {
    @EJB
    FantasyBean fantasyBean;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        try {
            fantasyBean.giveBobABigRaise(BigDecimal.TEN);
            try (Writer w = response.getWriter()) {
                w.write("Ask and ye shall receive...ho ho ho.");
            }
        } catch (UnthinkableException e) {
            response.sendError(HttpServletResponse.SC_BAD_REQUEST, e.getMessage());
        }
    }
}
```

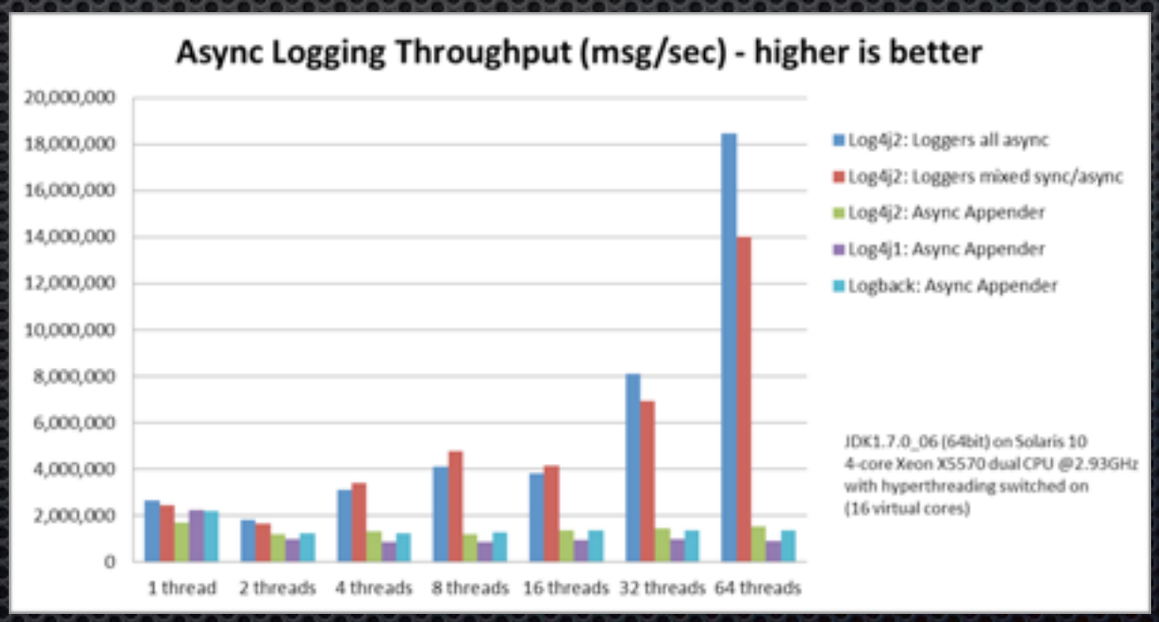
```
package ejb;

import javax.ejb.EJB;
import javax.ejb.Stateless;
import java.math.BigDecimal;

@Stateless
public class FantasyBean {
    @EJB
    LoggerBean loggerBean;

    public Boolean giveBobABigRaise(BigDecimal howManyMillions) throws UnthinkableException {
        String msg = String.format("Bob asked for %.2f million this time. Mad fool!", howManyMillions);
        loggerBean.log(msg);
        throw new UnthinkableException(msg);
    }
}
```

```
package ejb;

import javax.ejb.Asynchronous;
import javax.ejb.Stateless;

@Stateless
public class LoggerBean {
    @Asynchronous
    public void log(String msg) {
        System.out.println(msg);
    }
}
```

```
13:46:32,764 INFO  [stdout] (default task-9) doGet: Giving Bob a raise(?)
13:46:32,765 INFO  [stdout] (default task-9) FantasyBean: Bob asked for 10.00 million this time. Mad fool!
13:46:32,766 INFO  [stdout] (EJB default - 4) Bob asked for 10.00 million this time. Mad fool!
13:46:33,271 INFO  [stdout] (default task-9) doGet: Giving Bob a raise(?)...I'm guessing no...
```

(By The Way)

- Log4J 2 has AsyncLoggers
  - "…Performance close to insane…more than 18,000,000 messages per second, while others do around 1,500,000 or less in the same environment."



http://www.javacodegeeks.com/2013/07/log4j-2-performance-close-to-insane.html

CDI Events

- events allow beans to communicate without any compile-time dependency
- strangely neglected in JEE ’til recently
  - OOP ≈≈ properties, events, methods
- some advantages:
  - looser coupling
  - easier testing
  - clearer application structure

http://wordpress.transentia.com.au/wordpress/2014/05/30/using-groovys-configslurper-to-support-external-configurations-with-jee7s-cdi/

Example

