Spring Breakout Session
Spring JMS

by Vlad Costel Ungureanu
for FII UAIC, Learn Stuff
and Endava Pass It On

Spring Breakout Session

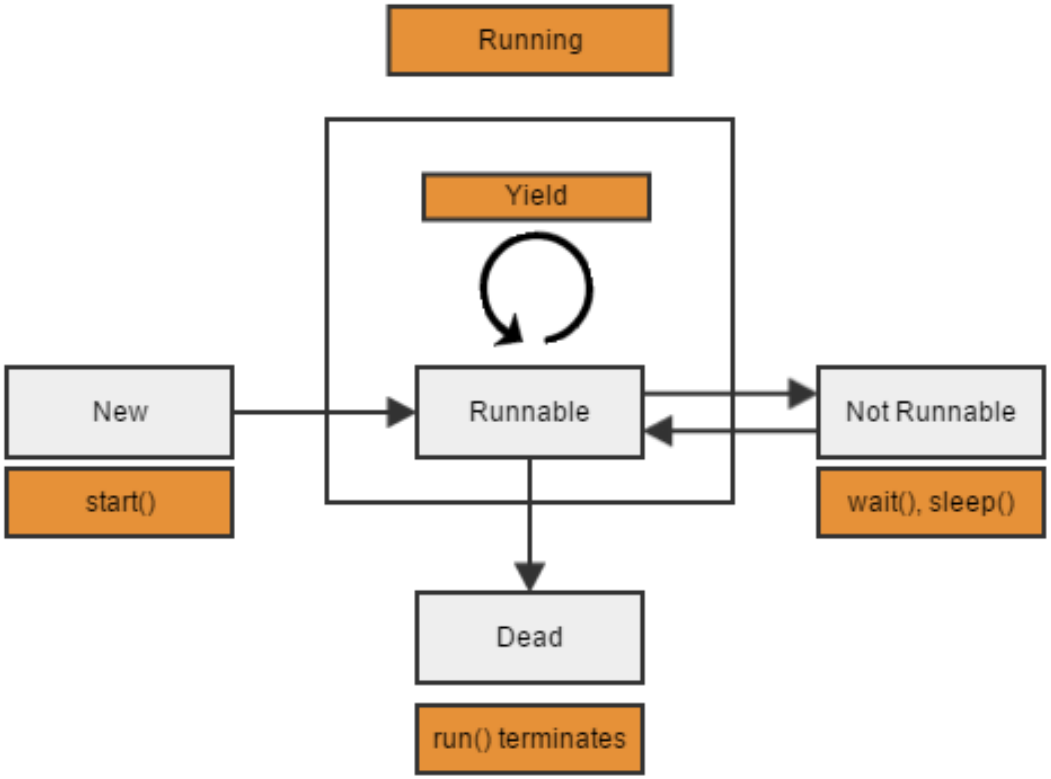
Spring JMS

- Asyncronicity
- Asyncronicity - Threads
- Asyncronicity – Executor Service
- Asyncronicity – Thread Pool Executor
- Asyncronicity – Ruisks
- Asyncronicity – Overview
- @Async
- Queues
- Active MQ
- Spring JMS – Producer Configuration
- Spring JMS Producer
- Spring JMS – Consumer Configuration
- Spring JMS Consumer
- Overview

Asyncronicity

- Some operations or processes can take too long to finish
- Sometimes we don't want to wait for the confirmation that an operation has been successfully performed
- Some things are better executed in parallel

