MAC 0434/5765 - Sistemas de Middleware Avançados - 2º Semestre de 2003

Primeiro Exercício-Programa: Invocador ICE para o JBoss

Data de Entrega: 8 de outubro de 2003

Neste exercício você se familiarizará com o lado interno de um servidor J2EE, o JBoss.

1 Os Invocadores do JBoss

Vimos em classe que (quase) todos os invocadores (*invokers*) do JBoss tem a mesma estrutura e funcionam de modo semelhante:

- Um invocador é um *service MBean* do JBoss. Além disso, todos os invocadores atualmente existentes são MBeans do tipo *standard*, isto é, `FooInvoker` especifica sua interface de gerenciamento implementando uma interface Java cujo nome é `FooInvokerMBean`.

- Cada invocador é específico para um certo protocolo (`JRMPInvoker`, `HTTPInvoker`, ...) e permite que clientes remotos utilizem esse protocolo para acessar uma interface de invocação genérica, que basicamente oferece o método `invoke()`. A definição precisa dessa interface depende do protocolo que os clientes remotos utilizam para acessá-la. No caso do protocolo JRMP, a interface de invocação genérica é definida da seguinte maneira:

```
package org.jboss.invocation;

public interface Invoker extends javax.rmi.Remote
{
    String getServerHostName() throws Exception;
    Object invoke(Invocation invocation) throws Exception;
}
```

  O `JRMPInvoker` implementa essa interface e portanto disponibiliza o método `invoke()` para clientes RMI/JRMP. Outros invocadores podem implementar interfaces diferentes dessa, mas com o mesmo objetivo: disponibilizar um método `invoke()` para clientes remotos que empregam um certo protocolo.

- No nível do código de aplicação, clientes remotos não chamam o método `invoke()`. Eles chamam os métodos negociais (*business methods*) do componente com o qual estão interagindo. Um proxy dinâmico faz a conversão entre a interface do componente vista pelo cliente e a interface de invocação genérica implementada pelo invocador remoto. Em outras palavras: o proxy dinâmico recebe uma chamada a um método negocial de um certo componente (`efetuaMatricula()`, por exemplo) e a traduz (através de seu `InvocationHandler`) numa chamada remota ao método `invoke()` de um "objeto invocador" residente num servidor JBoss. Para fazer isso, o proxy dinâmico teria que conhecer esse "objeto invocador" remoto. O significado preciso de "conhecer" depende do protocolo empregado na interação entre o proxy dinâmico e o "objeto invocador". No caso de um proxy dinâmico associado a um `JRMPInvoker`, "conhecer" significa "possuir uma referência RMI para o `JRMPInvoker` remoto".

- Pelo exposto acima, proxies dinâmicos associados a invocadores diferentes deveriam ter formatos diferentes. Um proxy dinâmico associado a um invocador JRMP possuiria uma referência RMI, um proxy dinâmico associado a um invocador HTTP possuiria uma URL, um proxy dinâmico associado a um invocador IIOP possuiria uma IOR, etc. Entretanto, isso não acontece. Todo proxy dinâmico possui simplesmente uma referência **local** para um *invoker proxy* que implementa a interface `Invoker`[1] e encapsula tudo que for específico de um certo protocolo. Dessa forma se consegue uniformizar os proxies dinâmicos associados aos diferentes invocadores. "Invoker proxies" para os diferentes protocolos são criados no servidor e registrados num contexto de nomes JNDI. Como eles são externalizáveis, podem ser repassados (por valor) aos clientes.

- O uso de proxies dinâmicos é transparente para as aplicações clientes. Quando um cliente recebe uma referência remota para um componente, na verdade ele está recebendo uma cópia de um proxy dinâmico criado no servidor e passado por valor (via seriação de objetos Java). O proxy dinâmico, por sua vez, possui um *invoker proxy* que "conhece" um "objeto invocador" remoto e usa determinado protocolo para interagir com este objeto.

Vimos também que o invocador IIOP do JBoss não segue o esquema descrito acima. (Você sabe explicar por que ele é uma exceção?)

2 O *Internet Communication Engine*

O *Internet Communication Engine* (ICE) é um middleware para objetos distribuídos que lembra bastante CORBA. Criado por pessoas profundamente envolvidas com o desenvolvimento de CORBA e insatisfeitas com certos problemas da especificação do OMG, o ICE pretende eliminar esses problemas e ser mais simples de usar do que CORBA.

