DEVOXX™

Concurrency in Enterprise Java

Alexander Heusingfeld

innoQ

Junior programmers think concurrency is hard.
Experienced programmers think concurrency is easy.
Senior programmers think concurrency is hard.

— Unknown

There ain’t no fuzz about this, right?

```
Runnable myRunnable = new Runnable() {
    public void run() {
        // do stuff in Thread
    }
};
Thread myThread = new Thread(myRunnable);
myThread.start(); // ...and off we go!
```

http://www.flickr.com/photos/girlaphid/3818224024/sizes/l/

1. No management of Thread lifecycle
2. No management of number of Threads
3. Monitoring tedious/ almost impossible
4. No propagation of context

http://www.flickr.com/photos/spanishflea/215064063/sizes/o/

- java.util.concurrent**?!**

ExecutorServices

```
> Executors.newSingleThreadExecutor()

> Executors.newCachedThreadPool()

> Executors.newFixedThreadPool()

> Executors.newScheduledThreadPool()
```

- JSR236
- javax.enterprise.concurrent

Why should I be interested?

Patterns of software stability

Patterns of software stability

- Timeouts - executor timeouts provide fallbacks

Patterns of software stability

- Timeouts - executor timeouts provide fallbacks
- Bulkheads - separate your application components

Patterns of software stability

- Timeouts - executor timeouts provide fallbacks
- Bulkheads - separate your application components
- Circuit Breaker - fail-fast if no threads available

Patterns of software stability

- Timeouts - executor timeouts provide fallbacks
- Bulkheads - separate your application components
- Circuit Breaker - fail-fast if no threads available
- Steady State - free unused resources

Patterns of software stability

- Timeouts - executor timeouts provide fallbacks
- Bulkheads - separate your application components
- Circuit Breaker - fail-fast if no threads available
- Steady State - free unused resources

http://michaelnygard.com/

What’s in it for the devs?

ManagedThreadFactory

ManagedThreadFactory

- extends the Java SE ThreadFactory
- same API as Thread.newThread(Runnable)
- Threads returned implement ManageableThread
- Enables use of Java SE concurrency utilities like Executors

ExecutorServices for Java EE

Executors.newSingleThreadExecutor(ThreadFactory)

Executors.newCachedThreadPool(ThreadFactory)

Executors.newFixedThreadPool(int, ThreadFactory)

Executors.newScheduledThreadPool(int, ThreadFactory)

```
@Resource ManagedThreadFactory factory;

protected void processRequest(ServiceRequestData data) {

    // Creating Java SE style ExecutorService
    ExecutorService executor = Executors.newFixedThreadPool(10, factory);
    Future f = executor.submit(new MyTask(data));
}
```

```
 1 public class MyTask implements Runnable {
 2     public void run() {
 3         InitialContext ctx = null;
 4         try {
 5             ctx = new InitialContext();
 6             UserTransaction ut = (UserTransaction) ctx.lookup(
 7                     "java:comp/UserTransaction");
 8             ut.begin();
 9             while (!((ManageableThread) Thread.currentThread()).isShutdown()){
10 
11                 // Perform transactional business logic?
12             }
13             // do cleanup
14             ut.commit();
15         } catch (Exception ex) {
16             ex.printStackTrace();
17         }
18     }
19 }
```

