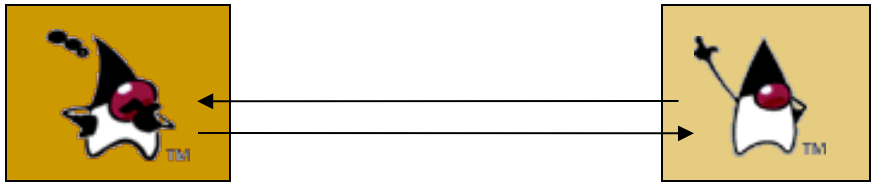
Sistemas Distribuidos.
Sesión de Prácticas: Java RMI



Conceptos Básicos
Recolección de Basura Distribuida
Carga de Clases Dinámica
Retrollamadas (*Callbacks*)

Java RMI

Utiliza internamente *sockets* sobre TCP/IP

Objetos residentes y asociados a las máquinas virtuales

Otra opción:

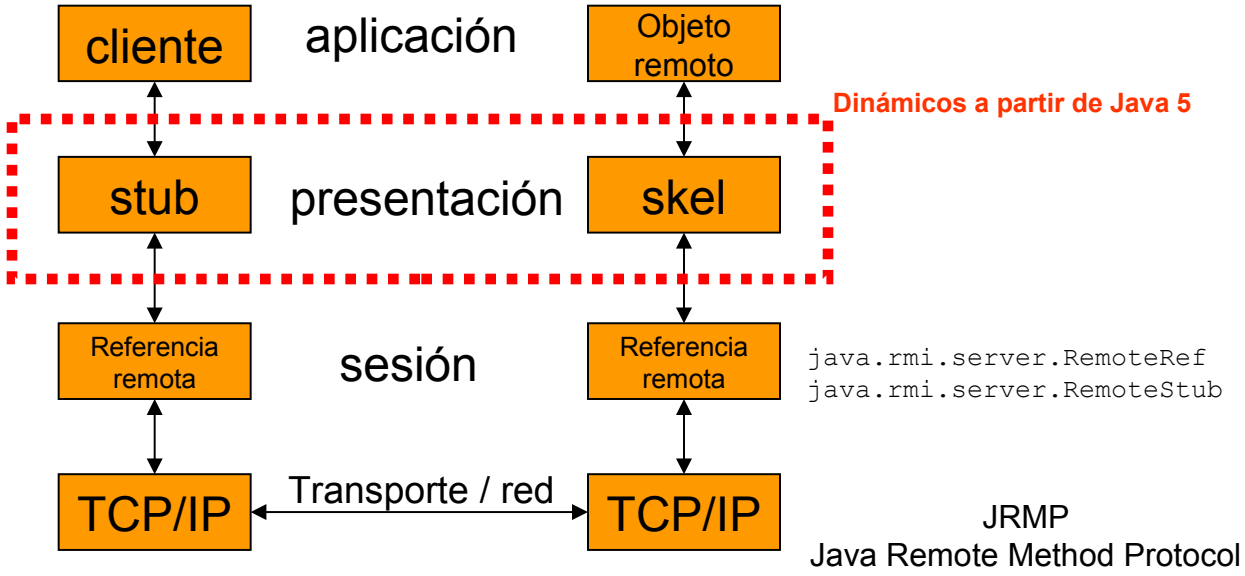
Java Message Service (JMS) con mensajes viajando <u>asíncronamente</u>

Capas:

