JBoss Remoting and JBoss Serialization

Improving performance of JBoss Application Server

Tom Elrod & Clebert Suconic, JBoss Inc.

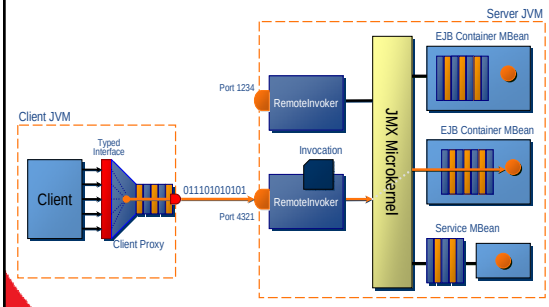
Overview

- Using JBoss Remoting and JBoss Serialization in favor of traditional detached invokers.
- JBoss Remoting overview
- JBoss Serialization overview
- Where JBoss Remoting and JBoss Serialization can (and will be) used
- Configurations for better performance

Detached Invokers

- Currently used within JBossAS when making most remote invocations (e.g. EJB 2.x calls).
- Detached invoker implementations
  - ✓ JRMPInvoker – JRMP (RMI) Transport
  - ✓ PooledInvoker - Socket Transport
  - ✓ IIOPInvoker - IIOP Transport
  - ✓ HttpInvoker - HTTP Transport

EJB 2.x invocation view



Drawbacks to detached invokers

- Completely independent implementations
  - ✓ No code re-use
    - Requires more time in dev, maintenance, and qa
  - ✓ No common configuration points
    - Configuring common properties, such as connection timeout, different depending on implementation (and not even available on some).
  - ✓ Difficult to customize or extend

Motivation for JBoss Remoting

- Common framework to be used any place a remote invocation needs to be made.
  - ✓ Standard API regardless of transport or data marshalling (along with low learning curve).
  - ✓ Standard configuration for common properties for all transports.
  - ✓ Pluggability and customization at all levels (e.g. custom transport, thread pool, encryption, serialization implementation, socket implementation)

Motivation for JBoss Serialization

- Performance
  - ✓ Smart cloning
  - ✓ Class loading caching
  - ✓ MetaData non Synchronized
  - ✓ Some objects take too long on JavaSerialization
  - ✓ Use of trove (fast HashMaps)
- Features
  - ✓ Ability to serialize object that do NOT implement Serializable interface
    - Only requirement is a default constructor
  - ✓ Support to any JVM 1.4+ JVM.
    - Except JRockit JVM 1.4 (Works fine with JRockit 1.5+)

7

JBoss Remoting

- What is it?
- Features
- How it works
- Sample code
- Configuration

8

JBoss Remoting – What is it?

- Framework for making network based invocations and related services.
- Provides a single API regardless transport protocol or data marshalling used.
- Hides the complexity of transport specific details behind a simple API.
- Highly customizable and extensible.

9

JBoss Remoting - Features

- Pluggable transports
  - ✓ Comes with Socket, RMI, HTTP, Multiplex, and Servlet (all support SSL).
- Pluggable serialization
  - ✓ Comes with Java and JBoss Serialization
- Pluggable data marshallers
- Automatic discovery
  - ✓ Comes with multicast and JNDI
- Callbacks

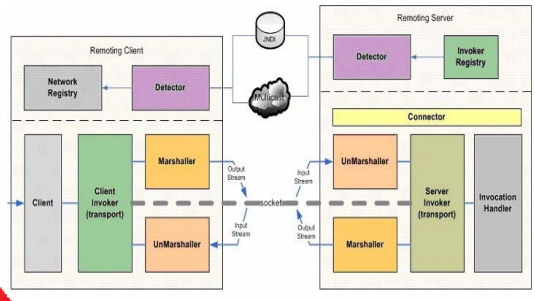
10

JBoss Remoting – Features

- Synchronous & Asynchronous invocations
- Remote classloading
- Sending of streams
- Failover of invocations
- Connection failure notification
- Data compression

