Design Patterns J2EE - Service Locator
Enviado por Terça, novembro 26 @ 10:38:13 EDT por **Erro! A referência de hyperlink não é válida.**

http://www.portaljava.com/home/modules.php?name=Search&query=&topic=9ricados
writes *"Nome*

Service Locator

Outros nomes

EJBHomeFactory (não é exatamente outro nome, mas ambos se assemelham em alguns aspectos)

Contexto

Antes de um cliente utilizar um EJB, é preciso que ele encontre ou então crie uma nova instância do bean.

Problema

Utilizar um EJB implica em alguns passos:

- criação de um InitialContext JNDI, que representa os mapeamentos entre nome lógico e nome da classe do bean
- lookup da interface home, que controla o ciclo de vida do componente
- criação ou recuperação do EJB
- finalmente, a utilização dos seus serviços

Além de incrementar a complexidade do código cliente, estas atividades são necessárias para todos os clientes que utilizarem um mesmo bean, o que gera um alto nível de repetição de código. Por exemplo, se um EJB é acessado por helper classes, por uma aplicação console, por um Session Facade e até por outros beans, as tarefas apontadas acima serão repetidas em todos os tipos de clientes.

Solução

Criar uma classe (ServiceLocator) que transpareça do cliente e concentre toda a complexidade JNDI.

Consequências

+ Melhora de performance, já que mecanismos de cache podem ser implementados
no ServiceLocator, para evitar lookups desnecessários
+ Reduz a complexidade do código cliente

+ Facilita a manutenção, por não pulverizar o mesmo trecho de código em vários
pontos da aplicação
+ Tira do cliente toda dependência ao mecanismo de lookup

Estratégias

```
/**
 * ServiceLocator que implementa um cache de interfaces home
 * @author alegomes
 */
class ServiceLocator
{
    /**
     * Contexto JNDI cuja instaciação será reduzida, já que o custo
     * deste tipo de operação é alto.
     */
    private InitialContext jndiContext;

    /**
     * Cache de interfaces home, para reduzir o numero de lookups
     */
    private Hashtable homeCache;

    /**
     * Este ServiceLocator será um Singleton
     */
```

```
private static ServiceLocator instance;

/**
 * Criação
 */
public ServiceLocator()
{
    homeCache = new Hashtable();
    jndiContext = new InitialContext();

    //Obs: excessões omitidas
}

/**
 * Recupera a instancia do Singleton
 */
public ServiceLocator getLocator()
{
    return instance;
}

/**
 * Recupera uma interface home.
 * O lookup do bean só é feito se não houver nenhuma
 * referência sua no cache
 */
```

```
    public EJBHome getService(String jndiName)
    {
        if (!homeCache.containsKey(jndiName))
        {
            //É a primeira vez que este bean é solicitado.
            //Vamos colocá-lo no cache.
            Object ref = jndiContext.lookup(jndiName)
            EJBHome home = (EJBHome)PortableRemoteObject.narrow(ref,
EJBHome.class);
            homeCache.put(jndiName, home);
        }

        return homeCache.get(jndiName);
    }
}

}
```

Do lado cliente, para um bean ser utilizado, basta utilizarmos:
ServiceLocator.getLocator().getService("meuEJBPreferido");

Observações
---
1 - Este código está longe de ser a solução ótima. Não é o objetivo.
2 - Uma outra estratégia é implementar o ServiceLocator como Stateful Session Bean.
3 - O ServiceLocator pode ser utilizado para a recuperação de qualquer tipo de serviço que esteja disponível na estrutura JNDI:
- QueueConnectionFactory
- Queue
- TopicConnectionFactory
- Topic
- DataSource
- URL
- EJBLocalHome

Próximo design pattern J2EE: ValueListHandler

Bons patterns !!!

Alexandre Gomes
Sou Jedi, Sou Java!"