Serviços OSGi - Tutorial (traduzido)

segunda-feira, 25 maio 2015, 11:51 AM

Serviços OSGi - Tutorial

Lars Vogel

Versão 5.1

2014/07/18

Serviços OSGi com o Eclipse Equinox

Este tutorial explica o uso de serviços OSGi primário com foco no uso de serviços declarativas. Eclipse Equinox é usado como um servidor OSGi autônomo. Para este tutorial Eclipse 4.4 (Luna) é usado.

Índice

1. Pré-requisito

O seguinte assume que você está familiarizado com o tempo de execução OSGi e sua camada de modularidade como descrito no **OSGi modularidade** .

2. Serviços OSGi

2.1. O que são serviços OSGi?

Um *serviço* em OSGi é definida por uma classe Java padrão ou interface. Um plug-in pode registrar novos serviços e consumir serviços existentes através do tempo de execução OSGi. OSGi fornece uma central de *registro de serviços* para esta finalidade.

Um serviço pode ser iniciado dinamicamente e parou, e plug-ins que utilizam os serviços devem ser capazes de lidar com esse comportamento dinâmico. Os plug-ins pode registrar ouvintes de ser informado se um serviço é iniciado ou parado.

2.2. Status de ciclo de vida para a prestação de serviços

Para fornecer um serviço de um plug-in precisa estar no `ACTIVE` status de ciclo de vida de OSGi.

Isto requer que o serviço de plug-in tem a *Activate este plug-in quando uma de suas classes é carregado* sinalizador definido no arquivo de manifesto.

2.3. Melhores práticas para a definição de serviços

É uma boa prática para definir um serviço por meio de um plug-in que só contém a definição de interface. Outro plug-in iria fornecer a implementação para este serviço. Isso permite que você mude a implementação do serviço através de um plug-in diferente.

2.4. Propriedades de serviço

Durante a declaração de um serviço, é possível especificar valores de chave / que podem ser utilizados para configurar o serviço.

2.5. Prioridades de serviço

É possível definir um ranking de serviço de um serviço através de uma propriedade de serviço. OSGi atribui por padrão um valor de zero como o ranking de serviços. Quanto mais alto o ranking melhor. Frameworks como o quadro de injeção de dependência Eclipse injetar automaticamente o serviço com o ranking mais alto de serviço.

O `Constantes` classe do `org.osgi.framework` pacote define a *`service.ranking`* valor através da `Constants.SERVICE_RANKING` constante. Esta constante pode ser usado para definir a propriedade de número inteiro do ranking serviço.

3. A funcionalidade de serviços declarativos OSGi

3.1. Definindo serviços declarativas

O OSGi *serviços declarativos* funcionalidade (DS) permite que você defina e consumir serviços através de metadados (XML) sem qualquer dependência em seu código-fonte para a estrutura OSGi.

O *componente de serviço OSGi* é responsável por iniciar o serviço (componente de serviço). Para o consumidor do serviço não é visível se o serviço foi criado por meio de serviços declarativas ou por outros meios.

Componentes de serviço consistem em uma descrição XML (descrição do componente) e um objeto (instância do componente). A descrição do componente contém todas as informações sobre o componente de serviço, por exemplo, o nome da classe da instância do componente ea interface de serviço que presta. Plug-ins geralmente definem descrições de componentes em um diretório chamado `OSGI-INF` .

Uma referência para o arquivo de descrição do componente é inserido no `MANIFEST.MF` arquivo por meio do `Service Component-` propriedade. Se o tempo de execução OSGi encontra tal referência, os `org.eclipse.equinox.ds` plug-in cria o serviço correspondente.

O exemplo a seguir `MANIFEST.MF` arquivo demonstra como uma referência a um arquivo de definição de componente parece.

```
Manifest-Version: 1.0
Bundle-manifestVersion: 2
Bundle-Nome: Service
Bundle-SymbolicName: com.example.e4.rcp.todo.service
Bundle-Version: 1.0 .0.qualifier
Bundle-Vendor: EXEMPLO
Bundle-RequiredExecutionEnvironment: JavaSE- 1.6
Bundle-ActivationPolicy: preguiçoso
Serviço de Componente: OSGi-INF / service.xml
```

3.2. Bundles necessários

Para usar os serviços declarativos os seguintes plug-ins deve estar disponível em tempo de execução.