Asynchronicity - Threads

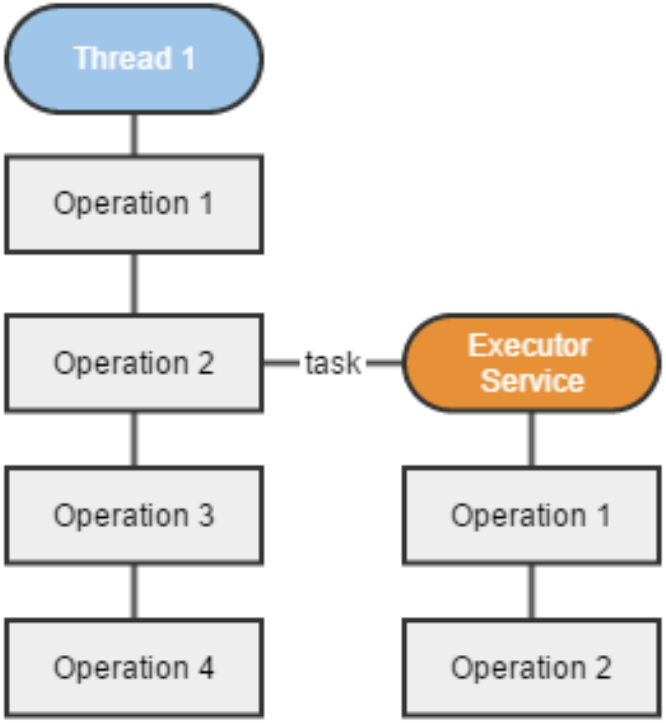


Asyncronicity - Threads

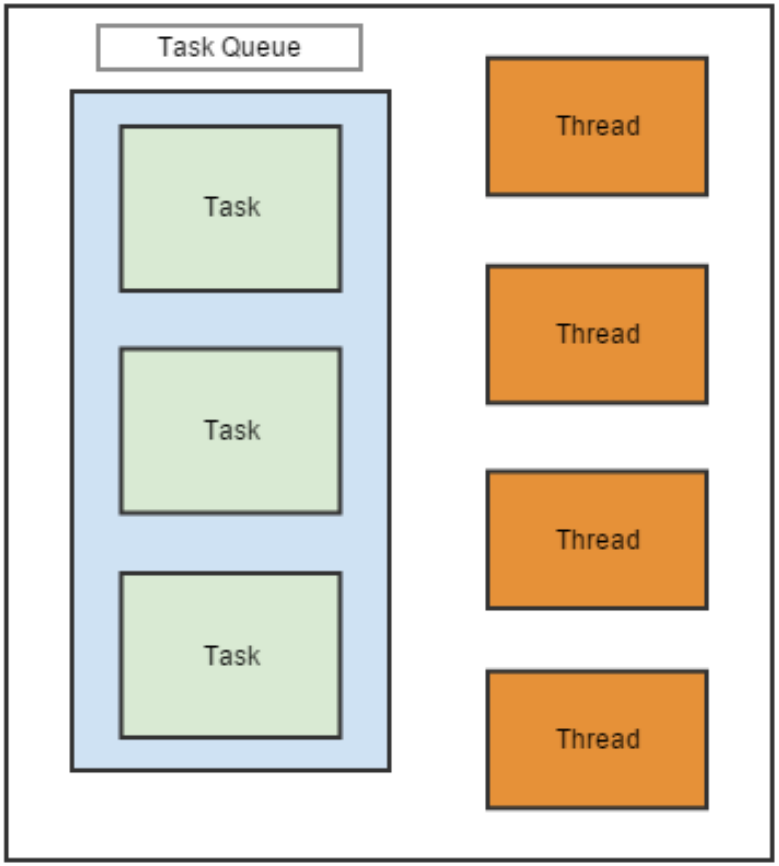
```
public class HelloRunnable implements Runnable {
    public void run() {
        System.out.println("Hello from a thread!");
    }
    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }
}

public class HelloThread extends Thread {
    public void run() {
        System.out.println("Hello from a thread!");
    }
    public static void main(String args[]) {
        (new HelloThread()).start();
    }
```