ManagedExecutorService

ManagedExecutorService

extends the Java SE ExecutorService

methods for submitting tasks to Java EE environment

available via JNDI look-up or @Resource injection

Tasks must implement Runnable or Callable

Lifecycle APIs disabled -> throw IllegalStateException

```
1  @Resource ManagedExecutorService mes;
2
3  protected String processRequest(ServiceRequest request) {
4      Future<String> f = mes.submit(new Callable<String>() {
5          public String call() {
6              // do something with context
7              return "result";
8          }
9      });
10     try {
11         return f.get();
12     } catch (InterruptedException | ExecutionException ex) {
13         throw new IllegalStateException("Cannot get the answer", ex);
14     }
15 }
```

```
1  @Resource ManagedExecutorService mes;
2
3  class MyLongReturningTask implements Callable<Long> { . . . }
4  class MyDateReturningTask implements Callable<Date> { . . . }
5
6  protected void processRequest(ServiceRequest request) throws
7    ExecutionException {
8
9     ArrayList<Callable> tasks = new ArrayList<();
10    tasks.add(new MyLongReturningTask());
11    tasks.add(new MyDateReturningTask());
12    List<Future<Object>> result = executor.invokeAll(tasks);
13 }
```

```
1  @WebServlet(urlPatterns = "/MyAsyncServlet", asyncSupported = true)
2  public class MyAsyncServlet extends HttpServlet {
3
4      @Resource ManagedExecutorService executor;
5
6      protected void doGet(HttpServletRequest request, HttpServletResponse
7                           response) throws ServletException, IOException {
8
9          final AsyncContext ac = request.startAsync();
10         Runnable task = new Runnable() {
11             public void run() {
12                 // ... evaluate result
13                 ac.getResponse().getWriter().print(result);
14                 ac.complete();
15             }
16         }
17         executor.submit(task);
18     }
19 }
```

ManagedScheduledExecutorService

ManagedScheduledExecutorService

- extends ManagedExecutorService
- extends ScheduledExecutorService
- additional methods to schedule tasks with new Trigger
- new methods also return ScheduledFuture objects

```
@Resource ManagedScheduledExecutorService executor;

protected void processRequest(ServiceRequest request) {
    ScheduledFuture<?> f = executor.scheduleAtFixedRate(
        new MyRunnableTask(request), 5, 10, TimeUnit.SECONDS);

}
```

```
 1 public class MyTrigger implements Trigger {
 2
 3     private final Date firetime;
 4
 5     public MyTrigger(Date firetime) { this.firetime = firetime; }
 6
 7     @Override
 8     public Date getNextRunTime(LastExecution le, Date taskScheduledTime) {
 9         if (firetime.before(taskScheduledTime)) {
10             return null;
11         }
12         return firetime;
13     }
14
15     @Override
16     public boolean skipRun(LastExecution le, Date scheduledRunTime) {
17         return firetime.before(scheduledRunTime);
18     }
19 }
```

ContextService

ContextService

- creating contextual proxies
- Examples of context: persistence, EJBs, security
- customisable through ExecutionProperties
- Can be run on transaction context of invoking thread (if any)

```
1 // within servlet or EJB method
2 @Resource ContextService service;
3
4 // Capture application context for later execution
5 IMessageProcessor processor = ...;
6 IMessageProcessor proxy = service.createContextualProxy(processor,
7                           IMessageProcessor.class);
8 cache.put(IMessageProcessor.class, proxy);
```

```
12 // at a later time in a different thread retrieve the proxy from
the cache
13 IMessageProcessor worker = cache.get(IMessageProcessor.class);
14 // and execute with the context of the servlet or EJB
15 worker.processMessage(msg);
```

ManagedTaskListener

```
void taskSubmitted(java.util.concurrent.Future<?> future,
                   ManagedExecutorService executor, Object task)
void taskStarting(java.util.concurrent.Future<?> future,
                  ManagedExecutorService executor, Object task)
void taskAborted(java.util.concurrent.Future<?> future,
                 ManagedExecutorService executor, Object task, java.lang.
                 Throwable exception)
void taskDone(java.util.concurrent.Future<?> future, ManagedExecutorService
              executor, Object task, java.lang.Throwable exception)
```

ManagedExecutors class

```
// Register listener for task with specific executionProperties.

// Adopts executionProperties of ManageableTask

Runnable managedTask(Runnable, ManagedTaskListener)

Runnable managedTask(Runnable, Map<String, String>,
                               ManagedTaskListener)

Callable managedTask(Callable, ManagedTaskListener)

Callable managedTask(Callable, Map<String, String>,
                               ManagedTaskListener)



// Checks “currentThread instanceof ManageableThread”

ManagedExecutors.isCurrentThreadShutdown()
```

DevOps:
How do we do it?

