



# JBossRemoting

---

## *Architecture Overview*



# What is JBoss Remoting

---

- **JBoss Remoting is a framework with a single, simple API for making network based invocations and other network related services.**
- Features of JBoss Remoting:
  - ✓ **Server identification** – a simple String identifier which allows for remoting servers to be identified and called upon.
  - ✓ **Pluggable transports** – can use different protocol transports, such as socket, rmi, http, etc., via the same remoting API.
  - ✓ **Pluggable data marshallers** – can use different data marshallers and unmarshallers to convert the invocation payloads into desired data format for wire transfer.
  - ✓ **Automatic discovery** – can detect remoting servers as they come on and off line.
  - ✓ **Server grouping** – ability to group servers by logical domains, so only communicate with servers within specified domains.

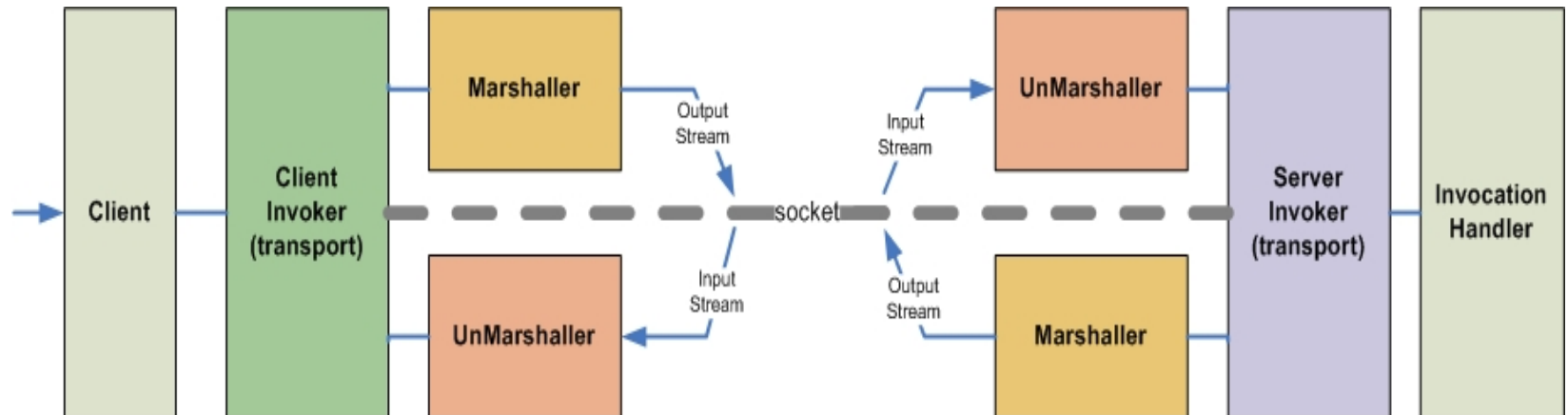


# What is JBoss Remoting

---

- Features of JBoss Remoting (cont.):
  - ✓ **Callbacks** – can receive server callbacks via push and pull models. Pull model allows for persistent stores and memory management.
  - ✓ **Asynchronous calls** – can make asynchronous, or one way, calls to server.
  - ✓ **Local invocation** – if making an invocation on a remoting server that is within the same process space, remoting will automatically make this call by reference, to improve performance.
  - ✓ **Remote classloading** – allows for classes, such as custom marshallers, that do not exist within client to be loaded from server.
  - ✓ **Sending of streams** – allows for clients to send input streams to server, which can be read on demand on the server.

# Core Architecture



- Client – the external API access point for client code.
- Client/Server Invoker – protocol specific implementation. For example, SocketClientInvoker and SocketServerInvoker.
- Marshaller/UnMarshaller – receives the streams from the invoker converting wire data to Object form and vice versa.
- Invocation Handler – end target interface implemented by user that receives the invocation from the client.