Asyncronicity – Executor Service



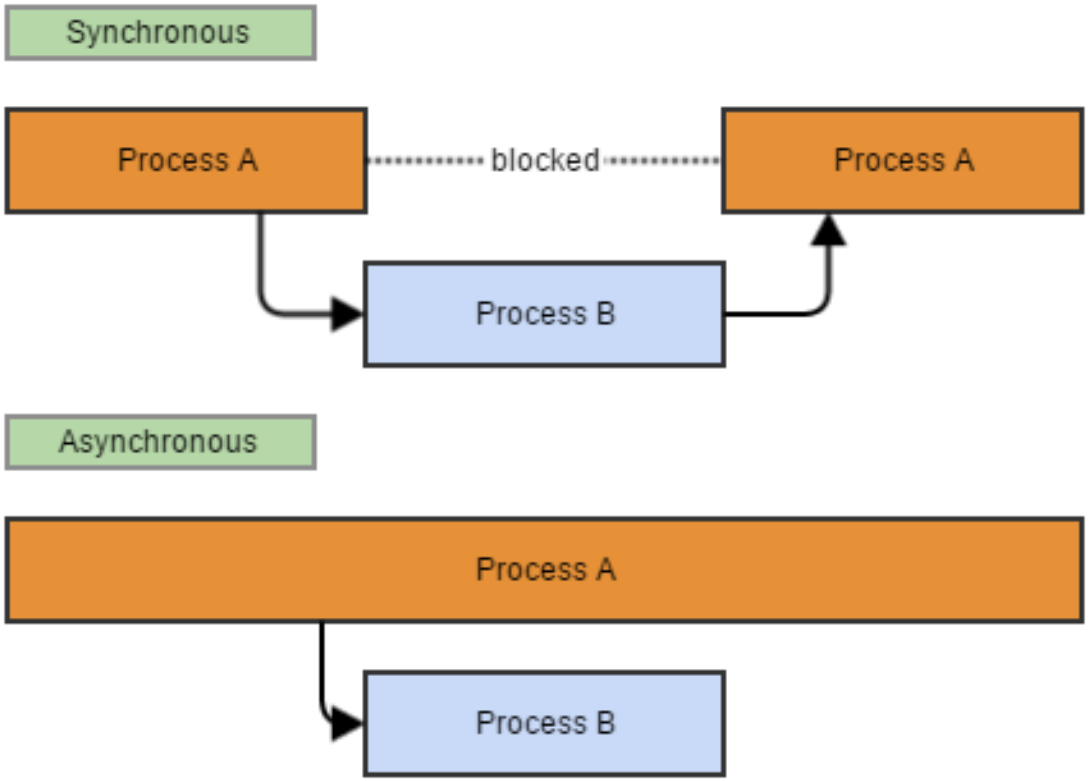
Asyncronicity – Thread Pool Executor



Asyncronicity - Risks

- ❑ Deadlock
- ❑ Resource consumptions
- ❑ Inconsistencies in the values shared by threads
- ❑ Waiting for too long to gain access to a resources

Asyncronicity - Overview



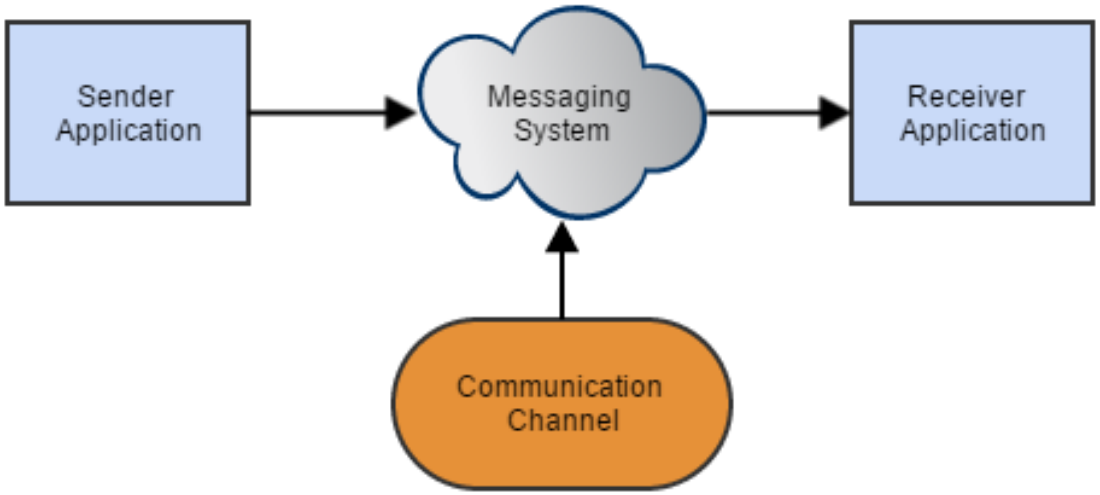
@Async

```
@Async
public Future<Page> findPage(String page) throws InterruptedException {
        Page results = restTemplate.getForObject("http://graph.facebook.com/" + page,
Page.class);
        Thread.sleep(1000L);
        return new AsyncResult<Page>(results);
}
```