Setup - CLI

```
$JBOSS_HOME/bin/jboss-cli.sh –connect “/subsystem=ee/managed-executor-
service=async-http-mes/:add(
    core-threads=20,
    jndi-name=java:comp/async-http-mes,
    context-service=default,
    hung-task-threshold=110000,
    keepalive-time=60000,
    thread-factory=default,
    queue-length=50,
    reject-policy=ABORT,
    max-threads=500)”
```

WildFly CLI - Monitoring

```
$JBOSS_HOME/bin/jboss-cli.sh --connect "/subsystem=ee/managed-executor-
service=async-http-mes/:read-resource(recursive-depth=0,include-runtime=true)"
{
    "outcome" => "success",
    "result" => {
        "context-service" => "default",
        "core-threads" => 20,
        "hung-task-threshold" => 110000L,
        "jndi-name" => "java:jboss/ee/concurrency/executor/async-http-mes",
        "keepalive-time" => 60000L,
        "long-running-tasks" => false,
        "max-threads" => 500,
        "queue-length" => 50,
        "reject-policy" => "ABORT",
        "thread-factory" => "default"
    }
}
```

Too complicated?

Wildfly CLI command

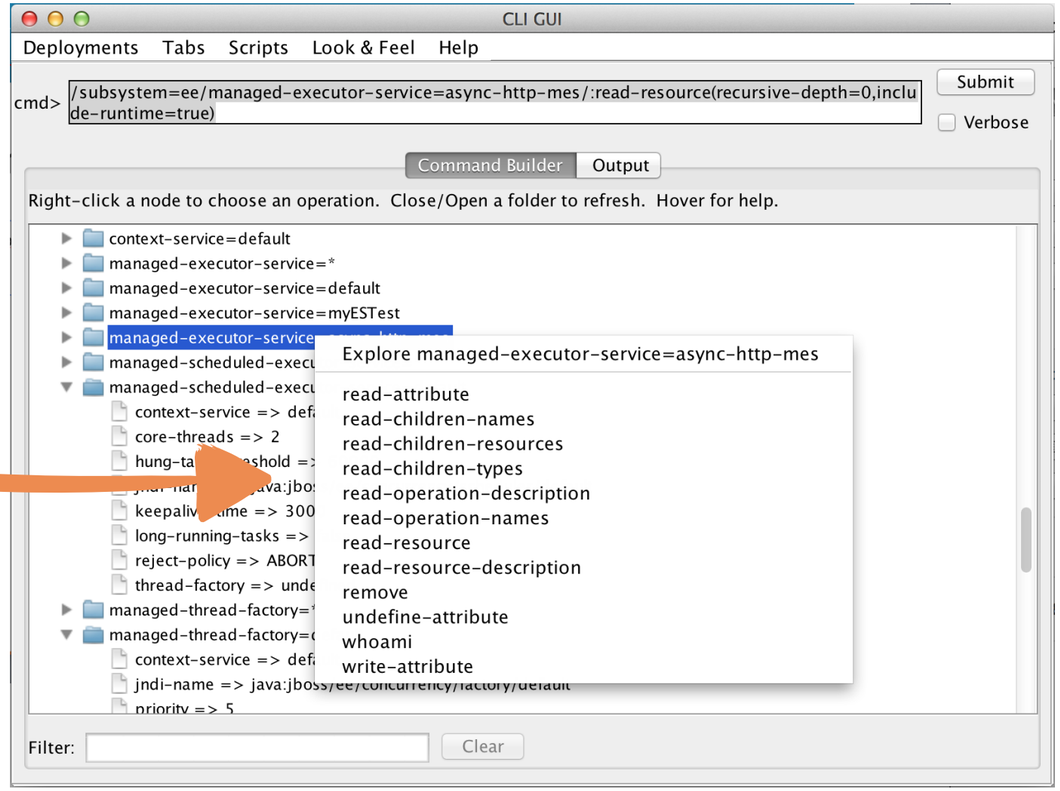
WildFly CLI comes with a command builder gui:

```
$JBOSS_HOME/bin/jboss-cli.sh --gui
```

