



# Liferay e Modularização com Arquitetura OSGi

**Liferay Symposium Brazil 2014**

André Ricardo Barreto de Oliveira "Arbo"  
Core Engineer  
Liferay do Brasil

@arbocombr  
@Liferay\_BR  
#SymposiumBrazil



# Modularização

## Paradigma

OSGi  
Tecnologia

Liferay Portal  
Plataforma



# Modularização

## Paradigma



## A busca por modularização

Tamanho do software crescendo extraordinariamente  
(Pressões de desenvolvimento)

Deploy e redeploy parcial com granularidade mais fina  
(Pressões de produção)

Oportunidades de comercialização com flexibilidade  
(Pressões de negócio)

# Pressões de desenvolvimento

## Não modular

Código fonte demora para checautar

Make demora para construir

Testes demoram para fornecer feedback

Distribuições demoram para fazer download

Histórico de commits misturando funcionalidades

Dependências circulares comprometendo refactoring

## Modular

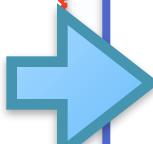
Projetos menores constroem e testam mais rápido, com histórico limpo

Dependências bem definidas, sem circularidades

Downloads reduzidos, incrementais e transparentes

Separação de interface e implementação

Melhor testabilidade de componentes individuais



# Pressões de produção

## Não modular

Upgrade? "Tem que parar o servidor"

Deploy de bugfix crítico?  
"Tem que parar o servidor"

Melhoria isolada?  
"Tem que parar o servidor"

Dependency hell: conflitos de versão  
em bibliotecas compartilhadas

Class loader hell:  
ClassNotFoundException,  
ClassCastException

## Modular

Hot deploy e redeploy de fixes e  
melhorias, sem paradas

Desligar pontos quentes conhecidos  
mantendo no ar o resto da aplicação

Subir recursos somente se  
dependências estiverem presentes

Múltiplas versões da mesma  
biblioteca coexistindo no servidor

Isolamento de class loaders com  
injeção de dependências



# Pressões de negócio

## Não modular

Relatórios esperando para entrar junto com funcionalidades

Falta framework de plugins para injetar consultas e relatórios dinamicamente, on the fly

Vender o produto base + extensões, plugins, módulos, combos

Enxugar o produto para derrubar barreiras de entrada

Novos segmentos de mercado

## Modular

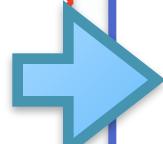
Relatórios, consultas, exports com time-to-market acelerado

Standard bem suportado para componentes e serviços dinâmicos

Produto base enxuto, simples de implantar e usar

Adaptar e reagir à entrada e saída de features no ambiente

Definir facilmente pontos de extensão para plugins de terceiros





**Um mundo modular**

Arquitetura modular é uma *mudança de paradigma*.

É projetar a sua aplicação assumindo ser possível:

**Fabricar e testar peças menores em isolamento;**

**Substituir peças em produção com a aplicação rodando;**

**Fornecer peças novas para estender e aumentar a aplicação.**

*E claro, tendo a tecnologia para fazer tudo isso.*



Liferay Portal

Liferay Portal: *portlet container*

Paradigmas modulares implementados e encorajados há anos:

Arquitetura extensível de portlets, plugins e serviços...

Hot deploy...

Separação entre interface e implementação...

Evolução natural: oferecer **tecnologia modular** de ponta a ponta.

Próximo passo e futuro da arquitetura modular do Liferay Portal?



# OSGi

## Tecnologia

- OSGi: padrão de tecnologia da plataforma Java para modularização
- Tecnologia madura: 15 anos de melhorias e evolução
- Bancada por grandes players: Oracle, **Red Hat**, Eclipse, IBM, **Liferay**
- Todos os principais app servers Java EE já são baseados em OSGi
- Focada em standards e especificações, variadas implementações
- Posta à prova com sucesso em numerosos projetos e produtos



# OSGi: conceitos essenciais

Bundle: o *módulo* OSGi

Bundle container

Ciclo de vida do bundle no deploy

Objeto BundleActivator

OSGi Services

Export-Package & Import-Package

Espaços de classes isolados

Injeção de Dependências



## Bundle: o *módulo* OSGi (nada mais que um JAR)

Em OSGi, o *módulo* não passa de um JAR comum.

### **META-INF/MANIFEST.MF**

Bundle-Name: Liferay Search API

Bundle-SymbolicName: com.liferay.search.api

Bundle-Version: 1.0.0

Export-Package: com.liferay.search.api;version="1.0.0"

(Não se preocupe,  
cabeçalhos são inseridos automaticamente  
por ferramentas como *bndtools*.)