```
package bob;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;

@WebServlet("/servlet")
public class Servlet extends HttpServlet {

    @EJB
    private CatMaker catMaker;

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        catMaker.make();
    }
}
```

```
package bob;

import javax.enterprise.inject.spi.CDI;
import javax.persistence.*;
import java.time.LocalDateTime;

@Entity
@EntityListeners(CatEventListener.class)
@Access(AccessType.FIELD)
public class Cat {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    Long id;

    @Basic
    String name;

    @Basic
    Short age;
}

class CatEventListener {
    @PostPersist
    public void postPersist(Object object) {
        Cat cat = (Cat) object;

        CDI.current().getBeanManager().fireEvent(new AuditEvent("postPersist", object));
    }
}

class AuditEvent {
    public final String what;
    public final LocalDateTime when = LocalDateTime.now();
    public final Object who;

    public AuditEvent(final String what, final Object who) {
        this.what = what;
        this.who = who;
    }
}
```

Example...

```
package bob;

import javax.ejb.Stateless;
import javax.persistence.*;
import javax.transaction.Transactional;

@Stateless
public class CatMaker {
   @PersistenceContext(unitName = "CatsPersistenceUnit")
   private EntityManager em;

   @Transactional
   public void make() {
      Cat cat = new Cat();
      cat.age = 5;
      cat.name = "Scotty";
      em.persist(cat);
     em.flush();  // needed for CDI
   }
}
```

```
package bob;

import javax.ejb.*
import javax.enterprise.event.*
import javax.transaction.Transactional;
import java.time.LocalDateTime;

@Stateless
@Transactional(Transactional.TxType.NOT_SUPPORTED)
public class AuditEventTracker {
   public void auditAfterFailure(@Observes(during=TransactionPhase.AFTER_FAILURE) AuditEvent ae) {
      System.out.printf("AuditEventTracker (%s AFTER_FAILURE) for %s at %s\n", ae.what, ae.who, ae.when.toString());
   }

   public void auditAfterSuccess(@Observes(during=TransactionPhase.AFTER_SUCCESS) AuditEvent ae) {
      System.out.printf("AuditEventTracker (%s AFTER_SUCCESS) for %s at %s\n", ae.what, ae.who, ae.when.toString());
   }

   @Schedule(minute = "*/1", hour = "*", persistent = false)
   public void heartbeat(@SuppressWarnings("unused") Timer timer) {
      System.out.printf("AuditEventTracker (heartbeat) at %s\n", LocalDateTime.now().toString());
   }
}
```

Netty

http://netty.io/

- "…an asynchronous event-driven network application framework for rapid development of maintainable high performance protocol servers & clients."
  - used everywhere
  - natively NIO.2
  - powerful; build everything; use plain UDP and up
  - low-level, so poor ergonomics
    - "provides practically no support for structuring an *application* beyond the networking protocol concerns."
      http://www.ratpack.io/manual/current/intro.html#netty

Example

```
package io.netty.example.qotm;

import io.netty.bootstrap.Bootstrap;
import io.netty.channel.*;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.nio.NioDatagramChannel;

public final class QuoteOfTheMomentServer {
    private static final int PORT = Integer.parseInt(System.getProperty("port", "7686"));

    public static void main(String[] args) throws Exception {
        EventLoopGroup group = new NioEventLoopGroup();
        try {
            Bootstrap b = new Bootstrap();
            b.group(group)
                    .channel(NioDatagramChannel.class)
                    .option(ChannelOption.SO_BROADCAST, true)
                    .handler(new QuoteOfTheMomentServerHandler());
            b.bind(PORT).sync().channel().closeFuture().await();
        } finally {
            group.shutdownGracefully();
        }
    }
}
```