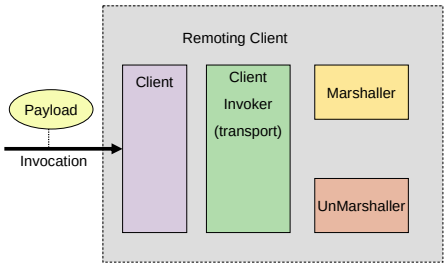
11

Architecture



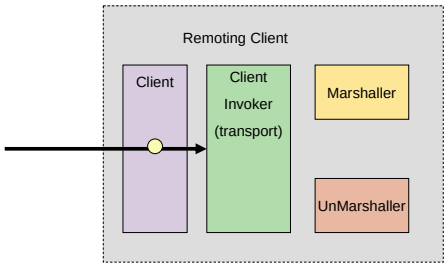
12

JBoss Remoting – How it works



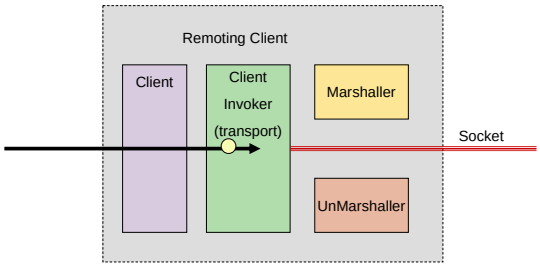
13

JBoss Remoting – How it works



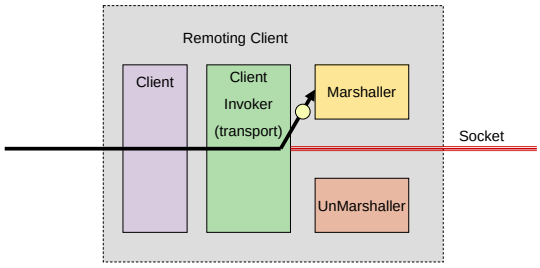
14

JBoss Remoting – How it works



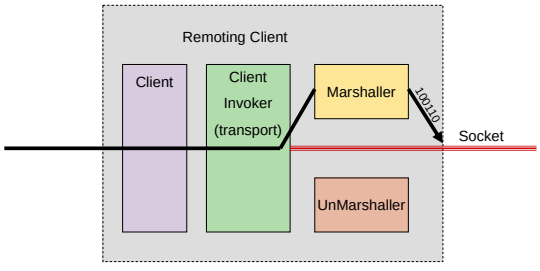
15

JBoss Remoting – How it works



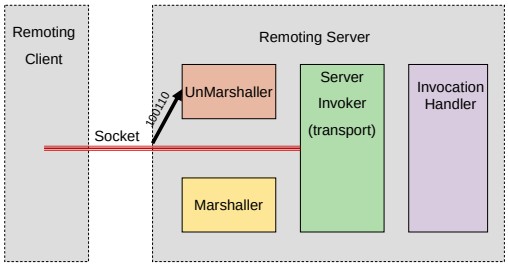
16

JBoss Remoting – How it works



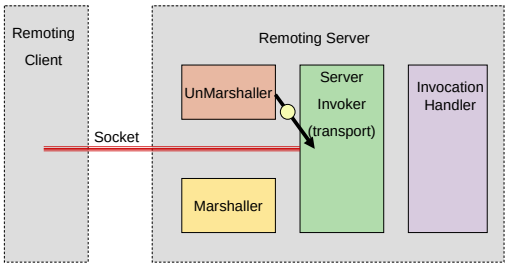
17

JBoss Remoting – How it works



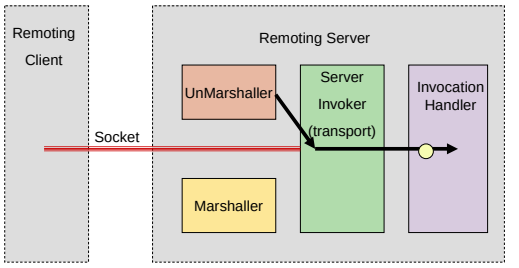
18

JBoss Remoting – How it works



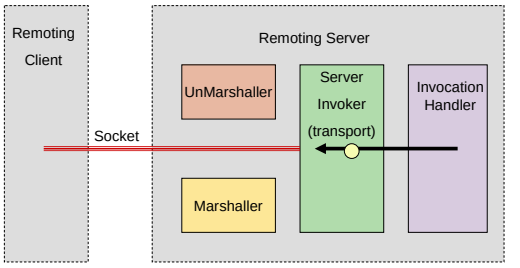
19

JBoss Remoting – How it works



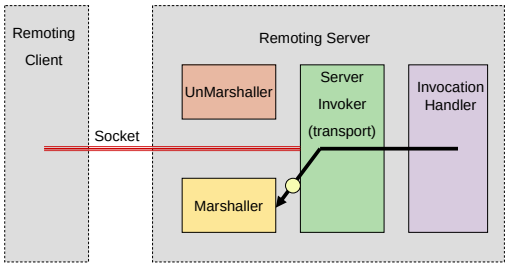
20

JBoss Remoting – How it works



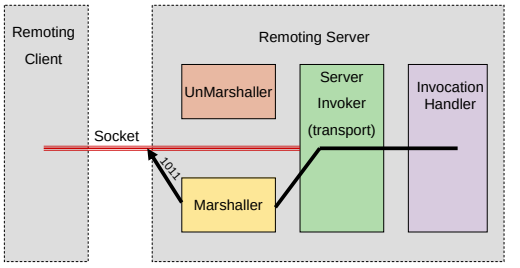
21

JBoss Remoting – How it works



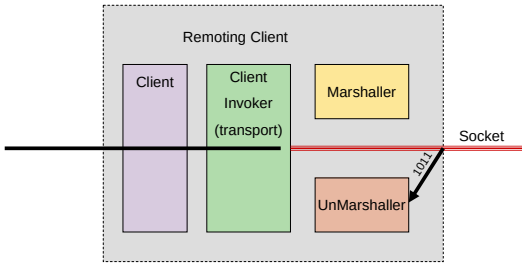
22

JBoss Remoting – How it works



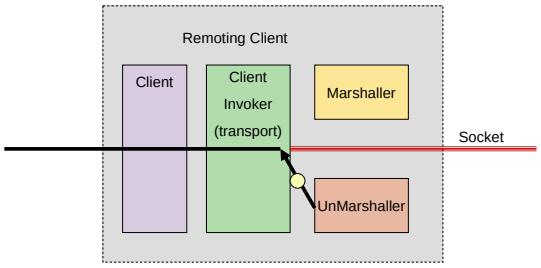
23

JBoss Remoting – How it works



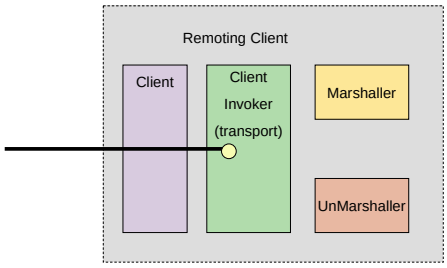
24

JBoss Remoting – How it works



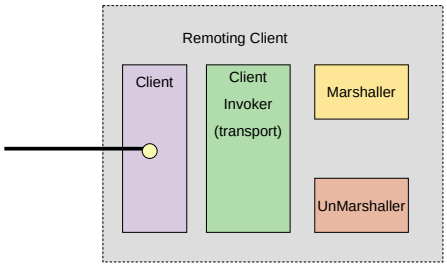
25

JBoss Remoting – How it works



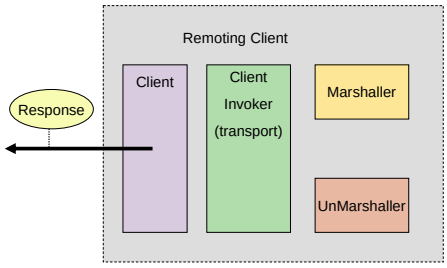
26