O ICE inclui:

- SLICE (Specification Language for ICE) uma linguagem de especificação de interfaces semelhante à IDL de CORBA;
- mapeamentos de SLICE para C++ e Java;
- um protocolo para chamadas remotas de métodos sobre TCP/IP, UDP/IP, e SSL (este protocolo substitui o IIOP);
- um adapatador de objetos mais simples que o POA;
- suporte para chamadas remotas sobre SSL;
- suporte para facetas, à la COM/DCOM.

Para mais informações, veja `http://www.zeroc.com/`. O ICE e sua documentação estão disponíveis nesse sítio. Para uma comparação entre o ICE e CORBA, veja `http://www.zeroc.com/iceVsCorba.html`.

[1] Todo *invoker proxy* usa algum protocolo para interagir com determinado "objeto invocador" remoto e implementa a mesma interface (`org.jboss.invocation.Invoker`), independentemente do protocolo empregado na interação com o invocador remoto.

3 O Que Você Deve Fazer

Neste exercício você implementará um invocador ICE que siga o esquema descrito na seção 1. Seu `ICEInvoker` receberá invocações remotas através do protocolo do ICE e repassará cada invocação para o *container MBean* associado ao EJB alvo. O invocador que você escreverá será voltado exclusivamente para clientes Java, pois utilizará proxies dinâmicos no lado do cliente. Antes de escrever esse invocador você deve ler o material sobre *service MBeans* do JBoss (esse material está no xerox do CAMAT) e estudar detalhadamente o código de algum dos invocadores existentes no JBoss. (O `JRMPInvoker` é um bom modelo. O `IIOPInvoker` não é, pois ele não segue o esquema descrito na seção 1.)

Além de ser um *service MBean* do JBoss, seu `ICEInvoker` deve ser um servente ICE que implementa a seguinte interface SLICE:

```
module org {

    module jboss {

        module invocation {

            module ice {

                sequence<byte> JavaSerializedObject;

                exception InvocationException {
                    JavaSerializedObject javaException;
                };

                interface Invoker {
                    nonmutating string getServerHostName();
                    JavaSerializedObject invoke(JavaSerializedObject invocation)
                        throws InvocationException;
                };
            };
        };
    };
};
```

O arquivo SLICE com a definição dessa interface está disponível em `http://www.ime.usp.br/~reverbel/SMA-03/trabalhos/ep1/ep1.ice`. Os arquivos-fonte Java automaticamente gerados (pelo `slice2java`) a partir dessa definição estão disponíveis no diretório `http://www.ime.usp.br/~reverbel/SMA-03/trabalhos/ep1/generated/`.

Use o método `startService()` para registrar o servente ICE com um adaptador de objetos[2] e para obter um *ICE proxy*[3] para a interface `org.jboss.invocation.ice.Invoker`.

Escreva também a classe `ICEInvokerProxy`, que implementa um *invoker proxy* associado a um `ICEInvoker`. Esse *invoker proxy* possui um *ICE proxy* para o objeto ICE remoto implementado

[2]Assim como um servente CORBA deve ser registrado com um POA, um servente ICE deve ser registrado com um *object adapter* do ICE, que é análogo a um POA.

[3]Um *ICE proxy* é análogo a uma referência CORBA. Para invocar métodos sobre um objeto ICE, um cliente precisa possuir um *ICE proxy* que referencia esse objeto. Assim como referências CORBA, *ICE proxies* podem ser convertidos para strings e vice-versa.

por seu `ICEInvoker`. Ele implementa a interface `org.jboss.invocation.Invoker` simplesmente repassando as chamadas ao `ICEInvoker` remoto. É importante que o `ICEInvokerProxy` seja externalizável e que seus métodos `readExternal()` e `writeExternal()` sejam definidos de modo a permitir que ele seja passado por valor do servidor para os clientes.

Para testar e exercitar seu `ICEInvoker`, crie uma configuração do JBoss que use esse invocador como default. Implante num servidor com essa configuração os EJBs desenvolvidos no "EP zero", que (em princípio) não devem precisar de alteração alguma para funcionarem com o `ICEInvoker`.

Sua solução deve rodar em JBoss 3.2.2. Use o programa `ant` para automatizar a geração do seu invocador.

Este exercício deve ser feito preferencialmente em equipes de duas pessoas. O ideal é o esquema de "programação pareada" (*pair programming*) de XP. Como alguns podem ter restrições de horário que os impeçam de trabalhar assim, aceitarei também exercícios individuais.

Dúvidas sobre o enunciado devem ser enviadas para `reverbel-sma@ime.usp.br`.

Bom trabalho!