```
package io.netty.example.qotm;

import io.netty.buffer.Unpooled;
import io.netty.channel.*;
import io.netty.channel.socket.DatagramPacket;
import io.netty.util.CharsetUtil;
import java.util.concurrent.ThreadLocalRandom;

public class QuoteOfTheMomentServerHandler extends SimpleChannelInboundHandler<DatagramPacket> {
    private static final String[] quotes = {
            "Where there is love there is life.",
            "First they ignore you, then they laugh at you, then they fight you, then you win.",
            "Be the change you want to see in the world.",
            "The weak can never forgive. Forgiveness is the attribute of the strong.",
    };

    private static String nextQuote() {
        return quotes[ThreadLocalRandom.current().nextInt(quotes.length)];
    }

    @Override
    public void messageReceived(ChannelHandlerContext ctx, DatagramPacket packet) {
        if ("QOTM?".equals(packet.content().toString(CharsetUtil.UTF_8))) {
            ctx.write(new DatagramPacket(
                    Unpooled.copiedBuffer("QOTM: " + nextQuote(), CharsetUtil.UTF_8), packet.sender()));
        }
    }

    @Override
    public void channelReadComplete(ChannelHandlerContext ctx) {
        ctx.flush();
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) {
        cause.printStackTrace();
// We don't close the channel because we can keep serving requests.
    }
}
```

https://github.com/netty/netty/tree/master/example/src/main/java/io/netty/example/qotm

Example...

```
package io.netty.example.qotm;

import io.netty.channel.*;
import io.netty.channel.socket.DatagramPacket;
import io.netty.util.CharsetUtil;

public class QuoteOfTheMomentClientHandler extends SimpleChannelInboundHandler<DatagramPacket> {
    @Override
    public void messageReceived(ChannelHandlerContext ctx, DatagramPacket msg) {
        String response = msg.content().toString(CharsetUtil.UTF_8);
        if (response.startsWith("QOTM: ")) {
            System.out.println("Quote of the Moment: " + response.substring(6));
            ctx.close();
        }
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) {
        cause.printStackTrace();
        ctx.close();
    }
}
```

```
package io.netty.example.qotm;

import io.netty.bootstrap.Bootstrap;
import io.netty.buffer.Unpooled;
import io.netty.channel.*;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.*;
import io.netty.util.CharsetUtil;
import java.net.InetSocketAddress;

public final class QuoteOfTheMomentClient {
    static final int PORT = Integer.parseInt(System.getProperty("port", "7686"));

    public static void main(String[] args) throws Exception {
        EventLoopGroup group = new NioEventLoopGroup();
        try {
            Bootstrap b = new Bootstrap();
            b.group(group)
                    .channel(NioDatagramChannel.class)
                    .option(ChannelOption.SO_BROADCAST, true)
                    .handler(new QuoteOfTheMomentClientHandler());
            Channel ch = b.bind(0).sync().channel();
            ch.writeAndFlush(new DatagramPacket(
                    Unpooled.copiedBuffer("QOTM?", CharsetUtil.UTF_8),
                    new InetSocketAddress("255.255.255.255", PORT))).sync();
            if (!ch.closeFuture().await(5000)) {
                System.err.println("QOTM request timed out.");
            }
        } finally {
            group.shutdownGracefully();
        }
    }
}
```

Grails

- “…an Open Source, full stack, web application framework for the JVM. It takes advantage of the Groovy programming language and convention over configuration to provide a productive and stream-lined development experience.”

Grails...

- asynchronous domain handling

```
import static grails.async.Promises.*

Cat.async.list().onComplete { List results ->
  log.debug "Got cat = ${results}"
}
def p = Cat.async.getAll(1L, 2L, 3L)
List results = p.get()

def p1 = Cat.async.findByName('Scotty')
def p2 = Cat.async.findByFirstName('Furball')
def p3 = Cat.async.findByFirstName('Kramer')
results = waitAll(p1, p2, p3)
```

- asynchronous controllers

```
import static grails.async.Promises.*
...
def index() {
   tasks cats: Cat.async.list(),
         totalCats: Cat.async.count()
}

def index2() {
   render view:"myView", model: tasks(one: { somethingBig() },
                                      two: { longRunningThing() })
   ...
```