## OSGi bundle container

JAR? Nada de Java SE classpath,  
nem Java EE lib em WAR ou EAR.

**Bundle JAR: deploy é copiar o arquivo  
no diretório do container OSGi.**

"Como assim container? Mais um servidor?"

Não, seu app server Java EE favorito  
na certa já é um container OSGi!

Implementações de bundle containers são bibliotecas.  
(Apache Felix)

Mesmo aplicações Java desktop embarcam bundle containers  
para oferecer mecanismo de plugins.  
(Eclipse Equinox)

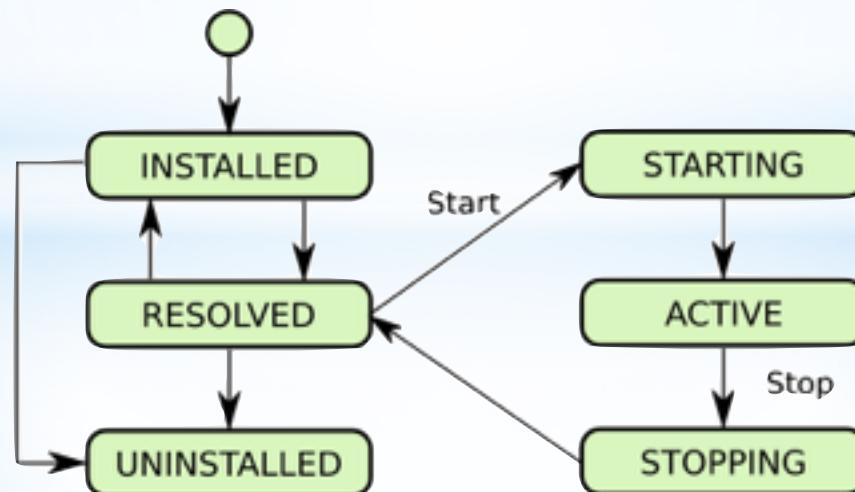
## Ciclo de vida do bundle: deploy no OSGi container

Assim que é copiado no container: **INSTALLED**

Todas as dependências satisfeitas: **RESOLVED**

Executando objeto BundleActivator: **STARTING**

Bundle levantado e funcionando: **ACTIVE**





## BundleActivator

Um bundle jar com BundleActivator **reage** ao entrar ou sair do container.  
(Sem necessidade de static class initializer)

Bundle-Activator: com.liferay.search.elasticsearch.Activator

```
package com.liferay.search.elasticsearch;

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

public class Activator implements BundleActivator {
    public void start(BundleContext context) {
        // Inicializações em geral, registrar serviços...
        // INSTALLED -> RESOLVED -> STARTING -> ACTIVE
    }
    public void stop(BundleContext context) {
        // De-registrar serviços, liberar recursos alocados
        // ACTIVE -> STOPPING -> RESOLVED -> UNINSTALLED
    }
}
```



## OSGi Services

(nada mais que uma interface)

Um OSGi Service pode ser qualquer interface Java. Não precisa mais que isso.

```
package com.liferay.search.api;

public interface SearchEngine {
    public SearchResult search(String query);
}
```

Bundle jar embarca a classe implementadora:

```
import com.liferay.search.api.SearchEngine;
import org.osgi.service.component.annotations.Component;

@Component(service = SearchEngine.class)
public class ElasticsearchSearchEngine implements SearchEngine {
    public SearchResult search(String query) { /* ... */ }
}
```

**Declarative Services (DS):** anotações com várias opções para definir Serviços.



## Export-Package Import-Package

Bundle jar declara quais Java packages **oferece** para outros bundles,  
e também quais Java packages **necessita** de outros bundles.

Ao iniciar um bundle,  
o container procura  
os **Import-Package**.

Bundle-Name: Liferay Search Portlet  
Bundle-SymbolicName: com.liferay.search.portlet  
**Import-Package:** com.liferay.search.api;version="1.0.0"

Bundle-Name: Liferay Search API  
Bundle-SymbolicName: com.liferay.search.api  
**Export-Package:** com.liferay.search.api;version="1.0.0"

Deve existir outro  
bundle oferecendo  
como **Export-Package**.

**Se não encontrar, o bundle não é resolvido e não inicia.**

**Se encontrar, roda o BundleActivator e registra os serviços.**

Bundle-Name: Liferay Search Elasticsearch Implementation  
Bundle-SymbolicName: com.liferay.search.elasticsearch  
**Import-Package:** com.liferay.search.api;version="1.0.0"

Java packages de *interfaces*  
de serviços são exportados,  
mas não os packages das  
implementações.



