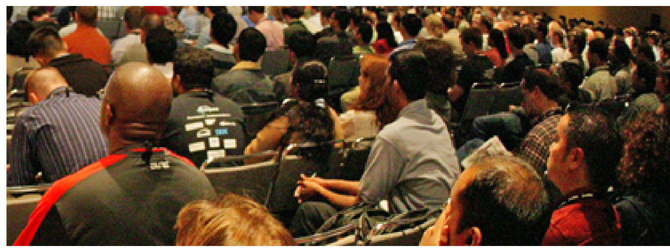
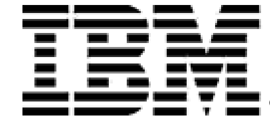

the POWER of JAVA™


JavaOne

JSRs 236 and 237; Concurrency Utilities for Java EE in Practice

Chris D Johnson
cdjohnson@us.ibm.com
IBM, Rochester Lab

Naresh Revanuru
naresh@bea.com
BEA Systems, Inc.

Session ID# BOF-0989

Goals of This Talk

What Your Audience Will Gain

Learn how to leverage concurrency in your Java EE applications.

Agenda

Introduction

Overview

ManagedThreadFactory

ManagedExecutorService

ManagedScheduledExecutorService

ContextService

Summary

Agenda

Introduction
Overview
ManagedThreadFactory
ManagedExecutorService
ManagedScheduledExecutorService
ContextService
Summary

Introduction

Brief History

- Java SE APIs
  - Java SE Timer and Thread
  - JSR-166 concurrent utilities for Java SE 5 (TS-4915)
- Java EE APIs
  - BEA-IBM Commonj API for Java EE environment
  - JSR 236-237 provides context aware Thread Pools and Timers to Java EE applications
  - Vendor-propietary APIs
- Reusing and extending existing Java SE 5 concurrency foundations
- Formalize Java EE concurrency specification through JCP. Adopt in next version of Java EE.

Introduction

JSR 236-237 group composition

- Specification Leads
  - Chris D Johnson, IBM
  - Naresh Revanuru, BEA
- Expert Group members
  - Andrew Evers, Redwood Software
  - Cameron Purdy, Tangosol
  - Cyril Bouteille, Hotwire
  - Doug Lea, JSR-166 lead
  - Gene Gleyzer, Tangosol
  - Pierre Vignéras

Introduction

Current Status

- Early draft preview published on 4/28/2006
- Draft available at
  - http://gee.cs.oswego.edu/dl/concurrencyee-interest
- Comments are very welcome
- Plan to turn it into official JSR draft
- EG discussion currently happening outside of JCP site

Agenda

Introduction

Overview

ManagedThreadFactory

ManagedExecutorService

ManagedScheduledExecutorService

ContextService

Summary

Overview

Limitation of concurrency in Java EE

- Servlet and EJB specifications explicitly prohibit or are ambiguous about threading support. (Promotes synchronous activity.)
- Java SE threads and timers are not well integrated with Java EE containers
- java.util.concurrent APIs are extensible
  - Need some enhancements for Java EE environments
  - Basis for these JSRs
- Existing solutions do not propagate thread context like class loader, security, naming and do not have manageability and isolation semantics.

Overview

Concurrency uses in Java EE

- Decouple user execution from slow moving background processing
- Improvements in processor architecture promote parallelism
- One big task into smaller concurrent tasks
- Asynchronous notification use case
- Timer use cases like periodic cleanup, cache maintenance

Overview

Special Java EE requirements

- Coordination between application server lifecycle and asynchronous task lifecycle
  - Server shutdown
  - Application deployment/undeployment
- Application-scoped threads
- Thread scheduling based on application resource constraints
- Intelligent workload classification and routing
- Application isolation

Overview

Goals of Concurrency Utilities for Java EE

- Provide consistent programming model
- Leverage existing technology to provide migration from Java SE
- Allow adding concurrency to existing applications
- Allow integration with previous Java EE versions
- Provide simple API for simple use cases
- Provide flexible API for advanced use cases