```
import static grails.async.Promises.*
...
def index3(String ticker) {
   task {
      ticker = ticker ?: 'GOOG'
      def url = new URL("http://download.finance.yahoo.com/d/quotes.csv?s=${ticker}&f=nsl1op&e=.csv")
      Double price = url.text.split(',')[-1] as Double
      render "ticker: $ticker, price: $price"
      }
   }
}
```

Vert.x

http://vertx.io/

- "…a lightweight, high performance application platform for the JVM that's designed for modern mobile, web, and enterprise applications."
  - asynchronous with a distributed event bus
    - wraps netty
  - mix-and-match/polyglot version of node.js
    - Java, Groovy, JavaScript, CoffeeScript, Ruby, Python, Scala, Ceylon, Kotlin, …

Example

- red-light traffic camera image upload server

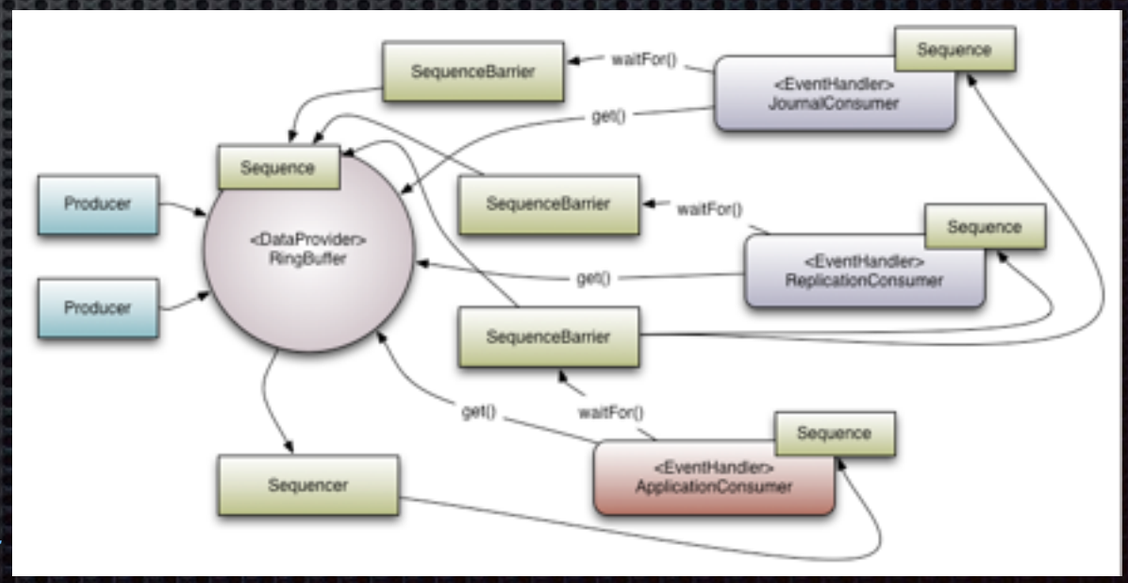
```
package helloworld

vertx.createHttpServer().requestHandler { req ->
  def tmpFilename

  req.expectMultiPart = true

  req.endHandler {
    final finalPath = "uploads/${req.formAttributes.camera}/${req.formAttributes.timestamp}"
    vertx.fileSystem.mkdir(finalPath, "rwxr-xr-x", true) { mkdir ->
      vertx.fileSystem.moveSync(tmpFilename, "${finalPath}/${tmpFilename}")
    }
  }

  req.uploadHandler { upload ->
    upload.streamToFileSystem tmpFilename = upload.filename
  }

  req.response.end()

}.listen(8080, "localhost")
```

```
import groovyx.net.http.HTTPBuilder
import org.apache.http.entity.ContentType
import org.apache.http.entity.mime.HttpMultipartMode
import org.apache.http.entity.mime.MultipartEntityBuilder

import static groovyx.net.http.ContentType.JSON
import static groovyx.net.http.Method.POST

new HTTPBuilder('http://localhost:8080/').request(POST, JSON) { req ->
  requestContentType = 'multipart/form-data'

  def builder = MultipartEntityBuilder.create()
  builder.with { b ->
    b.setMode(HttpMultipartMode.BROWSER_COMPATIBLE);

