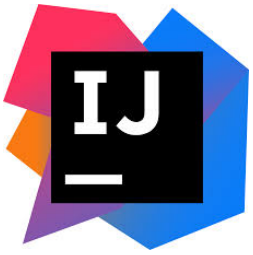
Java RMI
Introdução

Sistemas Distribuídos (PL)

3º ano da Licenciatura em Engenharia Informática

Professores: Ivo Pereira (ivopereira@ufp.edu.pt) e Rui Moreira (rmoreira@ufp.edu.pt)

Ferramentas de desenvolvimento



Java RMI

UNIVERSIDADE FERNANDO PESSOA

- **Remote Procedure Call (RPC)**
  - Permite executar métodos/procedimentos remotamente
  - Sockets
- **Java Remote Method Invocation (RMI)**
  - API em Java para executar métodos remotamente
  - É uma forma de fazer RPC
  - Permite invocações a métodos de objetos através da rede
    - Passar objetos:
      - Por valor (serialização)
      - Por referências remotas a objetos (proxies/stubs)

Java RMI

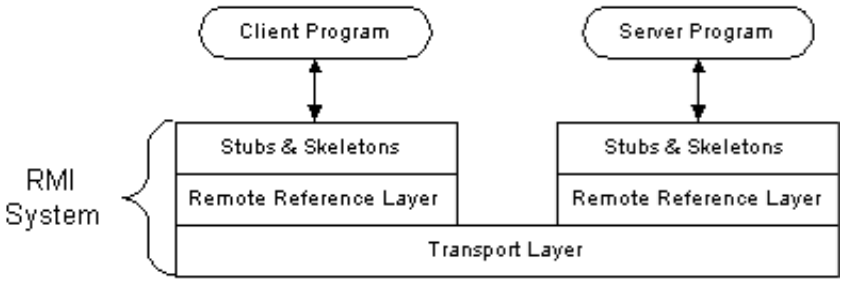
UNIVERSIDADE FERNANDO PESSOA

- **Algumas particulares:**
  - Os parâmetros dos métodos podem ser Java Objects (instâncias de classes Serializable)
  - Esconde o mecanismo subjacente ao transporte de argumentos de métodos e respetivo retorno através da rede
    - Marshalling / Unmarshalling
      - Serialization / Deserialization
  - Faz download das classes (bytecode) em tempo real, através da rede
  - Suporta Garbage Collection distribuído

Java RMI

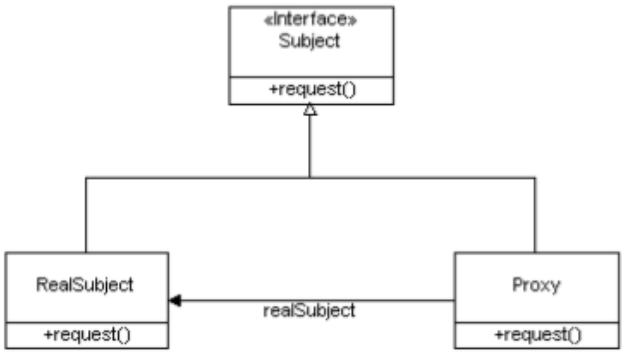
- **Baseado em 3 níveis:**
  - Implementa as camadas de Sessão e Apresentação
    - Camadas 5 e 6 do Modelo OSI
    - Cada uma delas pode ser melhorada/substituída sem afetar o resto do sistema

1. Camada de Stub e Skeleton
2. Camada de Referências Remotas
3. Camada de Transporte



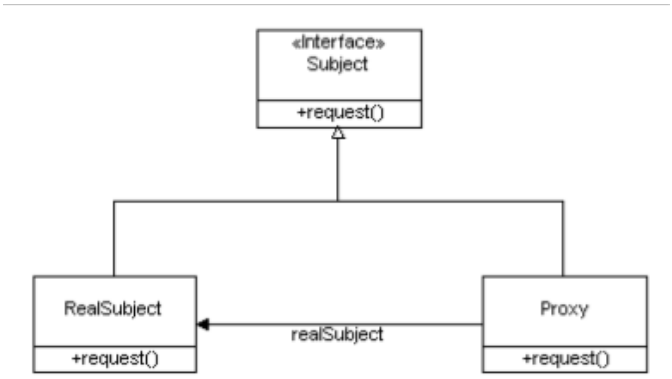
Java RMI

- **Camada de Stub e Skeleton**
  - Stub (Proxy)
    - Intercepta as chamadas aos métodos feita pelas aplicações cliente ao interface
      - **Proxy Design Pattern**
    - Redireciona as chamadas para um serviço RMI remoto



Java RMI

- **Camada de Stub e Skeleton**
  - Skeleton
    - Ajuda a perceber como comunicar com o Stub através do RMI link
    - Lê parâmetros, faz chamadas para objetos Servant e envia os valores de retorno para o Stub
      - Servant - classe que implementa uma interface remota