Extending Java SE

Administered Objects

- Extend existing Java SE 5 concurrency utilities by providing managed versions:
  - ManagedThreadFactory
  - ManagedExecutorService
  - ManagedScheduledExecutorService
- Add Java EE extensions
  - ContextService
  - ManagedTaskListener
  - Trigger
  - Identifiable

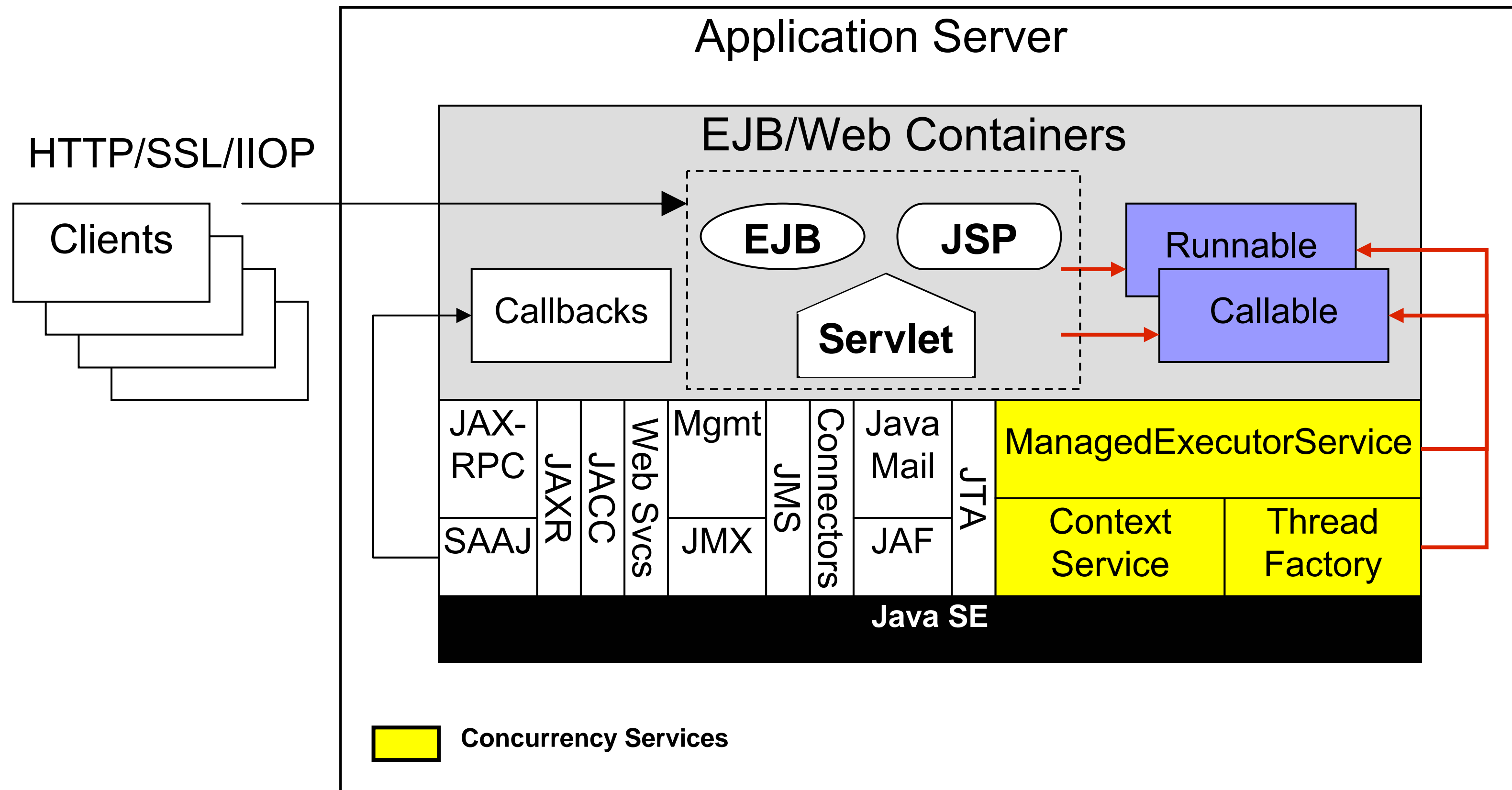
Extending Java SE

Managed Objects

- Provide manageability using JMX MBeans
  - ManagedThread
  - ManagedThreadFactory
  - ManagedExecutorService