    def gif = new File("assets/clear.gif")
    b.addBinaryBody("image", gif, ContentType.create("image/gif"), gif.name)
    b.addTextBody("timestamp", "${System.currentTimeMillis()}")
    b.addTextBody("camera", "RLTC-88")
  }
  req.entity = builder.build()
  response.success = { resp ->
    assert resp.statusLine.statusCode == 200
  }
}
```

Reactor

https://github.com/reactor/reactor

- “Reactor is a foundational library building for reactive fast data applications on the JVM. It provides abstractions for Java, Groovy and other JVM languages to make building event and data-driven applications easier. It’s also really fast. …it’s possible to process over 15,000,000 events per second with the RingBufferDispatcher and over 25,000,000 events per second in a single thread. Other dispatchers are available… from thread-pool style, long-running task execution to non-blocking, high-volume task dispatching.”
- ambitious: intended for more than just webby stuff; BIG focus on performance
  - “It can’t just be Big Data, it has to be Fast Data”
- being integrated into Grails
- lots of functional goodness



LMAX Disruptor

Reactor...

- excerpts

```
import static reactor.event.selector.Selectors.*;

// Only create one of these per JVM
static Environment env = new Environment();

// Create a Reactor and listen to a topic using a Selector
Reactor r = Reactors.reactor(env)
  .<String>on($("topic"), ev -> System.out.println("greeting: " + ev.getData()));

r.notify("topic", Event.wrap("Hello World!"));
```

https://github.com/reactor/reactor-quickstart

```
public static void main(String[] args) throws InterruptedException {
    Environment env = new Environment();
    final TradeServer server = new TradeServer();
    final CountDownLatch latch = new CountDownLatch(totalTrades);

    // Rather than handling Trades as events, each Trade is accessible via Stream.
    Deferred<Trade, Stream<Trade>> trades = Streams.defer(env);

    // We compose an action to turn a Trade into an Order by calling server.execute(Trade).
    Stream<Order> orders = trades.compose()
            .map(server::execute)
            .consume(o -> latch.countDown());

    // Start a throughput timer.
    startTimer();

    // Publish one event per trade.
    for (int i = 0; i < totalTrades; i++) {
        // Pull next randomly-generated Trade from server into the Composable,
        Trade trade = server.nextTrade();
        // Notify the Composable this Trade is ready to be executed
        trades.accept(trade);
    }

    // Wait for all trades to pass through
    latch.await(30, TimeUnit.SECONDS);

    // Stop throughput timer and output metrics.
    endTimer();