Java RMI

- **Camada de Stub e Skeleton**
  - O novo protocolo RMI a partir do Java 2 SDK tornou as classes Skeleton obsoletas
  - Hoje em dia o Java RMI usa **Reflection** para fazer a conexão ao objeto de serviço remoto
    - Os programadores já não necessitam de se preocupar com as classes Stub/Skeleton
  - **Reflection**:
    - Capacidade de um processo de examinar, fazer introspeção e modificar a sua própria estrutura e comportamento
    - Padrão de arquitetura que fornece um mecanismo para mudar a estrutura e o comportamento do software de forma dinâmica

```java
import java.lang.reflect.Method;

// Without reflection
Foo foo = new Foo();
foo.hello();

// With reflection
try {
    Object foo = Foo.class.newInstance();

    Method m = foo.getClass().getDeclaredMethod("hello", new Class<?>[0]);
    m.invoke(foo);
} catch (ReflectiveOperationException ignored) {}
```

Java RMI

UNIVERSIDADE FERNANDO PESSOA

- **Camada de Referências Remotas**
  - Interpreta e faz a gestão de referências feitas de clientes aos objetos de serviço remoto
  - Conecta clientes a objetos de serviço remoto
    - Link um-para-um (**unicast**)
    - **Multicast**
      - Um único proxy pode enviar um pedido de método para várias implementações simultaneamente e aceitar a primeira resposta (pode melhorar o tempo de resposta e a disponibilidade)

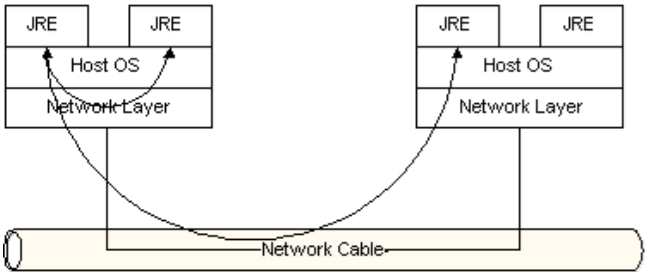
Java RMI

UNIVERSIDADE FERNANDO PESSOA

- **Camada de Referências Remotas**
  - Suporta a ativação de objetos de serviço remoto inativos
    - Objetos remotos **Activatable**
      - Para qualquer chamada de método, a camada RMI determina se o objeto de implementação de serviço remoto está inativo e instancia-o, restaurando o seu estado guardado no sistema de ficheiros
  - Fornece o objeto RemoteRef que representa um link para o objeto de implementação de serviço remoto
  - Objetos Stub usam o método **invoke()** em RemoteRef para encaminhar a chamada do método
  - O objeto RemoteRef entende a semântica de invocação para serviços remotos
    - Marshalling / Unmarshallig.

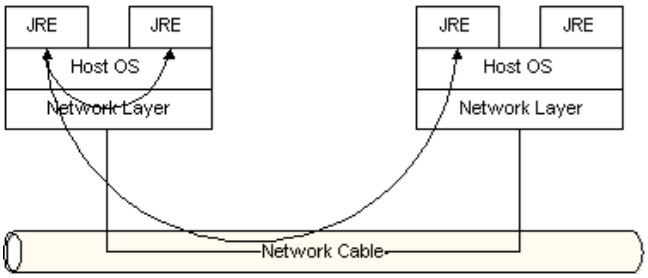
Java RMI

- **Camada de Transporte**
  - Usa conexões de rede entre JVMs baseadas em **TCP / IP**
  - Mesmo se as JVMs estiverem em execução no mesmo host, a conexão é feita através do protocolo TCP / IP
    - Assim, os pares RMI precisam de uma configuração operacional TCP / IP
  - Fornece conectividade básica, bem como algumas estratégias de penetração de firewall



Java RMI

- **Camada de Transporte**
  - O protocolo TCP / IP fornece conexões persistentes com base no **endereço IP** e no **número da porta**:
    - E.g. alberto.ufp.pt:4000
  - Normalmente, utiliza-se um **URL** em vez do endereço IP
  - O cliente e servidor são processos distintos que podem estar a executados na mesma máquina ou em hosts completamente separados

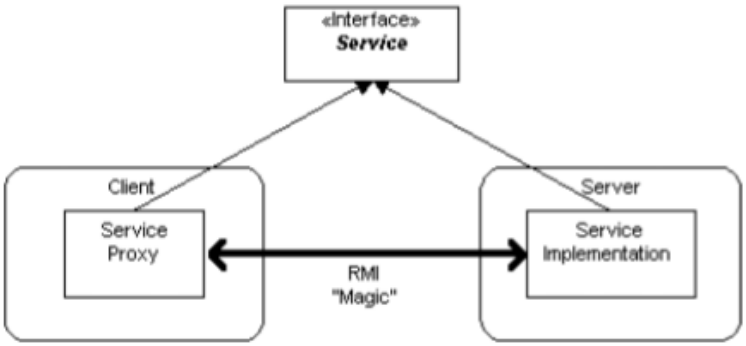


Desenvolvimento de Aplicações RMI

- **Implementação do Proxy Design Pattern:**

1. Definir o Remote interface
   - **Declaração** do serviço
   - *extends Remote*

2. Implementar o Remote interface
   - **Implementação do serviço**
     - Servant
   - Geração automática de Stubs e Skeletons
     - Service proxy