Extending Java EE

Java EE Architecture Diagram with Concurrency



Agenda

Introduction

Overview

ManagedThreadFactory

ManagedExecutorService

ManagedScheduledExecutorService

ContextService

Summary

ManagedThreadFactory

Overview

- Standard interface and method for creating threads
  - `Thread newThread(Runnable r)`
- Centrally defined on an application server
- Indirectly referenced by applications
- Java EE product providers provide the thread
- Extension of Java SE 5 ThreadFactory
  - Adds container context and manageability
  - UserTransaction support (does not enlist in parent component's transaction)

ManagedThreadFactory

Usage Scenarios

- Long Running Tasks
  - Work Consumers/Producers
  - Batch jobs
  - Embedded servers
- Custom Thread Pools
  - Use Java SE thread pools
  - Any service that can use ThreadFactory

Code Sample - Daemon

```
// Within your servlet or EJB method…
// Lookup the ThreadFactory
InitialContext ctx = new InitialContext();
ManagedThreadFactory tf = (ManagedThreadFactory)
    ctx.lookup("java:comp/env/concurrent/myTF");

// Create and start the thread.
Thread daemonThread = tf.newThread(myDaemonRunnable);
daemonThread.start();

// The runnable behaves as-if it were running in the
// servlet or EJB container.
// The thread's lifecycle is tied to the application and
// is interrupted.
```

Code Sample – Custom Thread Pool

```
// Within your servlet or EJB method…
// Lookup the ThreadFactory
@Resource
ManagedThreadFactory tf;

void businessMethod() {
 // Use a custom Java SE ThreadPoolExecutor
 CustomThreadPoolExecutor pool =
     new CustomThreadPoolExecutor(coreSize, maxSize, tf);

 // When the executor allocates a new thread, the
 // thread will use the current container context.
```

ManagedThreadFactory

Thread Management with JMX

- Monitor when threads are allocated using the ManagedThreadFactory MBean
- Monitor thread activity and health
  - What task is running on the thread?
  - How long has the task been running?
  - Correlate to the Java SE thread name and id.
- Cancel a thread (cooperative)
  - Hung threshold notifications help identify problems.
  - Proper interruption detection is essential in the task implementation.

ManagedThreadFactory

Identifiable Tasks

- Runnables that are run on a managed thread may optionally implement the Identifiable interface.
- Allows runtime introspection of thread's current state.
- Exposed on the ManagedThread MBean
- Short name available as an attribute
- Locale-specific description available as an attribute for the current locale or an operation for alternative locales.

Code Sample - Identifiable

```
class MyConsumerTask implements Runnable, Identifiable {
    private String currentName;
    public void run() {
        // Update the identity name periodically
        currentName="MonitorApp:MyConsumerTask:Phase1";
        …
        currentName="MonitorApp:MyConsumerTask:Phase2";
    }
    public String getIdentityName() {
        // Called by ManagedThread.taskIdentityName
        return currentName;
    }
    public String getIdentityDescription(Locale l) {
        // Called by ManagedThread.taskIdentityDescription
        // Get description from NLS bundle
    }
}
```

Agenda

Introduction

Overview

ManagedThreadFactory

ManagedExecutorService

ManagedScheduledExecutorService

ContextService

Summary

ManagedExecutorService

Overview

- Typical way of running tasks asynchronously from a Java EE container method
- Centrally defined on an application server
- Indirectly referenced by applications
- Java EE product providers provide the implementation
- Typically used for centralized thread pooling
- Implementations may offer extended capabilities

ManagedExecutorService

Overview continued…

- Extension of Java SE 5 ExecutorService
  - Adds container context, manageability and lifecycle tracking and constraints
  - UserTransaction support (does not enlist in parent component transaction)
  - Distributed (remote) capability
- Container context may be component-managed or server-managed
  - Server-managed is most common. Share a single executor between applications and components.
  - Component-managed is faster, but restricted to a single component (no container context switching)