Wildfly CLI command

WildFly CLI comes with a command builder gui:

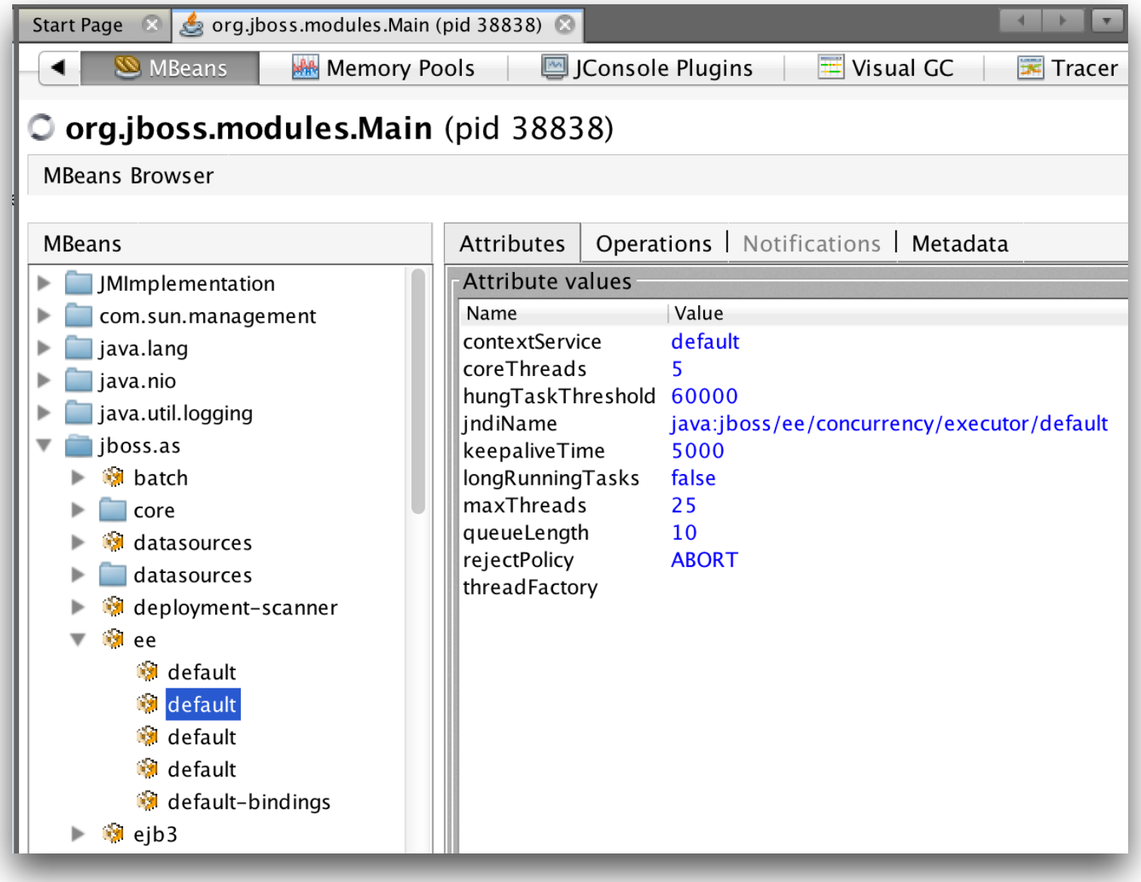
```
$JBOSS_HOME/bin/jboss-cli.sh --gui
```



Still too complicated?