    server.stop();
}
```

https://spring.io/blog/2013/11/12/it-can-t-just-be-big-data-it-has-to-be-fast-data-reactor-1-0-goes-ga

Ratpack

http://www.ratpack.io/

- “Ratpack is a set of Java libraries that facilitate fast, efficient, evolvable and well tested HTTP applications. It is built on the highly performant and efficient Netty event-driven networking engine. Ratpack focuses on allowing HTTP applications to be efficient, modular, adaptive to new requirements and technologies, and well-tested over time…..Ratpack is for nontrivial, high performance, low resource usage, HTTP applications.”

- java 8, groovy friendly

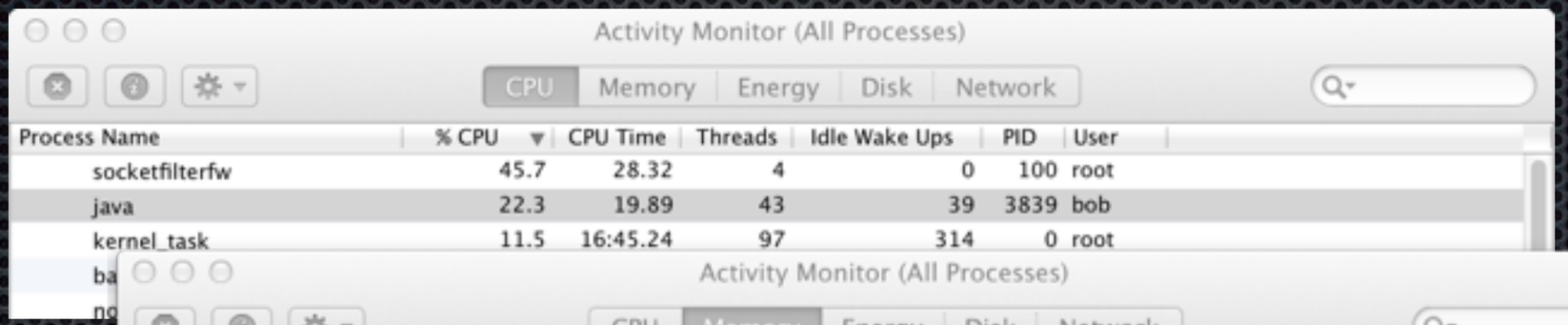
Example

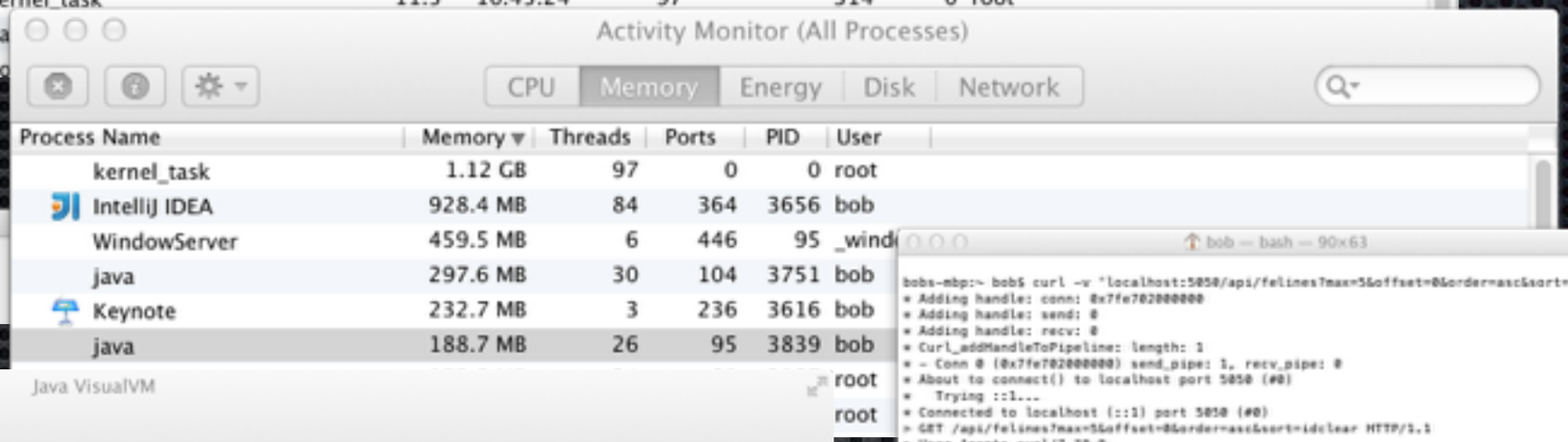
```
ratpack {
  bindings {
    add new JacksonModule(); add new AbstractModule() { ... bind(FelineStore).in(SINGLETON) ... };
    add new CodaHaleMetricsModule().jmx().console()

    init { FelineStore felineStore -> ... }
  }

  handlers { FelineStore datastore ->
    get("api/felines/count") { blocking { datastore.size() }.then { render json(count: it) } }
    handler("api/felines/:id?") {
      def id = pathTokens.id?.safeParseAsLong()
      byMethod {
        get { blocking { id ? datastore.get(id) : datastore.list(request.queryParams) }.then { if (it != null) render json(it) else clientError(404) } }  }
        post { blocking { def f = parse Feline; datastore.add(f) }.then { render json(it) }  }
        delete { blocking { id ? datastore.delete(id) : null }.then { clientError(it ? 204 : 404) }  }
        put { blocking { def f = parse Feline; f.id = id; f.id ? datastore.update(f) : null }.then { clientError(it ? 204 : 404) } }
      }
    }
    get { render groovyTemplate("grid.html", title: "AngularJS + Ng-grid + Bootstrap + Ratpack REST") }