Stubs / skeletons

Referencia remota

Transporte



Políticas de Seguridad

Concesión de permisos

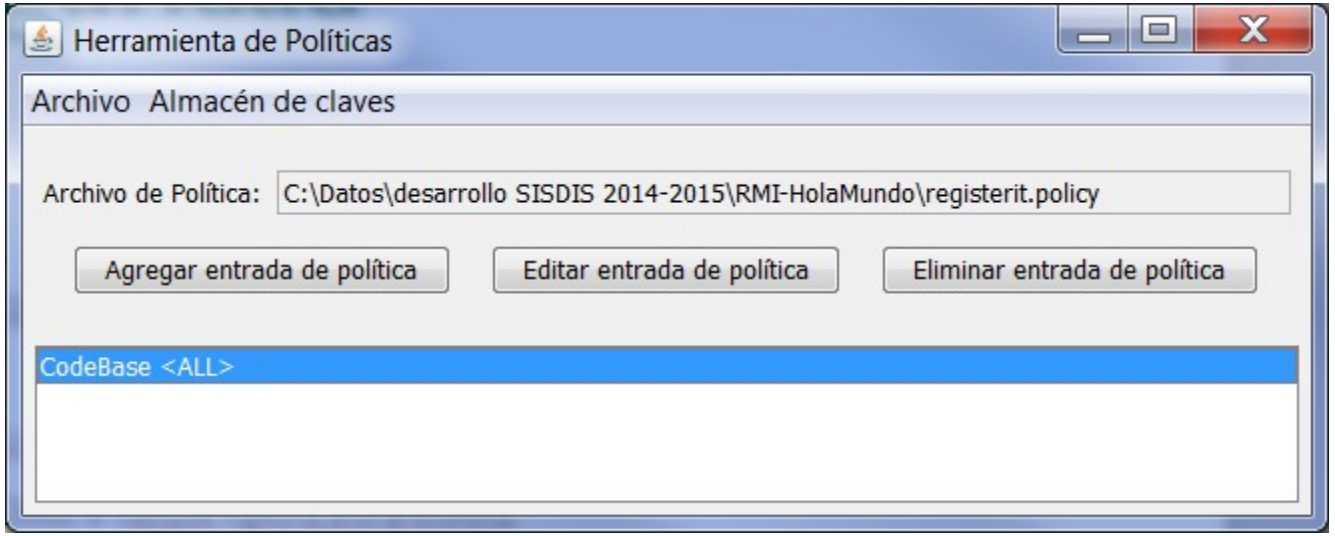
- A través de propiedades

  Ej: `java -Djava.security.policy=<policy_file> MiClase`

Archivo por defecto

`%JAVA_HOME%\jre\lib\Security\java.policy`

- Herramienta de edición `%JAVA_HOME%\bin\policytool`



Desarrollo de Aplicaciones Java RMI

Pasos (creando el proyecto con arquetipo `maven-archetype-quickstart`):

- Definir interfaz remota pública, extiende a `java.rmi.Remote`
  - Todo método que se invoque en remoto debe lanzar `java.rmi.RemoteException`
  - Si tenemos objetos remotos como parámetros o tipo de retorno, **deben ser interfaces** no clases de implementación.
- Implementar la interfaz remota en una clase de implementación
  - Hereda de `java.rmi.server.UnicastRemoteObject` o `Activatable`
  - O bien se exportará con `UnicastRemoteObject.exportObject`
- Implementar el cliente
  - Siempre interactuando con la interfaz Java
- Generar stubs (proxys de cliente) y skeletons (entidades de servidor)
  - `rmic -v1.2`
    - Nota: -v1.2 suprime la creación de skeletons (se usa reflectividad !!!)
  - **A partir de Java 1.5 este paso se puede omitir**
- Iniciar registro (`rmiregistry`)
- Ejecutar el servidor registrando el objeto remoto
- Ejecutar el cliente

Ejemplo: Servidor / Cliente (HolaMundo)

Fichero `register.policy`

Ficheros `.bat`

Lanzar el enlazador (`registro.bat`)

```
rmiregistry
```

Lanzar el servidor (`servidor.bat`)

```
java -Djava.security.policy=registerit.policy
  ejemplo.holamundo.Servidor
```

Lanzar el cliente (`cliente.bat`)

```
java -Djava.security.policy=registerit.policy
  ejemplo.holamundo.Cliente %1
```

- Nota: por el momento, la carga de clases se hace desde el CLASSPATH local por lo que será necesario que apunte a donde están las clases **en los tres casos (registro, cliente y servidor)**
  - Ej: `SET CLASSPATH=.\target\classes;%CLASSPATH%`

Recolección de Memoria Distribuida (DGC)

Tiempo de alquiler controlado por la propiedad de sistema

Propiedad	MVJ	Descripción	Valor por defecto
`java.rmi.dgc.leaseValue`	servidor	Duración estándar para las concesiones	10 minutos (600000)
`sun.rmi.dgc.server.gcInterval`	servidor	Intervalo de chequeo para comprobar la consecuencia de operaciones `clean` y concesiones caducadas, determinando si `unreferenced` puede ser llamado	1 minuto (60000)
`sun.rmi.dgc.checkInterval`	servidor	Intervalo para comprobar caducidades	5 minutos (300000) Recomendado a la mitad de leaseValue
`sun.rmi.dgc.client.gcInterval`	cliente	Intervalo de chequeo para comprobar que el stub cliente no es referenciado en local y por lo tanto, se puede invocar a `clean`	1 minuto (60000)
`sun.rmi.dgc.cleanInterval`	cliente	Tiempo de espera en reintentar una operación `clean` si hay un fallo	3 minutos (180000)