## Espaços de classes isolados

Em Java SE, qualquer classe pública fica visível para todo o classpath.

Em OSGi, *mesmo classes públicas só são visíveis internamente no bundle.*

Cada bundle *em seu próprio classloader:* isolamento de espaço de classes.

E ainda assim os objetos transitados são *Java puro, sem proxies.*

E mais: Import-Package e Export-Package declaram o *número da versão.*

Como cada bundle tem classloader isolado, o mesmo container pode ter *múltiplas versões da mesma biblioteca, componente ou serviço.*

**Fim do dependency hell.**

OSGi container com anotações **Declarative Services (DS)**:

*injeta automaticamente a instância recebida de outro bundle,  
remove a instância quando o outro bundle sai de operação.*

```
@Component(service=Searcher.class)
public class Searcher {
    @Reference
    protected void setSearchEngine(SearchEngine searchEngine) {
        this.searchEngine = searchEngine;
    }
    protected void unsetSearchEngine(SearchEngine searchEngine) {
        this.searchEngine = null;
    }
}
```

Frameworks clássicos: injeção estática, somente no wiring inicial.

OSGi: injeção e remoção dinâmicas, a qualquer momento, sem parar a aplicação.

# Tudo funcionando

## Apache Felix Web Management Console Bundles



Bundles Components Configuration Configuration Status Deployment Packages Event Admin Licenses OSGI Repository Shell

System Information

Bundle information: 8 bundles in total - all 8 bundles active.

		Browse...	- Start	<input type="checkbox"/>	- Start Level	5	Install or Update	Refresh Packages
Id	Name					Status	Actions	
4	Apache Felix Declarative Services					Active		
6	Apache Felix Shell Service					Active		
7	Apache Felix Shell TUI					Active		
1	Apache Felix Web Management Console					Active		
3	HTTP Service					Active		
5	osgi.compendium					Active		
2	Servlet 2.1 API					Active		
0	System Bundle					Active		

Bundle information: 8 bundles in total - all 8 bundles active.

OSGi runtimes: ferramental variado de operação e monitoria



# Liferay Portal

## Plataforma

Liferay Portal, portlet container: agora *também* OSGi container!

```
/bin/bash
33|Active   | 1|gemini-blueprint-core (2.0.0.M02)
34|Active   | 1|phidias (0.3.2)
35|Active   | 1|JavaServer Pages(TM) Standard Tag Library API (1.2.1)
36|Active   | 1|Expression Language 3.0 (3.0.0)
37|Active   | 1|JavaServer Pages(TM) API (2.3.2.b01)
38|Active   | 1|gemini-blueprint-extender (2.0.0.M02)
39|Active   | 1|gemini-blueprint-io (2.0.0.M02)
41|Active   | 1|Liferay Web Proxy Web (1.0.0)
42|Active   | 1|Liferay XSL Content Web (1.0.0)
43|Active   | 1|Liferay Portal JSON Web Service Tracker (1.0.0)
44|Active   | 1|Console plug-in (1.0.0.v20120522-1841)
45|Active   | 1|Liferay Portal Log Bridge (1.0.1)
46|Active   | 1|Liferay Portal Portlet Tracker (1.0.0)
47|Active   | 1|Liferay Portal Elasticsearch Adapter (1.0.0)
48|Active   | 1|Liferay Portal Servlet JSTL (1.2.3)
49|Active   | 1|com.liferay.portal.servlet.jsp.compiler (1.0.0)
50|Active   | 1|Liferay Portal Verify Extender (1.0.0)
51|Active   | 1|Liferay Portal WAB Extender (1.0.0)
52|Active   | 1|Liferay Portal WAB Generator (1.0.0)
53|Installed | 1|Liferay Portal Spring Extender (1.0.0)
54|Active   | 1|Sample Bundle - Maven (0.0.1.SNAPSHOT)
55|Active   | 1|Sample Bundle - Maven (0.0.1.SNAPSHOT)
56|Resolved  | 1|Sample Bundle - Maven (0.0.1.SNAPSHOT)

g!  █
```



## **Novidades no Liferay 7: Portlets clássicos JSR-168, JSR-286**

- Portlets e Liferay Services clássicos (WAR) continuam a funcionar "as is", sem qualquer tipo de conversão
- São expostos automaticamente no container como serviços OSGi
- Services clássicos podem ser injetados em novos componentes via API OSGi (em vez de static \*Util)
- Fazer deploy, redeploy, desinstalar, iniciar e parar: independente de ferramentas específicas de cada app server