    assets "public"
  }
}
```

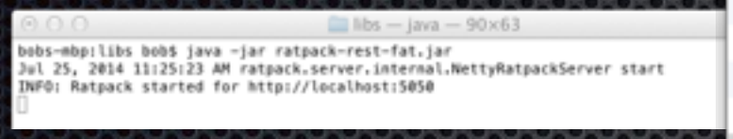
http://wordpress.transentia.com.au/wordpress/2014/07/04/livin-on-the-grid/

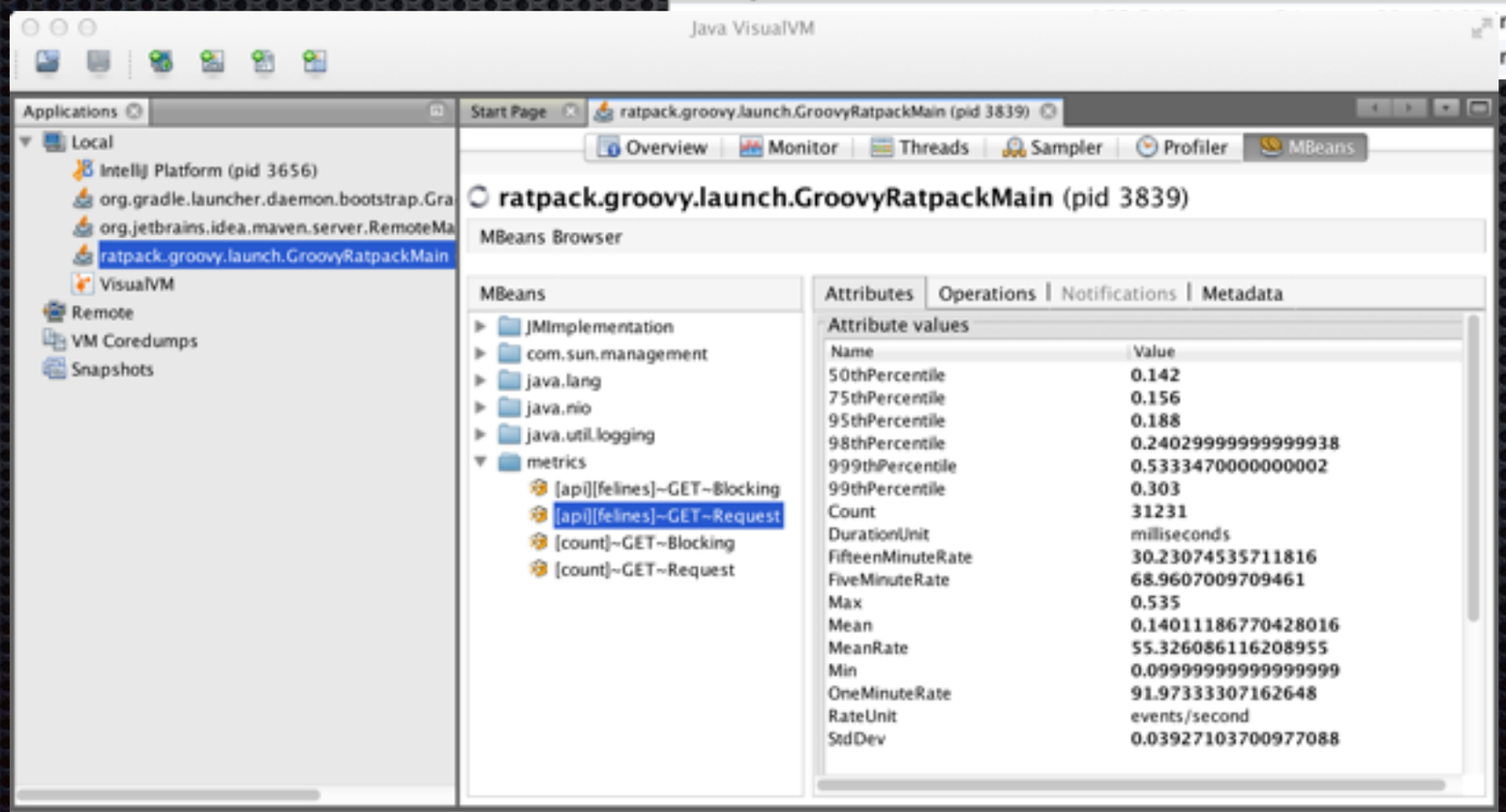
Example...


JAR
Name ratpack-rest-fat.jar
Kind Java JAR file
Size 14.8 MB
Created Today 10:19
Modified Today 10:19
Last opened Today 10:19


Activity Monitor (All Processes)
CPU Memory Energy Disk Network
Process Name % CPU CPU Time Threads Idle Wake Ups PID User
socketfilterfw 45.7 28.32 4 0 100 root
java 22.3 19.89 43 39 3839 bob
kernel_task 11.5 16:45.24 97 314 0 root


Activity Monitor (All Processes)
CPU Memory Energy Disk Network
Process Name Memory Threads Ports PID User
kernel_task 1.12 GB 97 0 0 root
IntelliJ IDEA 928.4 MB 84 364 3656 bob
WindowServer 459.5 MB 6 446 95 _windo
java 297.6 MB 30 104 3751 bob
Keynote 232.7 MB 3 236 3616 bob
java 188.7 MB 26 95 3839 bob
root
root


libs — java — 90×63
bobs-mbp:libs bob$ java -jar ratpack-rest-fat.jar
Jul 25, 2014 11:25:23 AM ratpack.server.internal.NettyRatpackServer start
INFO: Ratpack started for http://localhost:5050


Java VisualVM
Applications
Local
Intellij Platform (pid 3656)
org.gradle.launcher.daemon.bootstrap.Gra
org.jetbrains.idea.maven.server.RemoteMa
ratpack.groovy.launch.GroovyRatpackMain
VisualVM
Remote
VM Coredumps
Snapshots
Start Page
ratpack.groovy.launch.GroovyRatpackMain (pid 3839)
Overview Monitor Threads Sampler Profiler MBeans
ratpack.groovy.launch.GroovyRatpackMain (pid 3839)
MBeans Browser
MBeans
JMImplementation
com.sun.management
java.lang
java.nio
java.util.logging
metrics
[api][felines]~GET~Blocking
[api][felines]~GET~Request
[count]~GET~Blocking
[count]~GET~Request
Attributes Operations Notifications Metadata
Attribute values
Name Value
50thPercentile 0.142
75thPercentile 0.156
95thPercentile 0.188
98thPercentile 0.24029999999999938