Ejemplo: Modificación `Servidor` con DGC

Fichero `registerit.policy`

Ficheros `.bat`

Lanzar el enlazador (`registro.bat`)

```
rmiregistry
```

Lanzar el servidor (`servidor.bat`)

```
java -Dsun.rmi.dgc.checkInterval=500 -Djava.rmi.dgc.leaseValue=1000
   Djava.security.policy=registerit.policy ejemplo.dgc.Servidor
```

Lanzar el cliente (`cliente.bat`)

```
java -Djava.security.policy=registerit.policy ejemplo.dgc.Cliente %1
```

- Nota: por el momento, la carga de clases se hace desde el CLASSPATH local por lo que será necesario que apunte a donde están las clases **en los tres casos (registro, cliente y servidor)**
  - Ej: `SET CLASSPATH=.\target\classes;%CLASSPATH%`

Carga de Clases Dinámica

Propiedades de sistema

- `java.rmi.server.codebase`
  - localización `file://, http:// o ftp://`
- `java.rmi.server.useCodebaseOnly`
  - valor true o false, sólo permite la carga desde el `classpath` local y el `codebase` definido

Configuraciones:

Cerrada: en cada máquina virtual se tienen todas las clases en local utiliza el CLASSPATH local

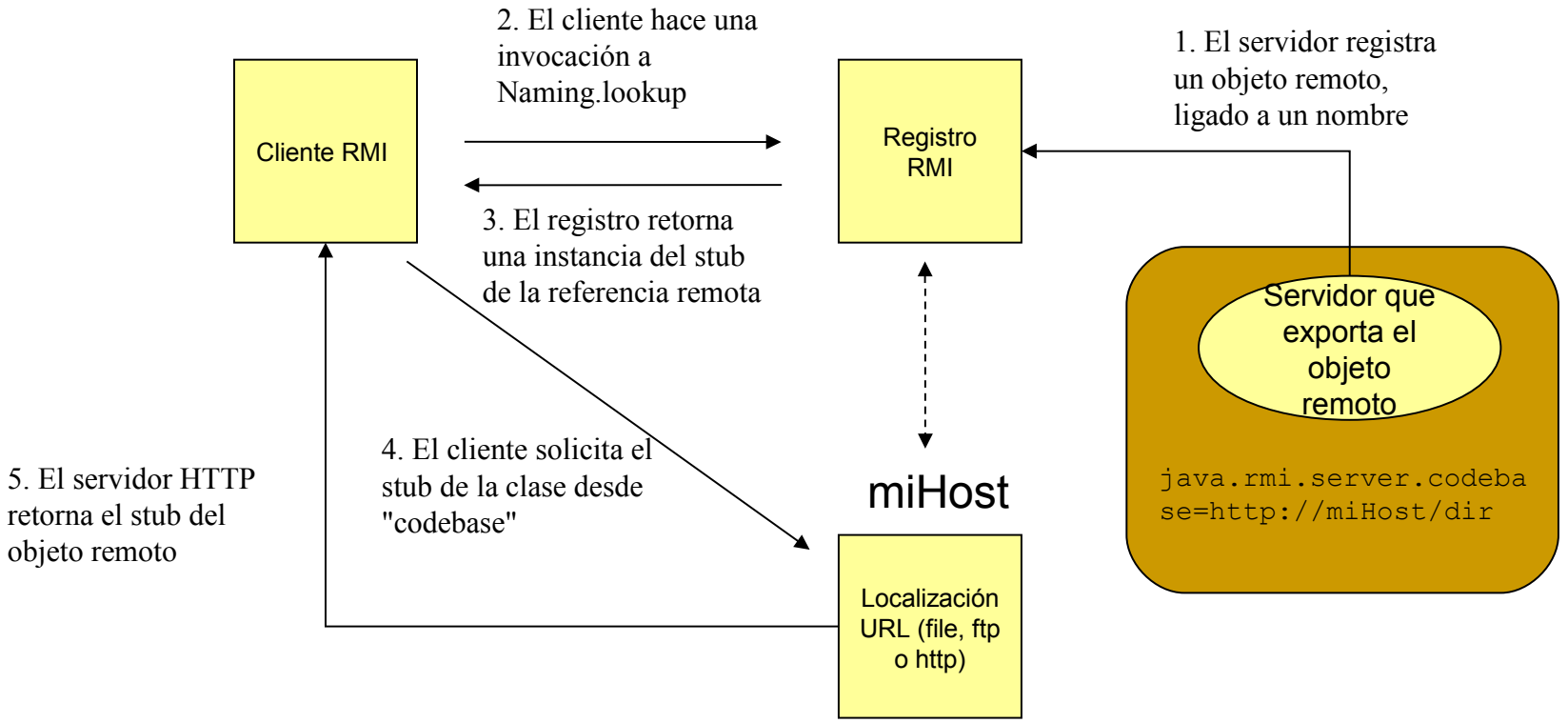
Dinámica lado cliente: algunas clases del cliente en local y otras en remoto