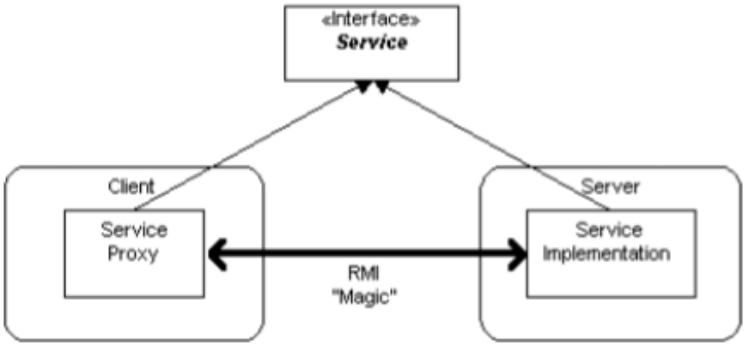


Desenvolvimento de Aplicações RMI

- **Implementação do Proxy Design Pattern:**

3. **Implementar** as aplicações **Cliente** e **Servidor**

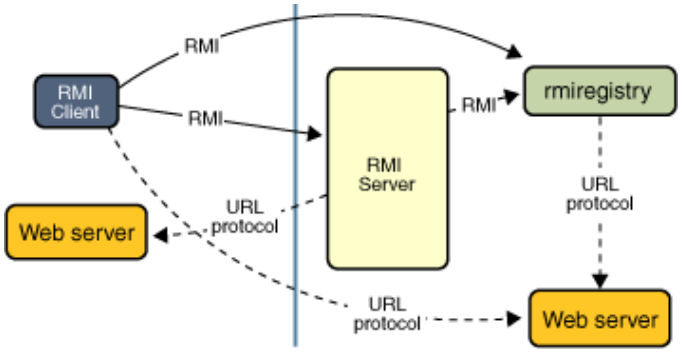
4. Fazer **deploy** ao serviço:
   1. runpython
      - SimpleHTTPServer 8000
   2. runregistry
      - Executar o serviço de registo RMI
   3. runserver
   4. runclient



Deployment de Aplicações RMI

- **Aplicação de objetos distribuídos:**
  1. Servidor
     - Cria Servants e regista-os no Registry (com um nome)
  2. Cliente
     - Procura por servidores no Registry (por nome) e obtém um proxy para os Servants
     - Depois pode usar o serviço através do proxy

  - O rmiregistry (nameservice) permite **descobrir serviços** por nome
  - O servidor HTTP deve ser usado como **base de código** para fazer download do bytecode das classes



Deployment de Aplicações RMI

- **Como é que o Servidor anuncia um Servant (serviço remoto)?**
  - Através de um serviço de naming que é executado num host e porta conhecidos
    - Tipo DNS
    - Mecanismo bootstrap

  - O Servidor exporta um objeto e a camada RMI cria um serviço que está à escuta e fica à espera de chamadas do cliente

  - Depois da exportação, o servidor regista o objeto no RMI Registry com um nome público:
    rmi://<servicehostname>[:<serviceport>]/<servantname>

Deployment de Aplicações RMI

UNIVERSIDADE FERNANDO PESSOA

- **Como é que um Cliente encontra um Servant (serviço remoto)?**
  - Através do mesmo serviço de naming onde o serviço foi registado
  - O Cliente pode usar a classe Naming que oferece métodos estáticos de lookup:
    lookup(service_name); //static method
  - O Cliente pode também usar a classe Registry:
    registry = LocateRegistry.getRegistry(hostaddr, port);
    registry.lookup(service_name);

Deployment de Aplicações RMI

UNIVERSIDADE FERNANDO PESSOA

- **Distribuição automática**
  - Quando o RMI envia o objeto via serialização, incorpora o URL da classe na stream, bem como o objeto
  - O RMI não envia ficheiros de classe juntamente com objetos serializados
    - As classes devem ser descarregadas da base de código
  - A JVM remota carrega ficheiros de classe para os objetos necessários, através da URL incorporada e contatando o servidor http que contém a base de código

Garbage Collection distribuído

UNIVERSIDADE FERNANDO PESSOA

- Não é necessário preocupar-nos com a alocação de memória
  - É diferente de aplicações standalone
- O servidor marca os objetos como "sujos" ou "limpos" sempre que os clientes fazem chamadas ou descartam a referência
- As referências do cliente “vivo” podem fazer com que o tempo limite se esgote, se não atualizarem a conexão antes da expiração
- O interface para o Garbage Collector está escondido na Camada de Stubs e Skeletons

Mais info

UNIVERSIDADE FERNANDO PESSOA

- An Overview of RMI Applications
  - https://docs.oracle.com/javase/tutorial/rmi/overview.html
- Getting Started Using Java™ RMI
  - https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html