JBoss Remoting – How it works



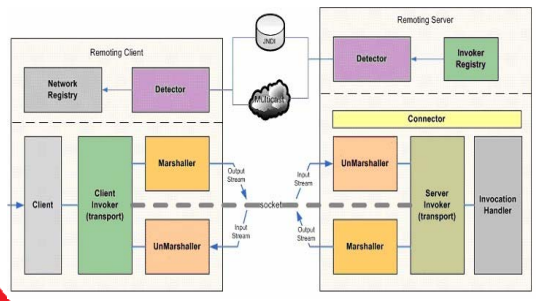
27

JBoss Remoting – How it works



28

Architecture



29

JBoss Remoting – sample code

- Simple server code for starting a remoting server.

```
String locatorURI = "socket://localhost:5400";
InvokerLocator locator = new InvokerLocator(locatorURI);
Connector connector = new Connector(locator);
connector.create();

SampleInvocationHandler invocationHandler = new SampleInvocationHandler();
connector.addInvocationHandler("sample", invocationHandler);

connector.start();
```

30

JBoss Remoting – sample code

- Implementation for the server handler

```
public static class SampleInvocationHandler implements ServerInvocationHandler
{
    public Object invoke(InvocationRequest invocation) throws Throwable
    {
       System.out.println("Invocation request is: " + invocation.getParameter());
       return "This is the response";
    }

    public void addListener(InvokerCallbackHandler callbackHandler)  { ... }
    public void removeListener(InvokerCallbackHandler callbackHandler) { ... }
    public void setMBeanServer(MBeanServer server) { ... }
    public void setInvoker(ServerInvoker invoker)  { ... }

}
```