- `org.eclipse.equinox.util` - contém classes de utilitários
- `org.eclipse.equinox.ds` - é responsável por ler os metadados do componente e para criar e registrar os serviços baseados esta informação
- org.eclipse.osgi.services - funcionalidade do serviço utilizado pelos serviços declarativas

3.3. Ativando o serviço declarativa plug-in

Um plug-in que fornece o serviço deve estar em sua `ACTIVE` status de ciclo de vida. Portanto, garantir que o *Activate este plug-in quando uma de suas classes é carregado* sinalizador é definido no `MANIFEST.MF` arquivo. Veja **Seção 2.2, "estado de ciclo de vida para a prestação de serviços"** para mais detalhes.

4. Definir o nível de arranque para os serviços declarativas

Se você usar serviços OSGi DS fora aplicações Eclipse RCP, você precisa se certificar de que os `org.eclipse.equinox.ds` plug-ins é iniciado antes de qualquer aplicação plug-in que quer consumir um serviço.

Você pode garantir isso em sua configuração de lançamento, definindo a *auto-start* campo a verdade eo nível de arranque inferior a 4 (quatro é o valor padrão) para os org.eclipse.equinox.ds plug-in

5. Passos para declarar um serviço OSGi

5.1. Definindo a interface de serviço

O primeiro passo para definir um serviço OSGi é definir a classe ou interface para o qual você deseja fornecer um serviço. Isto é chamado de *interface do serviço* , mesmo que ele também pode ser uma classe Java.

5.2. Fornecer uma implementação de serviço

Como segundo passo você escrever a classe de implementação para a interface de serviço.

5.3. Declaração serviço com um arquivo de definição de componente

Depois que você forneceu a implementação você precisa registrá-lo para a interface de serviço. Em OSGi DS isso é feito por meio de um arquivo de definição de componente.

O Eclipse IDE fornece um assistente para criar tais arquivos através da *New* → *Outros ...* → *Plug-in Development* → *Component Definition* entrada do menu. Esse assistente também adiciona a `Service Component-` entrada para o *`MANIFEST.MF`* arquivo.

Na primeira página do assistente, você pode digitar o nome do arquivo de definição de componente, um componente nome ea classe que implementa a interface de serviço.

Se você pressionar o *Concluir* botão, o arquivo é criado eo editor correspondente se abre.

Neste editor você pode especificar o serviço prestado e necessário no *Serviços* guia. Para fornecer um serviço que você pressione o *Add ...* botão abaixo *Serviços Prestados* e selecione a interface de serviço que você deseja fornecer.

Por exemplo supor que você quer para fornecer um serviço para o *ITodoService* de interface através do `MyTodoServiceImpl` classe. Um mantida corretamente *`component.xml`* arquivo XML seria parecido com o seguinte.

```
<? Xml version = "1.0" encoding = "UTF-8" ?>
<Scr: xmlns componentes: scr = "http://www.osgi.org/xmlns/scr/v1.1.0" name = "ITodoServic
e" >
   <Implementação class = "com.example.e4.rcp.todo.service.internal.MyTodoServiceImpl" />
   <Serviço>
      <Fornecer interface de = "com.example.e4.rcp.todo.model.ITodoService" />
   </ Service>
</ Scr: component>
```

5.4. Referência para o serviço no arquivo MANIFEST.MF

Após a definição da sua componente *`MANIFEST.MF`* ficheiro contém uma entrada para o componente de serviço.

```
Manifest-Version: 1.0
Bundle-manifestVersion: 2
Bundle-Nome: Service
Bundle-SymbolicName: com.example.e4.rcp.todo.service
Bundle-Version: 1.0 .0.qualifier
Bundle-Vendor: EXEMPLO
Bundle-RequiredExecutionEnvironment: JavaSE- 1.6
Exigir-Bundle: com.example.e4.rcp.todo.model; bundle-version = "1.0.0"
Bundle-ActivationPolicy: preguiçoso
Serviço de Componente: OSGI-INF / component.xml
```

5.5. Baixo nível de API do serviço OSGi

OSGi também oferece uma API de baixo nível para iniciar, parar e acompanhamento de serviços. Veja **Seção 8.1, "Usando a API do serviço"** para uma referência.