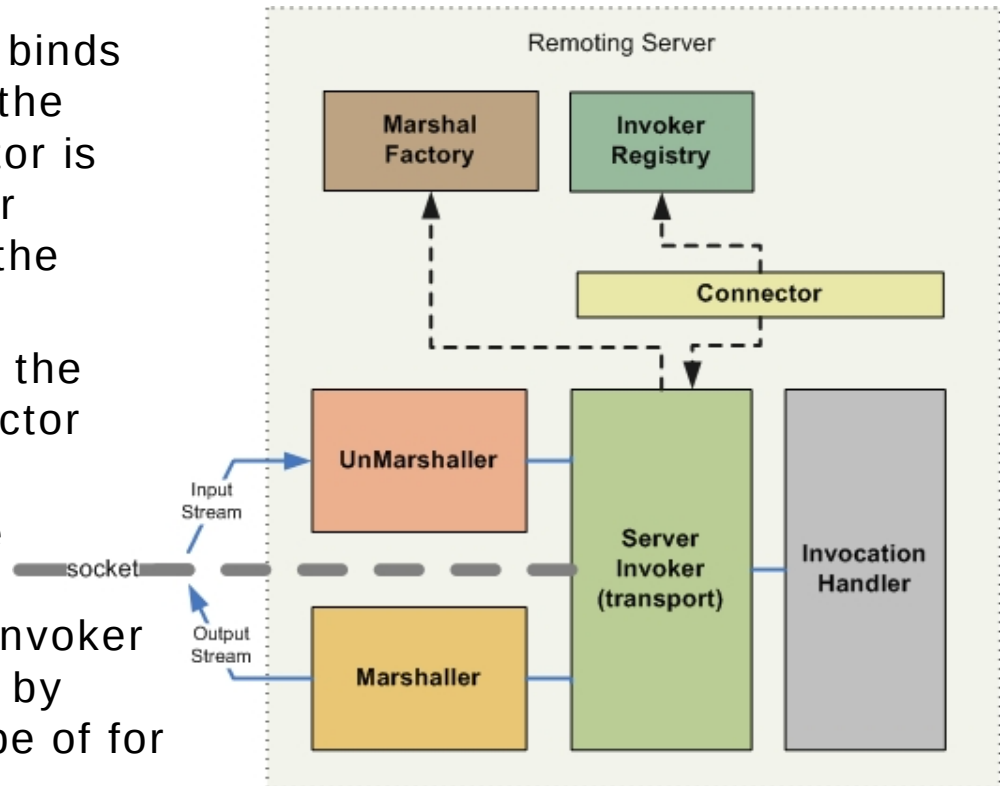
# Architecture - InvokerLocator

---

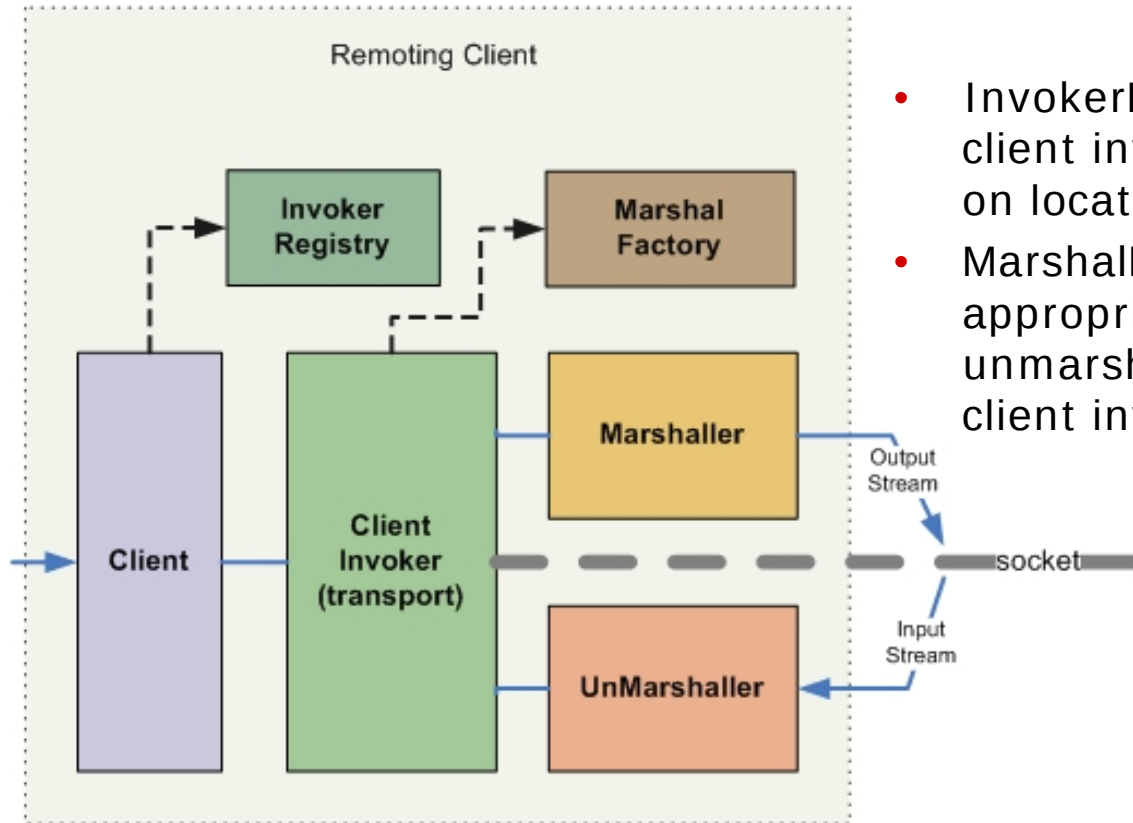
- InvokerLocator – is the Object that uniquely identifies a remoting server. The invoker locator can be constructed using a simple uri String. The InvokerLocator is what is used to construct both the remoting client and the remoting server; nothing else is needed.
- Examples of InvokerLocator uri strings:
  - ✓ `socket://test.somedomain.com:5400`
  - ✓ <http://test.somedomain.com:5401>
  - ✓ `rmi://test.somedomain.com:5402`
  - ✓ `socket://test.somedomain.com:8084/?enableTcpNoDelay=false&clientMaxPoolSize=30`
  - ✓ `socket://${jboss.bind.address}:4446/?datatype=invocation`

# Architecture – server side

- Connector – the service that binds the invocation handler(s) to the server invoker. The Connector is used as the external point for configuration and control of the remoting server.
- InvokerRegistry – will create the server invoker for the Connector based on locator uri.
- MarshalFactory – creates the appropriate marshaller and unmarshaller for the server invoker based on data type specified by locator or by default data type of for the server invoker.