ManagedExecutorService

Usage Scenarios

- Single server-managed thread pool
  - Most typical usage.
  - Easiest to use. Server manages the lifecycle.
  - Multiple applications share a single executor
  - Application developer defines the requirements of the executor:
    - What container contexts to propagate (e.g. namespace)
    - Server-managed
  - Deployer configures the appropriate executor and maps the resource environment reference to the executor

ManagedExecutorService

Usage Scenarios continued…

- Multiple component-managed thread pools
  - High performance scenario
  - A component has one executor and controls its lifecycle.
  - Container context is fixed.
  - Application developer defines the requirements of the executor:
    - What container contexts to propagate (e.g. namespace)
    - Component-managed
  - Deployer configures the appropriate executor definition and maps the resource environment reference to the executor

ManagedExecutorService

```
interface ManagedExecutorService extends ExecutorService {
  Future<?> submit(Runnable task,
   ManagedTaskListener taskListener);

  <T> Future<T> submit(Runnable task, T result,
   ManagedTaskListener taskListener);

  <T> Future<T> submit(Callable<T> task,
   ManagedTaskListener taskListener);

  // Time-out versions of invokeAll/Any available too…
  <T> List<Future<T>> invokeAll(Collection<? Extends
   Callable<T>> tasks, ManagedTaskListener taskListener);

  <T> T invokeAny(Collection<? extends Callable<T>> tasks,
   ManagedTaskListener taskListener)
}
```

ManagedExecutorService

Management

- Hung tasks can be monitored and cancelled using JMX.
  - Threads are created from a ManagedThreadFactory
  - Each thread therefore is associated with a ManagedThread MBean
  - Tasks can be Identifiable
- Task lifecycle can be monitored using ManagedTaskListeners
  - Monitoring extensions (logging)
  - Work-flow control and management

ManagedExecutorService

ManagedTaskListener

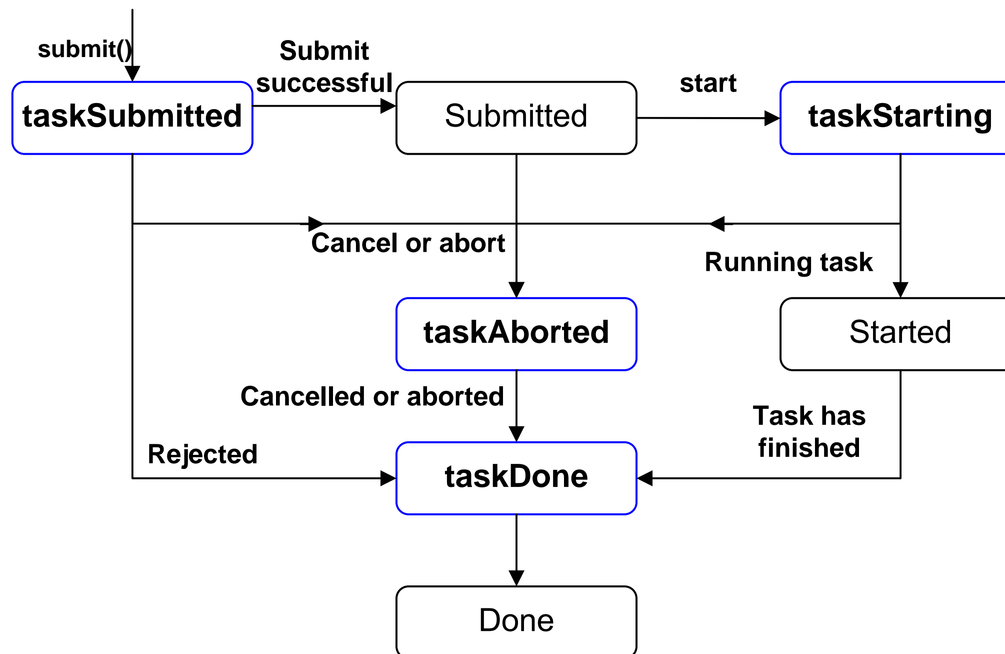
- Listeners are Java objects that are registered with the task when submitted to the executor.
- The listener method runs in the same container context as the task.
  - taskSubmitted – The task was submitted to the executor
  - taskAborted – The task was unable to start or was cancelled.
  - taskStarting – The task is about to start
  - taskDone – The task has completed (successfully or otherwise)