Dinámica lado servidor: algunas clases del servidor en local y otras en remoto

Cliente de autoarranque: todo el código del cliente se carga desde el `codebase` con un programa

Servidor de autoarranque: todo el código del servidor se carga desde el `codebase` con un programa

Carga de Clases Dinámica (II)



Carga de Clases Dinámica (III)

Clases:

- `java.rmi.RMISecurityManager`
- `java.rmi.RMIClassLoader`
  - método `public static Class loadClass(String codebase, String name)`
- El sistema RMI sólo puede descargar clases de una ubicación remota si se ha instanciado un gestor de seguridad `RMISecurityManager`

Modificación al ejemplo:

- Cliente de autoarranque: `ClienteDinamico`
- Servidor de autoarranque: `ServidorDinamico`

Ejemplo: Modificación `Servidor / Cliente (HolaMundo)` con autoarranque en cliente y servidor (I)

Fichero `registerit.policy`

- Generar un proyecto clásico con Maven (arquetipo **`maven-archetype-quickstart)`** con el código del proyecto, de nombre `RMI-Dinamico`.
- Generar una aplicación web con Maven (arquetipo **`webapp-javaee7`**) de nombre `RMI-Dinamico-Web`
- Modificar el `pom.xml` del proyecto clásico para:
  - Copiamos todas las clases de cliente y servidor a la web con una tarea ANT embebida (plugin `maven-antrun-plugin`).
  - Ajustamos el empaquetado con el `jar`, excluyendo los ficheros que se descargan de la web.
  - Como resultado tendremos:
    - Un fichero `RMI-Dinamico-0.0.1-SNAPSHOT.jar` con las clases de arranque
    - Una aplicación *web* con las clases a descargar dinámicamente

Ejemplo: Modificación `Servidor / Cliente (HolaMundo)` con autoarranque en cliente y servidor (II)

Modificaciones al `pom.xml`

```
<build>
	<plugins>
		<plugin>
			<artifactId>maven-antrun-plugin</artifactId>
			<executions>
				<execution>
					<phase>package</phase>
					<configuration>
						<tasks>
							<copy todir="../RMI-Dinamico-Web/src/main/webapp">
								<fileset dir="./target/classes">
									<exclude name="**/*Dinamico.class"/>
								</fileset>
							</copy>
						</tasks>
					</configuration>
					<goals>
						<goal>run</goal>
					</goals>
				</execution>
			</executions>
		</plugin>
		<plugin>
			<artifactId>maven-jar-plugin</artifactId>
			<version>2.6</version>
			<configuration>
				<includes>
					<include>**/*Dinamico.*</include>
				</includes>
			</configuration>
		</plugin>
	</plugins>
</build>
```

Plugin

Phase

Ant Task

Excluir

Copia los `.class` del proyecto clásico al proyecto web

Modifica el empaquetado excluyendo ficheros

Ejemplo: Modificación `Servidor / Cliente (HolaMundo)` con autoarranque en cliente y servidor (III)

Ficheros .bat

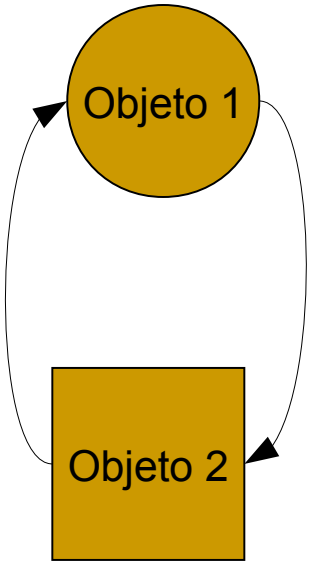
- Lanzar el enlazador (`registro.bat`)
  - `rmiregistry -Djava.rmi.server.codebase=<URL_APP_WEB>`
- Lanzar el servidor (`servidor.bat`)
  - `java -Djava.security.policy=registerit.policy -Djava.rmi.server.codebase=<URL_APP_WEB> -cp .\target\RMI-Dinamico-0.0.1-SNAPSHOT.jar es.ubu.lsi.ServidorDinamico`
- Lanzar el cliente (`cliente.bat`)
  - `java -Djava.security.policy=registerit.policy -Djava.rmi.server.codebase=<URL_APP_WEB> -cp .\target\RMI-Dinamico-0.0.1-SNAPSHOT.jar es.ubu.lsi.ClienteDinamico`