Management & Monitoring

- Everything published as JMX MBeans
- -> values
- -> operations
- -> metadata



Ok, are we fine?

http://www.flickr.com/photos/upturnedface/2700127506/sizes/o/

1. The single ExecutorService

```
@Resource ManagedExecutorService mes;

protected void processRequest(ServiceRequest request) throws
        ExecutionException {
    Future<String> f = mes.submit(new Callable<String>() {...});
    System.out.println("Callable result: " + f.get());
}
```

Only used here??

```
@Resource ManagedExecutorService mes;

protected void processRequest(ServiceRequest request) throws
        ExecutionException {
    Future<String> f = mes.submit(new Callable<String>() {...});
    System.out.println("Callable result: " + f.get());
}
```

Only used here??

```
@Resource ManagedExecutorService mes;

protected void processRequest(ServiceRequest request) throws
        ExecutionException {
    Future<String> f = mes.submit(new Callable<String>() {...});
    System.out.println("Callable result: " + f.get());
}
```

```
@Resource(lookup="java:comp/service-request-executor")
```

2. The queue that queued

```
@Resource ManagedThreadFactory factory;

protected void processRequest(ServiceRequestData data) {

    // Creating Java SE style ExecutorService
    BlockingQueue<Runnable> queue = new LinkedBlockingQueue<Runnable>(10);
    ThreadPoolExecutor executor = new ThreadPoolExecutor(5, 10,
            0L, TimeUnit.MILLISECONDS,
            queue,
                factory);

    // MyRunnable has to implement .equals() and .hashcode()!!!
    MyRunnable task = new MyRunnable(data);

    // prevent duplicate queue entries
    if (!queue.contains(task)) {
        executor.submit(task);
    }
}
```

```
@Resource ManagedThreadFactory factory;

protected void processRequest(ServiceRequestData data) {

    // Creating Java SE style ExecutorService
    BlockingQueue<Runnable> queue = new LinkedBlockingQueue<Runnable>(10);
    ThreadPoolExecutor executor = new ThreadPoolExecutor(5, 10,
            0L, TimeUnit.MILLISECONDS,
            queue,
                factory);

    // MyRunnable has to implement .equals() and .hashcode()!!!
    MyRunnable task = new MyRunnable(data);

    // prevent duplicate queue entries
    if (!queue.contains(task)) {
        executor.submit(task);
    }
}
```

.contains() locks queue!

3. A self overtaking job

```
@Resource ManagedScheduledExecutorService executor;

protected void processRequest(ServiceRequest job) {
    ScheduledFuture<?> f = executor.scheduleAtFixedRate(
            new MyRunnableTask(job), 5, 10, TimeUnit.SECONDS);

}
```

4. Know your RejectionPolicy

Competition from Spring

```
<bean class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">
    <property name="maxPoolSize" value="5"/>
    <property name="queueCapacity" value="180"/>
    <property name="rejectedExecutionHandler">
        <bean class="java.util.concurrent.ThreadPoolExecutor$DiscardPolicy"/>
    </property>
</bean>
```

Competition from Spring

```
<bean class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">
    <property name="maxPoolSize" value="5"/>
    <property name="queueCapacity" value="180"/>
    <property name="rejectedExecutionHandler">
        <bean class="java.util.concurrent.ThreadPoolExecutor$DiscardPolicy"/>
    </property>
</bean>
```

ManagedExecutors can only ABORT

5. Go Java 8 and shine

WildFly compatible with Java 8

WildFly compatible with Java 8

Possibility to combine executors with

WildFly compatible with Java 8

Possibility to combine executors with

-› Lambdas

WildFly compatible with Java 8

Possibility to combine executors with

-› Lambdas

-› Streams

WildFly compatible with Java 8

Possibility to combine executors with

-› Lambdas

-› Streams

-› CompletableFuture

WildFly compatible with Java 8

Possibility to combine executors with

-› Lambdas

-› Streams

-› CompletableFuture

WildFly compatible with Java 8

Possibility to combine executors with

-› Lambdas

-› Streams

-› CompletableFuture

Are you sure??