## Portlet clássico...

```
package com.liferay.symposium;

import java.io.IOException;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletException;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

public class MyPortlet extends GenericPortlet {

    protected void doView(RenderRequest request, RenderResponse response)
        throws PortletException, IOException {

        response.getWriter().print("Hello World!");
    }
}
```



## ... e o novo Portlet OSGi

```
package com.liferay.symposium;

import java.io.IOException;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletException;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import org.osgi.service.component.annotations.Component;

@Component(
    service = Portlet.class,
    property = {
        "com.liferay.portlet.display-category=category.sample", "com.liferay.portlet.instanceable=true",
        "javax.portlet.display-name=My DS Portlet", "javax.portlet.security-role-ref=power-user,user"
    }
)
public class MyPortlet extends GenericPortlet {

    protected void doView(RenderRequest request, RenderResponse response)
        throws PortletException, IOException {
        response.getWriter().print("Hello World!");
    }
}
```

(sem mudanças)

(sem mudanças)



## Portlets e Declarative Services (DS)

```
import org.osgi.service.component.annotations.Component;

@Component(
    service = Portlet.class,
    property = {
        "com.liferay.portlet.display-category=category.sample",
        "com.liferay.portlet.instanceable=true",
        "javax.portlet.display-name=My DS Portlet",
        "javax.portlet.security-role-ref=power-user,user"
    })
public class MyPortlet extends GenericPortlet
{
    /* ... */
}
```



## **Novidades no Liferay 7: Portlets e Services OSGi**

- Portlets e Services: serviços OSGi nativos, embarcados em bundles (JAR)
- Properties em anotações no portlet (em vez de arquivos XML)
- Dependências declaradas: independente da ordem de deploy, Portlets e Services aguardam suas dependências serem resolvidas e iniciadas
- Detecção de quebras de API, com baselines e versões: joyful upgrades, previsíveis, melhor documentados
- Seus Portlets e Services definindo os próprios pontos de extensão, com anotações, se beneficiando de todo o ciclo de vida OSGi... Marketplace!
- Ênfase em expor a API e esconder as implementações: melhores testes + abre as portas para remoting

## Novidades no Liferay 7: Hot deploy com OSGi

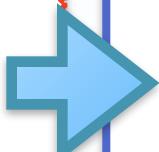
### Deployable WAR clássico em app server

WAR, para os app servers - mesmo só serviços e temas sem web context

Deploy Liferay dependente da implementação de deploy de cada app server

Especificação de portlets nem sequer define empacotamento e deploy

"Damos o war, confiamos no app server para expor serviços na outra ponta"



Deployable JAR bundle em Liferay Portal OSGi Container

Plugin simples JAR OSGi bundle - mesmo com portlets, web resources, JSP

Deploy de plugins não atravessa mais o app server, apenas o container OSGi

Em OSGi, o modelo de empacotamento e deploy é definido na especificação.

Simples, direto e previsível.



## **Novidades no Liferay 7: Escolha suas ferramentas**

Antes: plugin WAR obrigatoricamente  
construído via Liferay SDK  
(ou Maven, sob limites)

Agora: plugin OSGi bundle é um simples JAR,  
dispensando o Liferay SDK.

**Ant, Maven, Gradle, Eclipse, IntelliJ...  
ou a tecnologia de sua preferência!**

Repositório público de artefatos Liferay  
para resolução de dependências  
(Releases, oportunamente dev snapshots)



## **Novidades no Liferay 7: A Modular Liferay**

Desde sempre, destaque do Liferay Portal:  
grande variedade de features "out of the box"

Mas com isso... distribuição padrão "all or nothing"

**Para Liferay 7+: *Lightweight core***

Em andamento: extração massiva de módulos "presos" no core

Polls, Bookmarks, Currency, Web Proxy, Iframe, Blogs...

Download do Liferay Portal básico será bem menor...

Implantações enxutas... com os componentes que você escolher.



Para saber mais:

<http://www.osgi.org/Technology/HowOSGi>

*Liferay dev.life broadcast:*

<http://youtu.be/KPjmB5yj8Og>

[http://youtu.be/NB3jAh\\_Fhul](http://youtu.be/NB3jAh_Fhul)

*Liferay Developer Blogs:*

[https://www.liferay.com/web/raymond.uge/blog](https://www.liferay.com/web/raymond.auge/blog)

Welcome to a modular future :-)

[andre.oliveira@liferay.com](mailto:andre.oliveira@liferay.com)

@arbocombr

@Liferay\_BR

#SymposiumBrazil