6. Tutorial: Definir um OSGi Service declarativa

A seguir irá definir um serviço DS baseado no exemplo das citações. Por conseguinte, é necessário que você tenha criado o projecto "de.vogella.osgi.quote", que contém a definição de interface.

Criar um novo plug-in projeto "de.vogella.osgi.ds.quoteservice". Não use um modelo, não crie um ativador. Importação pacote "de.vogella.osgi.quote" em MANIFST.MF na guia *Dependências* .

Crie o *OSGI-INF* pasta no seu projeto. Crie uma nova definição de componente, conforme descrito anteriormente. A classe de implementação é de.vogella.osgi.ds.quoteservice.QuoteService que fornece o serviço para IQuoteService.

Criar a classe "QuoteService" que implementa o IQuoteService interface.

```
pacote de.vogella.osgi.ds.quoteservice;

importação java.util.Random;

importação de.vogella.osgi.quote.IQuoteService;

público  classe QuoteService implementa IQuoteService {

  Override
   público GetQuote String () {
    Random random = new aleatória ();
     // cria um número entre 0 e 2
    int nextInt = Random.nextInt ( 3 );
     interruptor (nextInt) {
     case  0 :
       return  "Ds: Diga-lhes que eu disse alguma coisa" ;
     caso  1 :
       retorno  "Ds: Já me sinto melhor" ;
     padrão :
       retorno  "Ds: Hubba Bubba, bebê!" ;
    }
  }

}
```

Abrir component.xml e selecione a aba "Source". O resultado final deverá ser parecido com o seguinte.

```
<? Xml version = "1.0" encoding = "UTF-8" ?>
<Scr: xmlns componentes: scr = "http://www.osgi.org/xmlns/scr/v1.1.0" name = "ITodoServic
e" >
   <Implementação class = "com.example.e4.rcp.todo.service.internal.MyTodoServiceImpl" />
   <Serviço>
      <Fornecer interface de = "com.example.e4.rcp.todo.model.ITodoService" />
   </ Service>
</ Scr: component>
```

Copie o "org.eclipse.equinox.ds * .jar", "org.eclipse.osgi.services.jar" e "org.eclipse.equinox.util * .jar" a partir do seu diretório de instalação do Eclipse / plugin em uma pasta, por exemplo, "C: \ temp \ feixes \ plugins" e instalar o pacote em seu tempo de execução OSGi via.

```
arquivo de instalação: c: \ temp \ feixes \ plugins \ org.eclipse.equinox.ds.jar
arquivo de instalação: c: \ temp \ feixes \ plugins \ org.eclipse.equinox.util.jar
arquivo de instalação: c: \ temp \ feixes \ plugins \ org.eclipse.osgi.services.jar
```

Inicie os pacotes manualmente para que os serviços declarativos estão disponíveis.

Exportar o seu próprio pacote de "de.vogella.osgi.ds.quoteservice.jar". e instalá-lo via:

```
arquivo de instalação: c: \ temp \ feixes \ plugins \ de.vogella.osgi.ds.quoteservice.jar
```

Para verificar se o serviço foi registrado usar os "serviços" de comando. Isto irá listar todos os serviços instalados e disponíveis.

Se você parar / desinstalar o prestador de serviços de idade e iniciar o novo seu serviço deve ser captado pelo consumidor.

7. Tutorial: Usando serviços através de serviços declarativas

Claro que você também pode definir o consumo de serviços via DS.

Criar um novo plug-in "de.vogella.osgi.ds.quoteconsumer". Não use um modelo, não crie um ativador. Importe o pacote "de.vogella.osgi.quote" em MANIFEST.MF no *Dependências* guia.