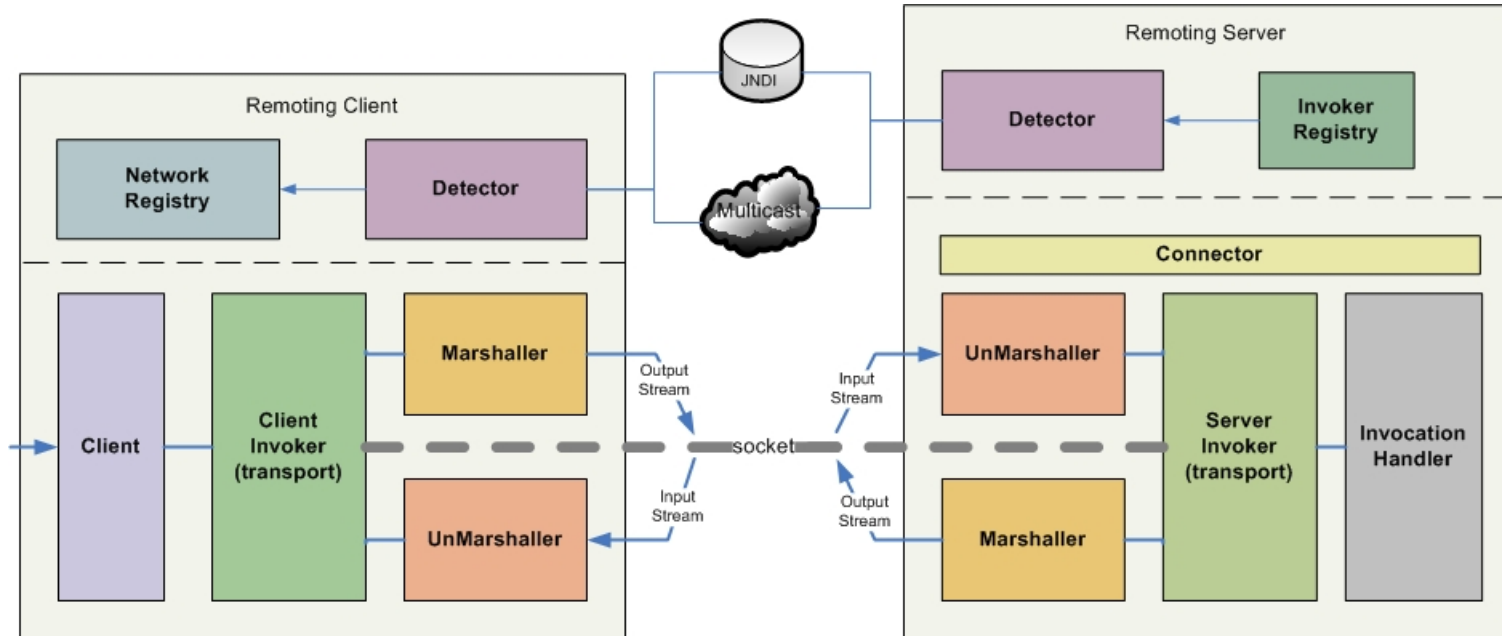


# Architecture – client side



- InvokerRegistry – will create the client invoker for the Client based on locator uri.
- MarshalFactory – will created the appropriate marshaller and unmarshaller to be used by the client invoker.

# Architecture - detection



- Detector - will receive (multicast) or poll for (JNDI) detection messages and publish changes to topology to the NetworkRegistry
- NetworkRegistry – contains listing of all remoting servers available.
- Detector - will broadcast (multicast) or bind (JNDI) detection messages based on server invokers registered within InvokerRegistry. Detection message will contain locator uri and supported subsystems.



# Configuration

---

- Configuration for remoting can be handled either programmatically or via JBoss service xml (when deployed within JBoss AS container).
- The full documentation for configuration can be found within the remoting User Guide or on the Wiki (see links on <http://www.jboss.org/products/remoting>).



# JBoss Remoting information

---

- Project page - <http://www.jboss.org/products/remoting>
  - ✓ User Guide
  - ✓ Wiki
  - ✓ Demo
  - ✓ Forum
- *Introducing JBoss Remoting*
  - ✓ published at OnJava.com
  - ✓ <http://www.onjava.com/pub/a/onjava/2005/02/23/remoting.html>



# Sample Code

---

Sample code for starting remoting server.

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

SampleInvocationHandler invocationHandler = new SampleInvocationHandler();
// first parameter is sub-system name. can be any String value.
connector.addInvocationHandler("sample", invocationHandler);

connector.start();
```



# Sample Code

---

Implementation for the ServerInvocationHandler.

```
public static class SampleInvocationHandler implements ServerInvocationHandler
{
    public Object invoke(InvocationRequest invocation) throws Throwable
    {
        // Print out the invocation request
        System.out.println("Invocation request is: " + invocation.getParameter());
        // Just going to return static string as this is just simple example code.
        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) { ... }
}
```



# Sample Code

---

Remoting client code.

```
String locatorURI = "socket://localhost:5400";  
InvokerLocator locator = new InvokerLocator(locatorURI);  
System.out.println("Calling remoting server with locator uri of: " + locatorURI);  
  
Client remotingClient = new Client(locator);  
Object response = remotingClient.invoke("Do something");  
  
System.out.println("Invocation response: " + response);
```