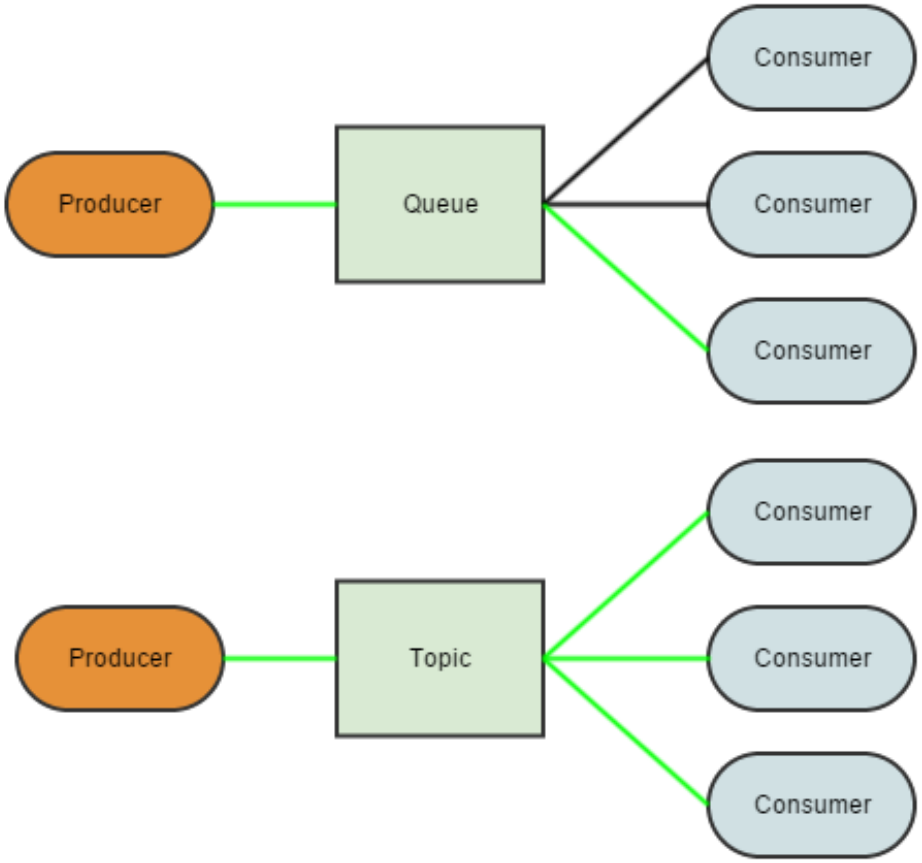
- A future object returns the result of an async processing method
- It offers a method of control for execution
- The actual value for a future can only be obtained through a getter method call
- If the thread obtaining the future has not finished it will block execution

- If a method with the @Async annotation returns void the execution will never get blocked