Crie a seguinte classe.

```
pacote de.vogella.osgi.ds.quoteconsumer;

importação de.vogella.osgi.quote.IQuoteService;

público  classe QuoteConsumer {
   privado serviço IQuoteService;

  público  vazio quote () {
    System.out.println (service.getQuote ());
  }

  // Método serão utilizados pela DS para definir o serviço de cotação
  pública  sincronizado  vazio setQuote (serviço IQuoteService) {
    System.out.println ( "Serviço foi criado Obrigado DS.!" );
     este .serviço = serviço;
     // Eu sei que não deveria usar o serviço aqui, mas apenas para demonstração
    System.out.println (service.getQuote ());
  }

  // Método serão utilizados pela DS para remover o serviço de cotação
  pública  sincronizado  vazio unsetQuote (serviço IQuoteService) {
    System.out.println ( "O serviço foi desactivado Por que você fez isso comigo.?" );
     se ( este serviço .serviço ==) {
```

```
        este .serviço = null;
    }
  }
}
```

> **Ponta:** Note-se que esta classe não tem nenhuma dependência para OSGi.

Crie o *OSGI-INF* pasta e criar uma nova *definição de componentes* nesta pasta.

Desta vez, vamos usar um serviço. Manter os "Serviços referenciado".

Faça a relação com o `bind ()` e `bind ()` método, selecionando a entrada pode pressionando o *Editar* botão.

O component.xml resultado deve ser parecido:

```
<? Xml version = "1.0" encoding = "UTF-8" ?>
<Scr: xmlns componentes: scr = "http://www.osgi.org/xmlns/scr/v1.1.0" name = "de.vogella.
osgi.ds.quoteconsumer" >
   <Implementação class = "de.vogella.osgi.ds.quoteconsumer.QuoteConsumer" />
   <Referência bind= "setQuote" cardinality= "1..1"  interface = "de.vogella.osgi.quote.I
QuoteService" name= "IQuoteService" policy= "static" unbind= "unsetQuote" />
</ Scr: component>
```

O resultado MANIFEST.MF deve ser semelhante a:

```
Manifest-Version: 1.0
Bundle-manifestVersion: 2
Bundle-Nome: Quoteconsumer
Bundle-SymbolicName: de.vogella.osgi.ds.quoteconsumer
Bundle-Version: 1.0 . 4
Bundle-RequiredExecutionEnvironment: JavaSE- 1.6
Import-Package: de.vogella.osgi.quote
Serviço de Componente: OSGI-INF / component.xml
```

Exporte seu plug-in e instalá-lo via: arquivo de instalação: c: \ temp \ feixes \ plugins \ de.vogella.osgi.ds.quoteconsumer.jar

"Se você iniciar o pacote agora com" começar id_of_your_bundle "você deve obter o feedback que o serviço foi criado e uma citação deve ser devolvido

8. Serviço OSGi de baixo nível API

8.1. Usando a API do serviço

OSGi fornece vários meios de declarar serviços. Este foco livro sobre a funcionalidade do serviço OSGi declarativa, mas também é possível utilizar outros meios para a definição de serviços. Estas opções estão representados na figura a seguir. Blueprint e Serviços declarativos fornecer abstrações de alto nível para o manuseamento de serviços.

Este capítulo descreve a API para trabalhar diretamente com serviços OSGi, mas, se você tem a opção, você deve preferir abstrações de nível superior como estas simplificar o manuseio de serviços OSGi.

8.2. BundleContext

O acesso ao registro serviço é realizado através do `BundleContext` classe.

Um pacote pode definir um `Bundle-Activator` (Activator) em sua declaração. Esta classe deve implementar a `BundleActivator` interface.