999thPercentile 0.5333470000000002
99thPercentile 0.303
Count 31231
DurationUnit milliseconds
FifteenMinuteRate 30.23074535711816
FiveMinuteRate 68.9607009709461
Max 0.535
Mean 0.14011186770428016
MeanRate 55.326086116208955
Min 0.09999999999999999
OneMinuteRate 91.97333307162648
RateUnit events/second
StdDev 0.03927103700977088


bob — bash — 90×63
bobs-mbp:~ bob$ curl -v 'localhost:5050/api/felines?max=5&offset=0&order=asc&sort=id'clear
* Adding handle: conn: 0x7fe702000000
* Adding handle: send: 0
* Adding handle: recv: 0
* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x7fe702000000) send_pipe: 1, recv_pipe: 0
* About to connect() to localhost port 5050 (#0)
* Trying ::1...
* Connected to localhost (::1) port 5050 (#0)
> GET /api/felines?max=5&offset=0&order=asc&sort=idclear HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:5050
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 504
< Connection: keep-alive
<
[{
"id" : 1,
"name" : "Scotty",
"description" : "Active young(ish) male",
"age" : 5,
"deceased" : false
}, {
"id" : 2,
"name" : "Furball",
"description" : "Fluffy!",
"age" : 5,
"deceased" : true
}, {
"id" : 3,
"name" : "Blackie",
"description" : "Black and very affectionate!",
"age" : 6,
"deceased" : false
}, {
"id" : 4,
"name" : "Midnight",
"description" : "Shy male!",
"age" : 4,
"deceased" : false
}, {
"id" : 5,
"name" : "Julius",
"description" : "Can clearly say 'Hello!'",
"age" : 6,
"deceased" : false
* Connection #0 to host localhost left intact
bobs-mbp:~ bob$

Others

- Scala Actors, http://www.scala-lang.org/old/node/242.html
- Netflix's Reactive Extensions for Async Programming, https://github.com/ReactiveX/RxJava/wiki
- Javascript Node.js, http://nodejs.org/
- JavaScript Promises/A+, http://promisesaplus.com/

That’s All, Folks!

- lots of code to digest…