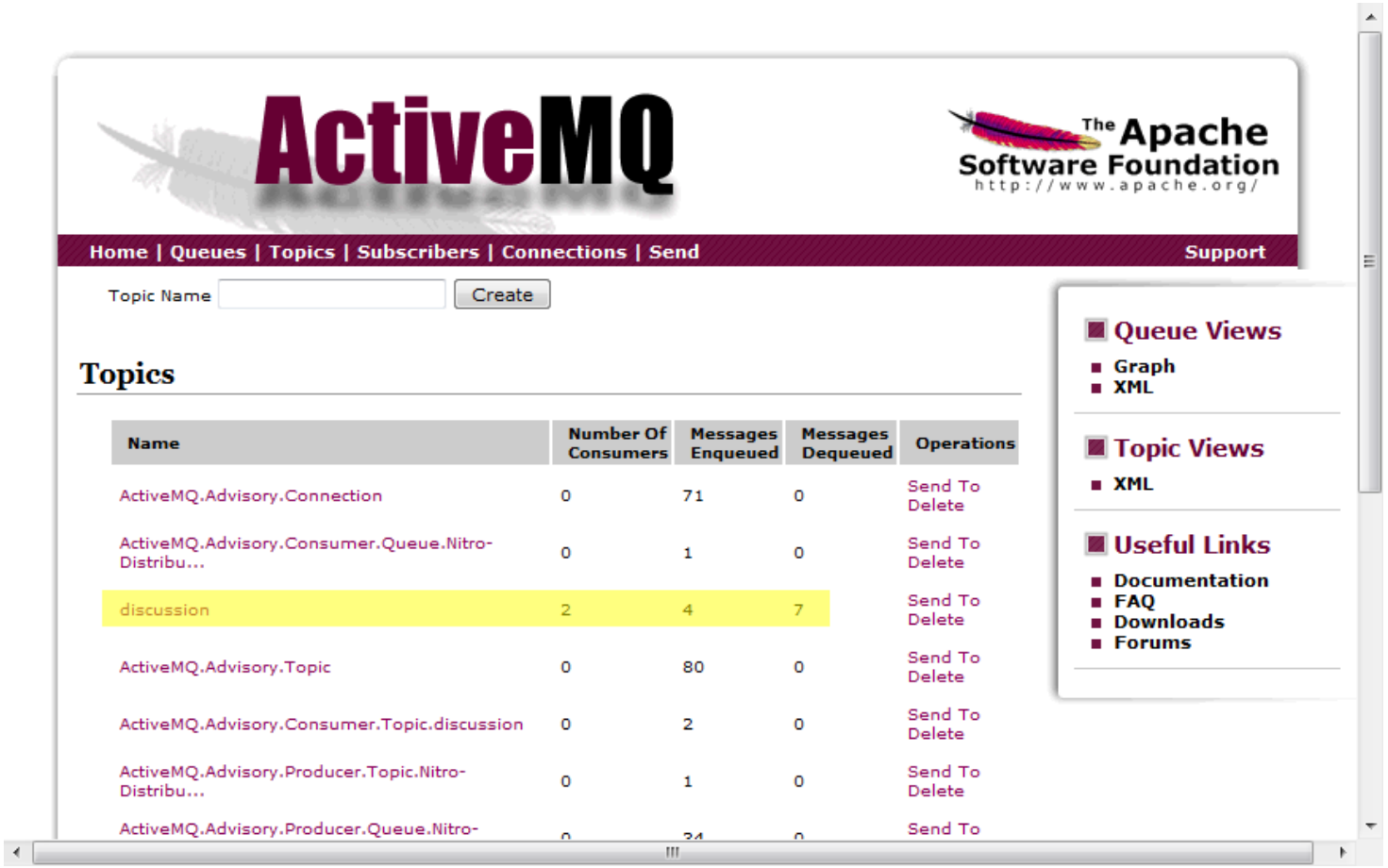
Queues


Sender Application
Messaging System
Receiver Application
Communication Channel

Queue vs Topic



Active MQ


ActiveMQ
The Apache Software Foundation
http://www.apache.org/
Home | Queues | Topics | Subscribers | Connections | Send
Support
Topic Name
Create
Topics
Name
Number Of Consumers
Messages Enqueued
Messages Dequeued
Operations
ActiveMQ.Advisory.Connection 0 71 0 Send To Delete
ActiveMQ.Advisory.Consumer.Queue.Nitro-Distribu... 0 1 0 Send To Delete
discussion 2 4 7 Send To Delete
ActiveMQ.Advisory.Topic 0 80 0 Send To Delete
ActiveMQ.Advisory.Consumer.Topic.discussion 0 2 0 Send To Delete
ActiveMQ.Advisory.Producer.Topic.Nitro-Distribu... 0 1 0 Send To Delete
ActiveMQ.Advisory.Producer.Queue.Nitro- Send To
Queue Views
Graph
XML
Topic Views
XML
Useful Links
Documentation
FAQ
Downloads
Forums

Spring JMS – Producer Configuration

```
<bean id="someactions-queue" class="org.apache.activemq.command.ActiveMQQueue">
        <constructor-arg value="someaction.queue" />
</bean>
<!-- ConnectionFactory Definition -->
<bean id="connectionFactory"
        class="org.springframework.jms.connection.CachingConnectionFactory">
        <constructor-arg ref="amqConnectionFactory" />
</bean>
<!-- Default Destination Queue Definition -->
<bean id="defaultDestination" class="org.apache.activemq.command.ActiveMQQueue">
        <constructor-arg index="0" value="someaction.queue" />
</bean>
<!-- JmsTemplate Definition -->
<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
        <property name="connectionFactory" ref="connectionFactory" />
        <property name="defaultDestination" ref="defaultDestination" />
        <property name="defaultDestinationName" value="someaction.queue" />
</bean>
<!-- Message Sender Definition -->
<bean id="someactionProducer" class="somepackage.producers.someactionProducer">
        <constructor-arg index="0" ref="jmsTemplate" />
</bean>
```

Spring JMS Producer

```
@Component
public class SomeActionProducer {

        private final JmsTemplate jmsTemplate;

        public SomeActionProducer() {
                this.jmsTemplate = new JmsTemplate();
        }
        public SomeActionProducer(final JmsTemplate jmsTemplate) {