  - En el ejemplo `<URL_APP_WEB> = http://localhost:8080/RMI-Dinamico-Web/`
    - Se debe lanzar el proyecto RMI-Dinamico-Web en un servidor web como Tomcat
    - Se purede realizar todo desde Eclipse, con el servidor integrado.

Retrollamadas Remotas

- ¡El cliente también puede ser objeto remoto!
  - Sin registro previo
- El servidor necesita comunicarse con los clientes (invocar a métodos en el cliente).
  - Ej: datos de progreso, notificación administrativa
  - Ej: aplicación Chat

Paso 1. El objeto 2 invoca un método remoto en el objeto1 y se pasa a sí mismo como un parámetro para ese método



Paso 2. El objeto 1 utiliza la referencia que ha obtenido en el paso 1 e invoca un método en esa referencia

Ejemplo: Retrollamadas remotas (Remote callbacks) (I)

Ficheros fuente

- Nota: el `Applet` necesita exportarse como objeto remoto. La línea `UnicastRemoteObject.exportObject(this)` lo permite, ya que no hay herencia múltiple en java.

- Ojo, para la correcta generación de stubs:
  - a la hora de utilizar el método `UnicastRemoteObject.exportObject` es fundamental utilizar la versión sobrecargada con un segundo argumento indicando el puerto (0 para indicar el puerto anónimo).
    - Ej: `UnicastRemoteObject.exportObject(obj, 0);`
- Si se utiliza la versión con un único argumento: es necesario generar los stubs con **rmic** y tenerlos accesibles en el `classpath`.

Fichero `registerit.policy`

Ejemplo: Retrollamadas remotas (Remote callbacks) (II)

Ficheros `.bat`

- Lanzar el enlazador (`registro.bat`). Necesita tener accesibles los stubs (previamente `set CLASSPATH=.\target\classes; %CLASSPATH%`).

```
rmiregistry
```

- Lanzar el servidor (`servidor.bat`). Necesita clases `Servidor`, `Cliente` y `Hola`

```
java -Djava.security.policy=registerit.policy -cp
   .\target\classes ejemplo.callback.Servidor
```

- Lanzar el cliente (`cliente.bat`). No necesita la clase `Servidor` pero sí el resto.

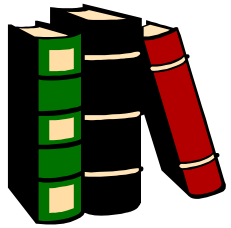
```
appletviewer -J-Djava.security.policy=registerit.policy Applet.html
```

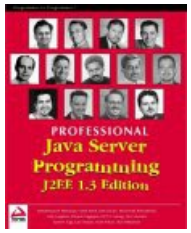
Ejemplo: Retrollamadas remotas (Remote callbacks) (III)

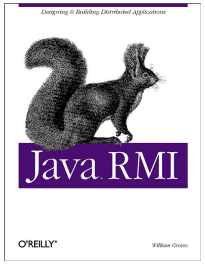
- Cuando se carga el applet se pasa a sí mismo como argumento
- El stub se transporta/carga al servidor
- Se pueden invertir los roles

Resultado

Bibliografía







[Coulouris et al., 2011], Distributed Systems, Concepts and Design
Addison Wesley, (2011) 5th Edition
Chapter 5. Remote Invocation. 5.5 Case study: Java RMI

- http://www.cdk5.net/wp/

[Subrahmanyam et al., 2001] Programación Java Server con J2EE Edición 1.3 (2001) Subrahmanyam et al. Ed. Wrox Programmer to Programmer.
Capítulo 3. Procesamiento distribuido con RMI

[Grosso, 2002] Java RMI (2002). W.Grosso. Ed. O'Reilly

Documentación en línea (2015)

- Remote Method Invocation Home
- Java Remote Method Invocation API
- FAQ Java RMI and Serialization