Se definido, OSGi injeta o `BundleContext` para o `start ()` e `stop ()` métodos da implementação `Activator` classe.

```
importação org.osgi.framework.BundleActivator;
 importação org.osgi.framework.BundleContext;


público  classe Activator implementa BundleActivator {

  público  vazio partida (contexto BundleContext) lança Exceção {
    System.out.println ( "Iniciando o pacote" );
     // fazer alguma coisa com o contexto, por exemplo
    // registrar serviços
  }

  público  vazio stop (contexto BundleContext) lança Exceção {
    System.out.println ( "Parando pacote" );
     // fazer alguma coisa com o contexto, por exemplo
    // unregister serviço
  }

}
```

Se você não tiver uma `Activator` , você pode usar o `FrameworkUtil` classe a partir da estrutura OSGi, que lhe permite recuperar o `BundleContext` para uma classe.

```
BundleContext BundleContext =
        FrameworkUtil.
        getBundle ( este .getClass ()).
        getBundleContext ();
```

8.3. Registrando serviços via API

Um pacote também pode registrar-se para os eventos (`ServiceEvents`) do `BundleContext` . Estes são, por exemplo, acionado se um novo pacote é instalado ou de-instalado ou se um novo serviço está registrado.

Para publicar um serviço no seu uso pacote:

```
público  classe Activator implementa BundleActivator {
   // ...
  público  vazio partida (contexto BundleContext) lança Exceção {
    contexto.
      RegisterService (. IMyService classe .getName (),
```

```
            novo ServiceImpl (), null);

    }
    // ...
}
```

Uma vez que o serviço não é mais usado, você deve cancelar o registro do serviço com OSGi. OSGi conta o uso dos serviços de permitir a substituição dinâmica de serviços. Então, uma vez que o serviço não é mais utilizado por sua aplicação, você deve dizer o ambiente OSGi isso:

```
context.ungetService (ServiceReference);
```

No `RegisterService ()` método do `BundleContext` classe que você pode especificar propriedades arbitrárias no parâmetro dicionário.

Você pode usar o `getProperty ()` método da `ServiceReference` classe do `org.osgi.framework` pacote, para aceder a uma propriedade específica.

8.4. Acessando um serviço via API

Um pacote pode adquirir um serviço através do `BundleContext` classe. O exemplo que se segue demonstra que.

```
ServiceReference <?> ServiceReference = contexto.
  getServiceReference (. IMyService classe .getName ());
Serviço IMyService = (IMyService) contexto.
  getService (ServiceReference);
```

8.5. De baixo nível API vs serviços declarativos OSGi

Serviços OSGi pode ser iniciado e interrompido dinamicamente. Se você trabalha com o OSGI de baixo nível API você tem que lidar com essa dinâmica em seu código. Isto faz o complexo desnecessário código fonte. Se você não lidar com isso corretamente o consumidor de serviço pode manter uma referência para o serviço eo serviço não pode ser removido por meio da estrutura OSGi.

Para lidar com a dinâmica de serviços automaticamente declarativas foram desenvolvidos. Prefere, portanto, o uso de OSGi *serviços declarativas* através da API de baixo nível.

9. Tutorial: Usando a API do serviço OSGi

A seguir vamos definir e consumir um serviço. Nosso serviço irá retornar "citações famosas".

9.1. Definir a interface de serviço

Criar um projeto de plug-in "de.vogella.osgi.quote" eo pacote "de.vogella.osgi.quote". Não use um modelo. Você não precisa de um ativador. Depois selecione o MANIFEST.MF e do *Runtime* guia. Adicionar "de.vogella.osgi.quote" para os pacotes exportados.

Criar a seguinte interface "IQuoteService".

```
pacote de.vogella.osgi.quote;

público  interface de IQuoteService {
  Corda GetQuote ();
```

```
}
```

9.2. Crie serviço

Vamos agora definir um pacote que irá fornecer o serviço.

Criar um projeto de plug-in "de.vogella.osgi.quoteservice". Não use um modelo.

Selecione o MANIFEST.MF e ser dependente guia. Adicionar "de.vogella.osgi.quote" para os plugins necessários.