Java 8 Streams API

Java 8 Streams API

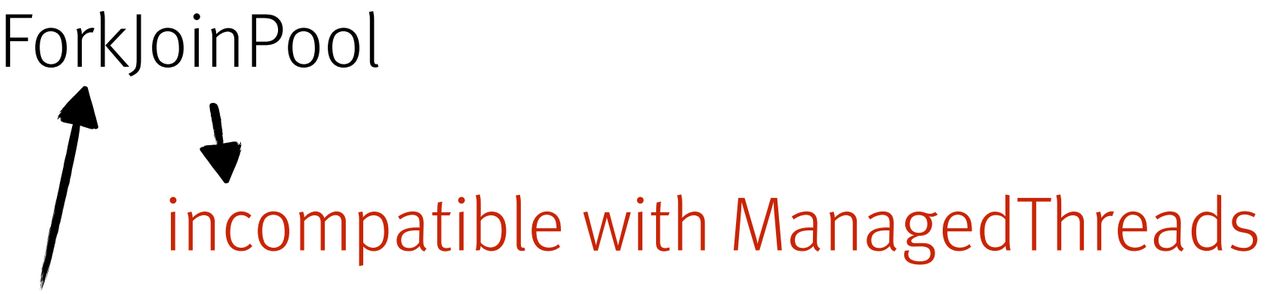
```
final List<Product> products = productUris.parallelStream()
        .map(productUri -> CompletableFuture.supplyAsync(() -> resolveProducts(productUri)))
        .map(CompletableFuture::join)
        .filter(res -> res.second.hasBody())
        .map(this::responsePairToProduct)
        .collect(Collectors.toList());
```

Java 8 Streams API

ForkJoinPool

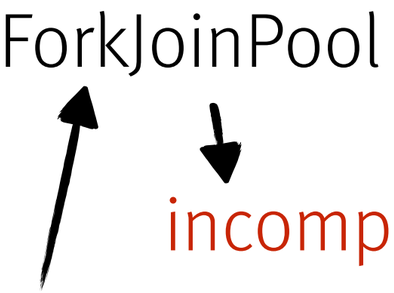
```
final List<Product> products = productUris.parallelStream()
        .map(productUri -> CompletableFuture.supplyAsync(() -> resolveProducts(productUri)))
        .map(CompletableFuture::join)
        .filter(res -> res.second.hasBody())
        .map(this::responsePairToProduct)
        .collect(Collectors.toList());
```

Java 8 Streams API




```
final List<Product> products = productUris.parallelStream()
        .map(productUri -> CompletableFuture.supplyAsync(() -> resolveProducts(productUri)))
        .map(CompletableFuture::join)
        .filter(res -> res.second.hasBody())
        .map(this::responsePairToProduct)
        .collect(Collectors.toList());
```

Java 8 Streams API



incompatible with ManagedThreads

```
final List<Product> products = productUris.parallelStream()
        .map(productUri -> CompletableFuture.supplyAsync(() -> resolveProducts(productUri)))
        .map(CompletableFuture::join)
        .filter(res -> res.second.hasBody())
        .map(this::responsePairToProduct)
        .collect(Collectors.toList());
```

Cannot recommend to use parallel Streams API in Java EE!

No Java 8? Try Guava!

https://code.google.com/p/guava-libraries/wiki/ListenableFutureExplained

No Java 8? Try Guava!

```
@Resource(name = "BackendCallExecutorService")
ManagedExecutorService executor;
```

https://code.google.com/p/guava-libraries/wiki/ListenableFutureExplained

No Java 8? Try Guava!

```
@Resource(name = "BackendCallExecutorService")
ManagedExecutorService executor;

final String productId = ...;
ListeningExecutorService service = MoreExecutors.listeningDecorator(executor);
```

https://code.google.com/p/guava-libraries/wiki/ListenableFutureExplained

No Java 8? Try Guava!

```
@Resource(name = "BackendCallExecutorService")
ManagedExecutorService executor;

final String productId = ...;
ListeningExecutorService service = MoreExecutors.listeningDecorator(executor);

ListenableFuture<Product> product = service.submit(new Callable<Product>() {
    public Product call() {
        return resolveProduct(productId);
    }
});
```

