

MAC 0462/5863 - Sistemas de Middleware Avançados

Segundo Semestre de 2006

Invocador Java/IIOP para o JBoss

No primeiro trabalho você conheceu EJB sob o ponto de vista de um usuário. Neste você se familiarizará com o lado interno de um servidor Java EE, 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 de negócios (*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

¹Todo *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.

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 serialização 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 Que Você Deve Fazer

Sua tarefa é implementar e testar um invocador IIOP que siga o esquema acima. Ao contrário do invocador IIOP existente no JBoss, o invocador que você escreverá será voltado exclusivamente para clientes Java, pois utilizará proxies dinâmicos no lado do cliente. Por esse motivo, vamos chamá-lo de `JavaIIOPInvoker`. Antes de escrever esse invocador, leia as seções 2.4.2 (*JBoss MBean Services*) e 2.4.3 (*Writing JBoss MBean Services*) do *JBoss Application Server Guide* (disponível em <http://labs.jboss.com/portal/jbossas/docs>) e estude detalhadamente o código de algum dos invocadores existentes no JBoss (o `JRMPInvoker`, por exemplo).

Além de ser um *service MBean* do JBoss, seu `JavaIIOPInvoker` deve ser um servente CORBA que implementa a seguinte interface IDL:

```
#ifndef _INVOKER_IDL_
#define _INVOKER_IDL_

module org {

    module jboss {

        module invocation {

            module javaiiop {

                typedef sequence<octet> JavaSerializedObject;

                exception InvocationException {
                    JavaSerializedObject javaException;
                };

                interface Invoker {
                    readonly attribute string serverHostName;
                    JavaSerializedObject invoke(in JavaSerializedObject invocation)
                        raises(InvocationException);
                };
            };
        };
    };
};
```

Use o método `startService()` para registrar o servente CORBA com um POA persistente e para obter uma referência CORBA para a interface `org.jboss.invocation.javaiiop.Invoker`.

Escreva também a classe `JavaIIOPInvokerProxy`, que implementa um *proxy invoker* associado a um `JavaIIOPInvoker`. Esse *proxy invoker* possui uma IOR para o objeto CORBA remoto implementado por seu `JavaIIOPInvoker`. Ele implementa a interface `org.jboss.invocation.Invoker` simplesmente repassando as chamadas ao objeto CORBA remoto. É importante que o `JavaIIOPInvokerProxy` 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.

Sua solução deve rodar na versão 4.0.4 GA do JBoss. Use o programa `ant` para automatizar a geração do seu invocador. Para testar e exercitar seu `JavaIIOPInvoker`, crie uma configuração do JBoss que use esse invocador como default. Implante num servidor com essa configuração EJBs compatíveis com a versão 2.1 da especificação EJB, que (em princípio) não devem precisar de alteração alguma para funcionarem com o `JavaIIOPInvoker`. Use, em seus testes, os componentes EJB 2.1 disponibilizados juntamente com este enunciado.

Dúvidas sobre este trabalho devem ser enviadas para a lista de discussão de SMA.

Bom trabalho!