Crie a seguinte classe "QuoteService".

```
empacotar de.vogella.osgi.quoteservice.internal;

importação java.util.Random;

importação de.vogella.osgi.quote.IQuoteService;

público  classe QuoteService implementa IQuoteService {

  Override
   público GetQuote String () {
    Random random = new aleatória ();
     // cria um número entre 0 e 2
    int nextInt = Random.nextInt ( 3 );
     interruptor (nextInt) {
     caso  0 :
       retorno  "Diga-lhes que eu disse alguma coisa" ;
     caso  1 :
       retorno  "Eu já se sentir melhor " ;
     padrão :
       retornar  "Hubba Bubba, Baby!" ;
    }

  }
}
```

Registar o serviço no Activator classe.

```
pacote de.vogella.osgi.quoteservice;

import java.util.Hashtable;

importação org.osgi.framework.BundleActivator;
 importação org.osgi.framework.BundleContext;

importação de.vogella.osgi.quote.IQuoteService;
 importação de.vogella.osgi.quoteservice.internal.QuoteService;

público  classe Activator implementa BundleActivator {
```

```
  público  vazio partida (contexto BundleContext) lança Exceção {
    IQuoteService service = new QuoteService ();
     // terceiro parâmetro é um hashmap que permite configurar o serviço
    // Não é necessária neste exemplo
    context.registerService (IQuoteService. class .getName (), serviço,
        null);
    System.out.println ( "IQuoteService está registrado" );
  }

  público  vazio stop (contexto BundleContext) lança Exceção {
  }
}
```

9.3. Instalar pacotes de serviços

Exporte seus pacotes e instalá-los no seu servidor. Comece o pacote serviço.

> **Ponta:**Nada extravagante acontece, como nós ainda não estão consumindo o nosso serviço.

9.4. Use o seu serviço

Criar um novo plug-in "de.vogella.osgi.quoteconsumer". Adicione também uma dependência para o pacote "de.vogella.osgi.quote".

> **Ponta:**Por favor, note que nós adicionamos a dependência contra o pacote não contra o plugin. Desta forma, mais tarde substituir o serviço com uma implementação diferente.

Permite registrar diretamente para o serviço e usá-lo.

```
pacote de.vogella.osgi.quoteconsumer;

import org.osgi.framework.BundleActivator;
 import org.osgi.framework.BundleContext;
 import org.osgi.framework.ServiceReference;

importação de.vogella.osgi.quote.IQuoteService;

público  classe Activator implementa BundleActivator {

  privado contexto BundleContext;
   privado serviço IQuoteService;

  público  vazio partida (contexto BundleContext) lança Exceção {
     este .context = contexto;
     // registrar diretamente com o serviço
    Referência ServiceReference = contexto
        .getServiceReference (. IQuoteService classe .getName ());
```

```
    service = (IQuoteService) context.getService (referência);
    System.out.println (service.getQuote ());
  }

  público  vazio stop (contexto BundleContext) lança Exceção {
    System.out.println (service.getQuote ());
  }

}
```

Exportar este pacote, instalá-lo e iniciar e parar. Tudo trabalho. Mas se você parar o pacote serviço, em seguida, seu receber um erro.

A razão para isso é que OSGi é um ambiente muito dinâmico e serviço podem ser registradas e desregistados qualquer momento. O próximo capítulo vai usar um rastreador de serviço para melhorar esta situação.

9.5. Use o seu serviço com um rastreador de serviço

Declare uma dependência de pacotes para o pacote "org.osgi.util.tracker" em seu pacote.