https://code.google.com/p/guava-libraries/wiki/ListenableFutureExplained

No Java 8? Try Guava!

```
@Resource(name = "BackendCallExecutorService")
ManagedExecutorService executor;

final String productId = ...;
ListeningExecutorService service = MoreExecutors.listeningDecorator(executor);

ListenableFuture<Product> product = service.submit(new Callable<Product>() {
    public Product call() {
        return resolveProduct(productId);
    }
});

Futures.addCallback(explosion, new FutureCallback<Product>() {
   public void onSuccess(Product product) {
        addProductToCart(product);
    }
    public void onFailure(Throwable thrown) {
        offerFallbackProduct(); // business decision!!
    }
});
```


https://code.google.com/p/guava-libraries/wiki/ListenableFutureExplained

http://www.flickr.com/photos/weaselmcfee/3721267658/sizes/o/

sounds familiar?

```
<management:executor-service
    id="async-http-mes"
    pool-size="20-500"
    queue-capacity="50"
    keep-alive="60"
    rejection-policy="ABORT" />
```

sounds familiar?

- Yes! Xebia came up with something alike for Spring in 2011

```
<management:executor-service
    id="async-http-mes"
    pool-size="20-500"
    queue-capacity="50"
    keep-alive="60"
    rejection-policy="ABORT" />
```

sounds familiar?

- Yes! Xebia came up with something alike for Spring in 2011

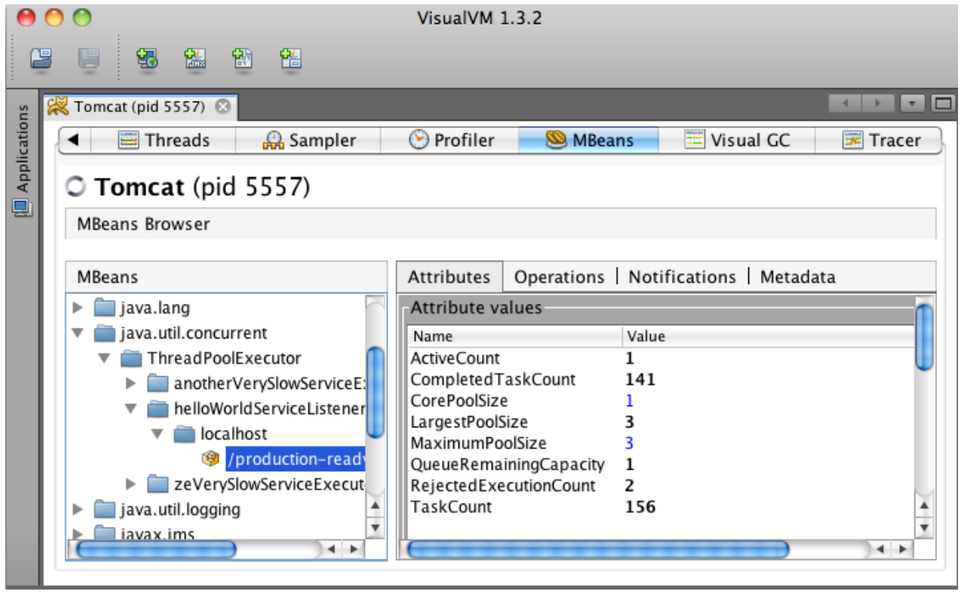
```
<management:executor-service
    id="async-http-mes"
    pool-size="20-500"
    queue-capacity="50"
    keep-alive="60"
    rejection-policy="ABORT" />
```

WildFly

```
async-http-mes/:add(
    core-threads=20,
    jndi-name=java:comp/async-http-mes,
    context-service=default,
    hung-task-threshold=110000,
    keepalive-time=60000,
    thread-factory=default,
    queue-length=50,
    reject-policy=ABORT,
    max-threads=500)
```

sounds familiar?

```
<management:executor-service
      id="async-http-mes"
      pool-size="20-500"
      queue-capacity="50"
      keep-alive="60"
      rejection-policy="ABORT" />
```