JBoss Remoting – sample code

- Remoting client code

```
String locatorURI = "socket://localhost:5400";
InvokerLocator locator = new InvokerLocator(locatorURI);

Client remotingClient = new Client(locator);
Object response = remotingClient.invoke("Do something");

System.out.println("Invocation response: " + response);
```


JBoss Remoting – Transporter

- Transporters take a plain old java object (POJO) and expose a remote proxy to it via JBoss Remoting
- Dynamic proxies and reflection are used to make the typed method calls on that target POJO.
- Provides built in failover of call (even over different transport types)

JBoss Remoting – Transporter

- Plain old java interface and class

```
public interface DateProcessor
{
  public String formatDate(Date dateToConvert);
}

public class DateProcessorImpl implements DateProcessor
{
  public String formatDate(Date dateToConvert)
  {
    DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.MEDIUM);
    return dateFormat.format(dateToConvert);
  }
}
```


JBoss Remoting – Transporter

- Transporter server

```
public class Server
{
  public static void main(String[] args) throws Exception
  {
    TransporterServer server = TransporterServer.createTransporterServer("socket://localhost:5400",
            new DateProcessorImpl(),
            DateProcessor.class.getName());
    Thread.sleep(10000);
    server.stop();
  }
}
```


JBoss Remoting – Transporter

- Transporter client

```
public class Client
{
  public static void main(String[] args) throws Exception
  {
    String locatorURI = "socket://localhost:5400";
    DateProcessor dateProcessor = (DateProcessor) TransporterClient.createTransporterClient(locatorURI,
            DateProcessor.class);
    String formattedDate = dateProcessor.formatDate(new Date());
    System.out.println("Current date: " + formattedDate);
  }
}
```

Output from running Client is "Current date: Jun 7, 2006"

JBoss Remoting – configuration

```
String locatorURI = "socket://localhost:5400/?timeout=3000&serializationtype=jboss";
InvokerLocator locator = new InvokerLocator(locatorURI);
Connector connector = new Connector(locator);
connector.create();

SampleInvocationHandler invocationHandler = new SampleInvocationHandler();
connector.addInvocationHandler("sample", invocationHandler);

connector.start();
```


JBoss Remoting – configuration

```
String locatorURI = "socket://localhost:5400";
InvokerLocator locator = new InvokerLocator(locatorURI);
Map config = new HashMap();
config.put("timeout", "3000");
config.put("serializationtype", "jboss");
Connector connector = new Connector(locator, config);
connector.create();

SampleInvocationHandler invocationHandler = new SampleInvocationHandler();
connector.addInvocationHandler("sample", invocationHandler);

connector.start();
```


JBoss Remoting – configuration

```
String locatorURI = "socket://localhost:5400/?timeout=3000&serializationtype=jboss";
InvokerLocator locator = new InvokerLocator(locatorURI);

Client remotingClient = new Client(locator);
Object response = remotingClient.invoke("Do something");

System.out.println("Invocation response: " + response);
```


JBoss Remoting – configuration

```
String locatorURI = "socket://localhost:5400";
InvokerLocator locator = new InvokerLocator(locatorURI);

Map config = new HashMap();
config.put("timeout", "3000");
config.put("serializationtype", "jboss");

Client remotingClient = new Client(locator, config);
Object response = remotingClient.invoke("Do something");

System.out.println("Invocation response: " + response);
```


JBoss Remoting - configuration

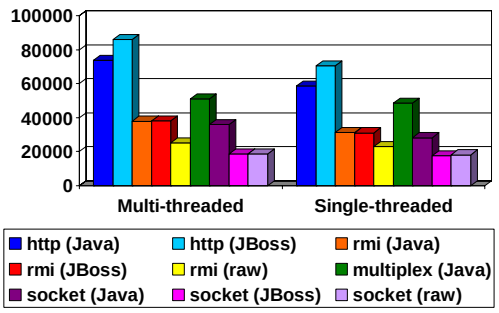
```
<mbean code="org.jboss.remoting.transport.Connector"
    name="jboss.remoting:service=Connector,transport=Socket"
    display-name="Socket transport Connector">
    <attribute name="InvokerLocator">
      <![CDATA[socket://foobar.com:8084]]>
    </attribute>
    <attribute name="Configuration">
      <config>
        <handlers>
          <handler subsystem="mock">
            org.jboss.remoting.mock.MockServerInvocationHandler
          </handler>
        </handlers>
      </config>
    </attribute>
</mbean>
```