Para usar este defina o seguinte classe "MyQuoteServiceTrackerCustomizer"

```
pacote de.vogella.osgi.quoteconsumer;

import org.osgi.framework.BundleContext;
 import org.osgi.framework.ServiceReference;
 import org.osgi.util.tracker.ServiceTrackerCustomizer;

importação de.vogella.osgi.quote.IQuoteService;

público  classe MyQuoteServiceTrackerCustomizer implementos
    ServiceTrackerCustomizer {

  privado  final de contexto BundleContext;

  público MyQuoteServiceTrackerCustomizer (contexto BundleContext) {
     este .context = contexto;
  }

  privado fio MyThread;

  Override
   público addingService Object (referência ServiceReference) {
    Serviço IQuoteService = (IQuoteService) context.getService (referência);
    thread = new MyThread (serviço);
    Thread.Start ();
    retornar serviço;
  }

  Override
   público  vazio modifiedService (referência ServiceReference, serviço Object) {
```

```
     // removedService (referência, serviço);
    // addingService (referência);
  }

  Override
   público  vazio removedService (referência ServiceReference, serviço Object) {
    context.ungetService (referência);
    System.out.println ( "Como é triste Serviço de cotação está desaparecido." );
    thread.stopThread ();
  }

  público  estático  classe MyThread estende Linha {

    privado  volátil  booleano ativo = true;
     privada  último serviço IQuoteService;

    público MyThread (serviço IQuoteService) {
       este .serviço = serviço;
    }

    público  vazio run () {
       enquanto (ativo) {
        System.out.println (service.getQuote ());
        tente {
          Thread.sleep ( 5000 );
        } prendedor (exceção e) {
          System.out.println ( "Thread interrompida" + e.getMessage ());
        }
      }
    }

    público  vazio stopThread () {
      ativo = false;
    }
  }

}
```

Você também precisa registrar um rastreador de serviço em seu ativador de sua ServiceConsumer.

```
pacote de.vogella.osgi.quoteconsumer;

import org.osgi.framework.BundleActivator;
 import org.osgi.framework.BundleContext;
 import org.osgi.util.tracker.ServiceTracker;

importação de.vogella.osgi.quote.IQuoteService;

público  classe Activator implementa BundleActivator {

  privado serviceTracker ServiceTracker;
```

```
  público  vazio partida (contexto BundleContext) lança Exceção {
    System.out.println ( "Iniciando pacotes quoteconsumer" );
     // registrar diretamente com o serviço
    ao cliente MyQuoteServiceTrackerCustomizer = new MyQuoteServiceTrackerCustomizer (con
texto);
    serviceTracker = new ServiceTracker (contexto, IQuoteService. aula
        .getName (), cliente);
    serviceTracker.open ();
  }

  público  vazio stop (contexto BundleContext) lança Exceção {
    System.out.println ( "Parando pacotes quoteconsumer" );
    serviceTracker.close ();
  }

}
```

Exporte seu pacote novamente. Inicie o console OSGi. Use o comando de atualização ou o comando de instalação para obter a nova versão de seu pacote e iniciá-lo. Uma vez que você começar o seu serviço o rastreador será chamado eo feixe consumidor vai começar a escrever mensagens para o console. Pare o serviço e verificar se o consumidor não usar o serviço mais.

10. Bndtools

Eclipse usar o ferramental PDE para gerenciar pacotes. Alternativamente, você pode usar Bndtools hospedado no **http://bndtools.org/** .

Por favor, veja **Bndtools tutorial** para uma introdução.

11. Sobre este site

11.1. Doações para apoiar cursos livres

Por favor, considere uma contribuição , se este artigo lhe ajudou. Ela vai ajudar a manter nosso conteúdo e nossas atividades de Open Source.

11.2. Perguntas e discussão

Escrever e atualizar estes tutoriais é um monte de trabalho. Se este serviço à comunidade livre foi útil, você pode apoiar a causa, dando uma dica, bem como relatar erros de digitação e erros factuais.

Se você encontrar erros neste tutorial, por favor avise-me (veja o **topo da página**). Por favor, note que, devido ao alto volume de feedback que eu receber, eu não posso responder perguntas a sua implementação. Certifique-se de ter lido o **vogella FAQ**

como eu não responder a perguntas já respondidas lá.

11.3. Licença para este tutorial e seu código

12. Links e Literatura

12.1. Código Fonte

Fonte Código de exemplos

12.2. Recursos OSGi

http://www.osgi.org OSGi Homepage

http://www.eclipse.org/equinox Equinox Homepage

OSGi tutorial serviço remoto RESTFul Equinox Homepage

OSGi remover tutorial serviço com ECF Equinox Homepage

http://www.eclipse.org/equinox Equinox Homepage

http://www.eclipse.org/equinox/documents/quickstart.php guia Equinox Quickstart

http://www.ibm.com/developerworks/opensource/library/os-osgiblueprint/ serviços OSGi Blueprint

12.3. Recursos vogella