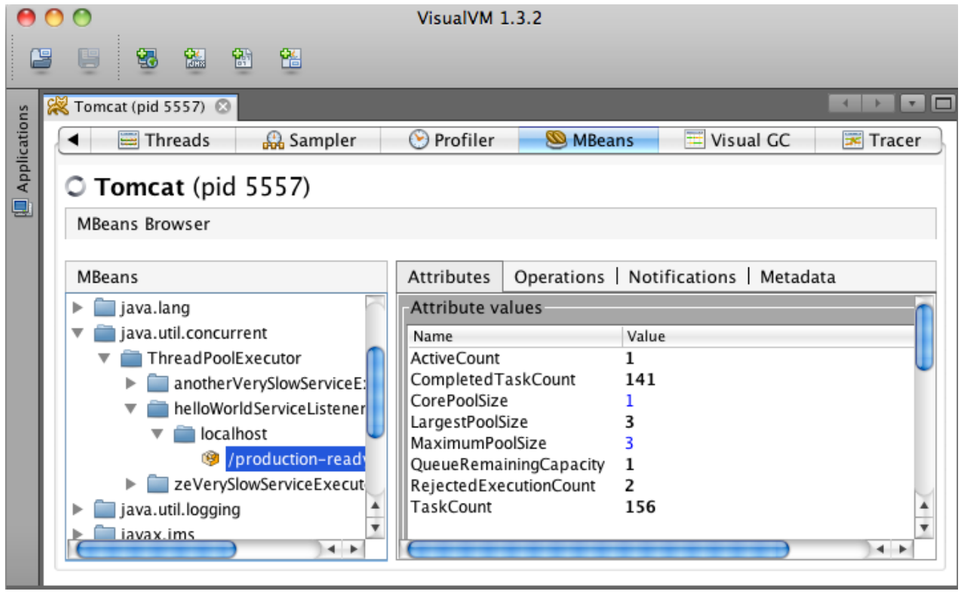


https://code.google.com/p/xebia-france/wiki/XebiaManagementExtras

sounds familiar?

- Yes! Xebia came up with something alike for Spring in 2011

```
<management:executor-service
      id="async-http-mes"
      pool-size="20-500"
      queue-capacity="50"
      keep-alive="60"
      rejection-policy="ABORT" />
```



https://code.google.com/p/xebia-france/wiki/XebiaManagementExtras

A note on Hystrix

- Netflix Hystrix is a well-known implementation of the stability patterns - especially Circuit-Breaker
- Hystrix in Java EE is no free lunch
  - set env var

    “`-Dhystrix.plugin.HystrixConcurrencyStrategy.implementation=\`
    `full.package.of.MyConcurrencyStrategy`”

Spring 4.0 supports JSR236

Provides configurability & monitoring via JSR236

-› ConcurrentTaskExecutor auto-detects JSR236 classes

-› Uses specified ManagedExecutor transparently

-› executor manageable via CLI or JMX

-› Check the code, it’s open-source! **@Github #4004e53**

Summary

- good mix of convenience & flexibility
- still potential for optimisation:
  - API to create ManagedExecutorService instead of defaults
  - validation/ constraints for ManagedExecutorService properties
  - Runtime.availableProcessors() for parallelism is deceptive!!!
  - Please become compatible with ForkJoinPool/ JDK8 Streams API

Links

JSR236 spec: https://jcp.org/en/jsr/detail?id=236

Java EE 7 JavaDoc: http://docs.oracle.com/javaee/7/api/

Java EE 7 samples: https://github.com/javaee-samples/javaee7-samples

Wildfly docs: https://docs.jboss.org/author/display/WFLY8/EE+Subsystem+Configuration

Thanks for your attention!

Alexander Heusingfeld, **@goldstift**

alexander.heusingfeld@innoq.com

https://www.innoq.com/people/alex/

innoQ