JBoss Remoting - configuration

```
<mbean code="org.jboss.remoting.transport.Connector"
    name="jboss.remoting:service=Connector,transport=Socket"
    display-name="Socket transport Connector">
    <attribute name="Configuration">
      <config>
        <invoker transport="socket">
          <attribute name="serverBindAddress">foobar.com</attribute>
          <attribute name="serverBindPort">8084</attribute>
          <attribute name="serializationtype">jboss</attribute>
          <attribute name="socketTimeout"
                     isParam="true">60000</attribute>
        </invoker>
        <handlers>
          <handler subsystem="mock">
            org.jboss.remoting.mock.MockServerInvocationHandler
          </handler>
        ...
```

socket://foobar.com:8084/?serializationtype=jboss&socketTimeout=60000

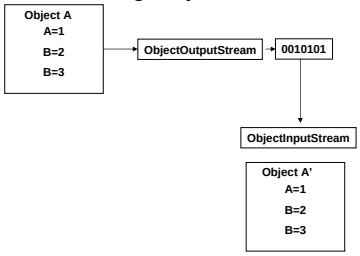
JBoss Remoting - performance



43

JBossSerialization

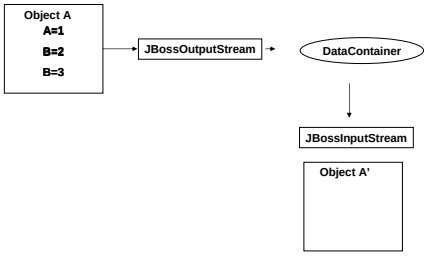
- Cloning objects with serialization



44

JBossSerialization

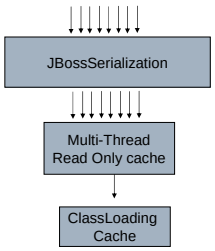
- Smart Cloning



45

JBoss Serialization

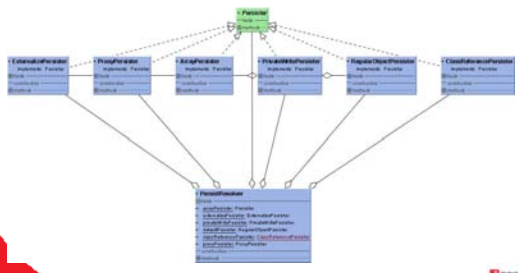
- MetaData / Reflection



46

JBossSerialization

- Implementation



47

JBossSerialization

- Real DataOutputs
  - ✓ ByteOutputs (on regular streams)
    - DirectDataOutput
    - DirectDataInput
  - ✓ DataContainer (over smart cloning)
    - DataContainDataOutput
    - DataContainerDataInput

48

JBoss Serialization

- Performance
  - ✓ Smart cloning
  - ✓ Class loading caching
  - ✓ MetaData non Synchronized
  - ✓ Some objects take too long on JavaSerialization
  - ✓ Use of trove (fast HashMaps)
- Features
  - ✓ Ability to serialize object that do NOT implement Serializable interface
    - Only requirement is a default constructor
  - ✓ Support to any JVM 1.4+ JVM.
    - Except JRockit JVM 1.4 (Works fine with JRockit 1.5+)

JBoss Remoting & Serialization

- Where are they being used now?
  - ✓ EJB 2.x – JBossAS 4.0.4 (using UnifiedInvoker) and will be default in JBossAS 5.
  - ✓ EJB3
  - ✓ JMS – JBoss Messaging
  - ✓ Web services - JBossWS
- Where will they be used?
  - ✓ JNDI
  - ✓ JMX Remoting

General Configuration

- Use JDK 5 – automatic tuning and garbage collection much better
- Use Parallel garbage collection types (better throughput)
- Increase page sizes (typical defaults to 4k, which is tiny).
- Reduce Thread stack size – 256K
- Turn off full garbage collection

General Configuration

- Example JDK options:

-server -Xms2650m -Xmx2650m

-Xss128k -XX:+DisableExplicitGC

-XX:+UseParallelGC

-XX:ParallelGCThreads=8

-XX:LargePageSizeInBytes=256m

Summary

- JBoss Remoting & JBoss Serialization provides:
  - ✓ Speed
  - ✓ Ease of use
  - ✓ Consistency
  - ✓ Extensibility
  - ✓ Configuration
- Q&A