ManagedTaskListener

```
interface ManagedTaskListener {
  void taskSubmitted(Future<?> future,
    ManagedExecutorService executor);

  void taskAborted(Future<?> future,
    ManagedExecutorService executor);

  void taskDone(Future<?> future,
    ManagedExecutorService executor);

  void taskStarting(Future<?> future,
    ManagedExecutorService executor);

}
```

ManagedExecutorService

ManagedTaskListener - Lifecycle



Code Sample – Typical Parallelism

```
// Within your servlet or EJB method…
@Resource
ManagedExecutorService mes;
void businessMethod() {
 Callable<Integer> c = new Callable<Integer>() {
    Integer call() {
        // Interact with a database... Return answer.
        // The namespace is available here!
    }
 }
 // Submit the task and do something else.  The task
 // will run asynchronously on another thread.
 Future result = mes.submit(c);
 ...
 // Get the result when ready…
 int theValue = result.get();
 ...
```

ManagedExecutorService

Distributable Overview

- Same rules as a ManagedExecutorService
- Allows distributing the task to a peer on another server instance (JVM).
  - Task must implement serializable
- Providers do not have to supply a distributable version.
- Two distributable types are available:
  - With and without affinity

Agenda

Introduction

Overview

ManagedThreadFactory

ManagedExecutorService

ManagedScheduledExecutorService

ContextService

Summary

ManagedScheduledExecutorService

Overview

- Typical way of running periodic tasks asynchronously from a Java EE container method
- Typically used for transient timers
- Inherits semantics of ManagedExecutorService:
  - Centrally defined on an application server
  - Indirectly referenced by applications
  - Java EE product providers provide the implementation
  - Implementations may offer extended capabilities

ManagedScheduledExecutorService

Overview continued…

- Extension of ScheduledExecutorService
  - Adds container context, manageability and lifecycle tracking and constraints
  - UserTransaction support (does not enlist in parent component transaction)
  - Trigger mechanism.
- Container context may be component-managed or server-managed
  - Server-managed is most common. Share a single executor between applications and components.
  - Component-managed is faster, but restricted to a single component.

ManagedScheduledExecutorService

Usage Scenarios

- Periodic cache invalidations
- Request timeouts
- Polling
- Custom Scheduler
  - Would need implementation extension to support persistence.
  - Use Triggers for custom calendaring:
    - N-time fixed-rate with time-sensitive skip.
    - Run time based on previous task calculation result.
    - Condition-based trigger
    - Centralized business calendar.

ManagedScheduledExecutorService

```
interface ManagedScheduledExecutorService extends
  ScheduledExecutorService {
  // Same methods as ScheduledExecutorService…
  // Add ManagedTaskListener and Trigger
  ScheduledFuture<?> schedule(Runnable command,
    long delay,  TimeUnit unit,
    ManagedTaskListener taskListener);

  ScheduledFuture<?> schedule(Runnable command,
    Trigger trigger, ManagedTaskListener taskListener);

  ScheduledFuture<?> scheduleAtFixedRate(Runnable command,
    long initialDelay,  long period, TimeUnit unit,
    ManagedTaskListener taskListener);

  ScheduledFuture<?> scheduleWithFixedDelay(
    Runnable command, long initialDelay,  long delay,
    TimeUnit unit, ManagedTaskListener taskListener);
}
```

Trigger

```
interface Trigger {

  // Return true if you want to skip the
  // currently-scheduled execution.  Is invoked after
  // taskStarting().
  boolean skipRun(Future lastFuture,
    Date scheduledRunTime);

  // Retrieves the time in which to run the task next.
  // Invoked during submit time and after each task has
  // completed.
  Date getNextRunTime(Future lastFuture, Date baseTime,
    Date lastActualRunTime, Date lastScheduledRunTime,
    Date lastCompleteTime);

}
```