                this.jmsTemplate = jmsTemplate;
        }

        public void send(final SomeActipon someActions) {
                jmsTemplate.send(new MessageCreator() {
                        public Message createMessage(Session session) throws JMSException {
                                return session.createObjectMessage(someActions);}
                });      }
}
```

Spring JMS – Consumer Configuration

```
<bean id="someactions-queue" class="org.apache.activemq.command.ActiveMQQueue">
        <constructor-arg value="someaction.queue" />
</bean>

<bean id="somepayments-queue" class="org.apache.activemq.command.ActiveMQQueue">
        <constructor-arg value="somepayment.queue" />
</bean>

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
        <property name="connectionFactory" ref="connectionFactory" />
</bean>

<bean id="someactionHandler" class="somepackage.handlers.someactionHandler" />

<bean
        class="org.springframework.jms.listener.DefaultMessageListenerContainer">
        <property name="connectionFactory" ref="connectionFactory" />
        <property name="maxConcurrentConsumers" value="25" />
        <property name="destinationName" value="someaction.queue" />
        <property name="messageListener" ref="someactionHandler" />
        <property name="sessionTransacted" value="false" />
</bean>
```

Spring JMS - Consumer

```
@Service
public class SomeActionHandler  implements  MessageListener {

        public void onMessage(Message  message) {
                try {
                        SomeAction someAction = (SomeAction) ((ObjectMessage) message).getObject();
                } catch (JMSException e) {
                        // log exception
                }
        }
}
```

Assignments

- ✓ For the previous application use a JMS queue to pass messages larger than 160 characters
- ✓ Include the producer of the message in a distinct REST service
- ✓ Call the appropriate REST service based on UI based validation
- ✓ Create a failed messages queue for messages that cannot be processed
- ✓ Create a consumer for the failed messages queue

THANK YOU!

Vlad Costel Ungureanu
ungureanu_vlad_costel@yahoo.com

This is a free course from LearnStuff.io
– not for commercial use –