Agenda

Introduction

Overview

ManagedThreadFactory

ManagedExecutorService

ManagedScheduledExecutorService

ContextService

Summary

ContextService

Overview

- Mechanism for applications to capture container context and run within that context later
  - ManagedExecutorServices likely to use this service internally to propagate container context.
- Centrally defined on an application server
- Indirectly referenced by applications
- Java EE product providers provide the implementation
- Implementations may offer extended capabilities

ContextService

Overview continued…

- Current thread context is captured and stored within a context proxy for your object
- Serializable
- Customizable
  - Can enable transaction pass-through
- Used in advanced scenarios.
- Use with non-ManagedThreadFactory-created threads (threads created with new Thread())

ContextService

Use Cases

- Workflow
  - Store and propagate user identity
- Java SE or third-party thread reuse
  - Allows thread to behave as-if it were on a container thread.
- Hybrid ManagedExecutorService
  - Use component-managed executor from multiple components.

ContextService

```
interface ContextService {
  String USE_PARENT_TRANSACTION = "ctxsvc.useparenttran";

  Object createContextObject(Object instance,
    Class[] interfaces);

  Object createContextObject(Object instance,
    Class[] interfaces, Properties contextProperties);

  void setProperties(Object contextObject,
    Properties contextProperties);

  Properties getProperties(Object contextObject);
}
```

Code Example – Creating Context

```
// Within your servlet or EJB method…
@Resource
ContextService ctxSvc;
void businessMethod() {
 Runnable runnableTask = new Runnable() {
     void run() {
      // Interact with a database… use component's security
    }
 }
 // Wrap with the current context
 Runnable runnableTaskWithCtx = (Runnable)
   ctxSvc.createContextObject(runnableTask,
   new Class[]{Runnable.class}

 // Store the runnable with context somewhere and run
 // later..
 store.putIt(runnableTaskWithCtx);
```

Code Example – Using Context

```
// Retreive the Runnable with Context
Runnable runnableTaskWithContext = store.getIt();

// Runnable will run on this thread, but with the context
// of the servlet/EJB that created it.
runnableTaskWithContext.run();

// If the Runnable implemented Serializable and it
// was serialized/deserialized… the context would still
// come with it.
```

Agenda

Introduction
Overview
ManagedThreadFactory
ManagedExecutorService
ManagedScheduledExecutorService
ContextService
Summary

Summary

- The Concurrency Utilities for Java EE is in Early Draft Review stage.
  - Mailing list available for comments.
- Extends Java SE concurrency utilities
- Provides simple and advanced APIs for adding concurrency to J2EE 1.3 and later applications:
  - ManagedThreadFactory
  - ManagedExecutorService
  - ManagedScheduledExecutorService
  - ContextService

For More Information

- Concurrency EE Interest Site and Specification
  - http://gee.cs.oswego.edu/dl/concurrencyee-interest/
- JSR 236 and 237
  - http://www.jcp.org/en/jsr/detail?id=236
  - http://www.jcp.org/en/jsr/detail?id=237
- Related Sessions
  - TS-4915 – Concurrency Utilities in Java SE 5

Q&A

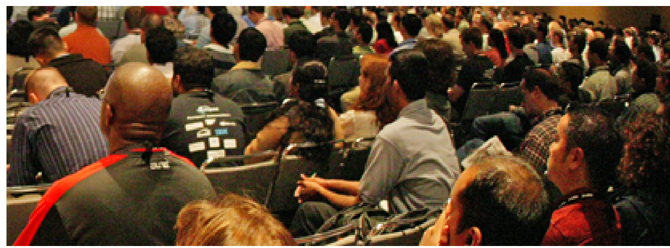
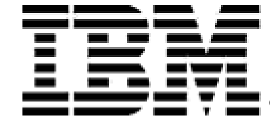
Chris D. Johnson, IBM Corp.

Naresh Revanuru, BEA Systems, Inc.

Andrew Evers, Redwood Software

Cyril Bouteille, Hotwire.com

<code/>

the POWER of JAVA™

JavaOne

JSRs 236 and 237; Concurrency Utilities for Java EE in Practice

Chris D Johnson
cdjohnson@us.ibm.com
IBM, Rochester Lab

Naresh Revanuru
naresh@bea.com
BEA Systems, Inc.

Session ID# BOF-0989