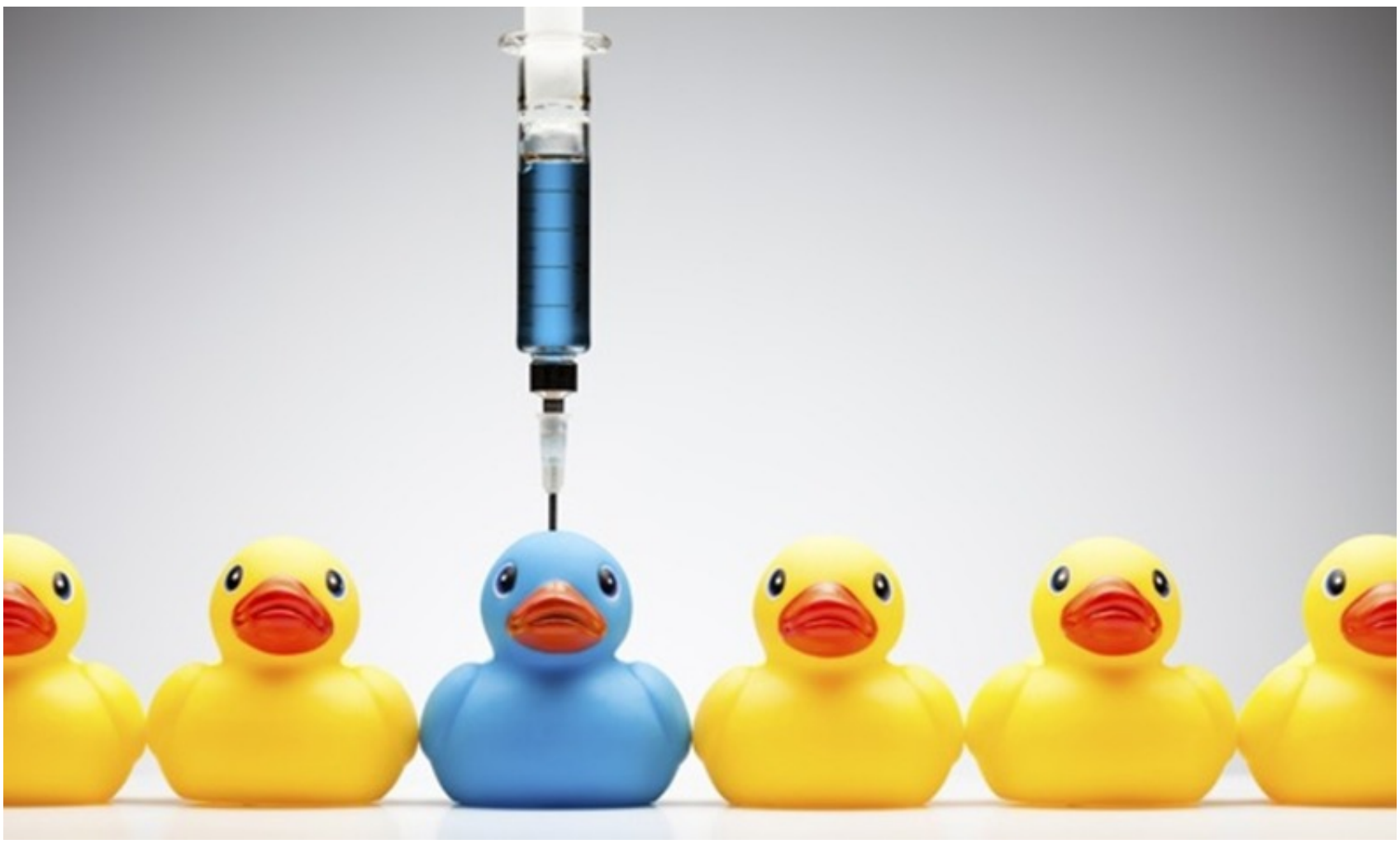
Não misture as anotações do JSF com as anotações do CDI

Entenda por que devemos priorizar as anotações do CDI em vez das anotações JSF

Rafael Ponte - *Desenvolvedor e instrutor na TriadWorks*

September 08, 2015



Com o lançamento do JSF 2 se tornou possível eliminar a necessidade de configurar managed beans e regras de navegação em seu arquivo XML, o tal do **faces-config.xml**. Não só isso, além de trabalharmos com anotações nós ainda diminuímos o número de detalhes de configuração devido as novas convenções do framework. Essas mudanças fizeram com que o **micro container IoC/DI** do JSF ficasse ainda mais conhecido do que na sua versão anterior, como podemos ver a seguir:

```
@ManagedBean
public class AlunosBean {

    @ManagedProperty("#{alunosDao}")
    private AlunosDao dao;

}
```

No código acima, por termos anotado a classe `AlunosBean` com as anotações do JSF ela passou a ser instanciada e gerenciada pelo seu micro container. Além disso a classe `AlunosDao` também está sendo injetada no managed bean pelo JSF. Mas quem está gerenciando a classe `AlunosDao`?

Embora não seja muito comum, podemos deixar o JSF gerenciar outras classes que não os managed beans, como DAOs, Services etc. Se olharmos mais de perto o código da `AlunosDao` veremos ela anotada com as anotações do JSF:

```
@ManagedBean
@ApplicationScoped
public class AlunosDao {

    // ...
}
```

Utilizar o próprio JSF para fazer injeção de dependências é muito melhor do que instanciar todas nossas classes manualmente. Isso permite termos classes menos acopladas e um design mais favorável para testes de unidade.

Mas uma aplicação Web não vive somente com o JSF, não é?

Integrando JSF com outros frameworks

Numa aplicação Web o JSF cuida apenas da camada de apresentação, enquanto para camada de persistência é muito comum o uso da JPA com Hibernate. Integrar estes dois frameworks não é uma tarefa tão fácil assim quando se trata de aplicações de médio-grande porte, pois acabamos com muito código duplicado e de difícil manutenção. Por esse motivo a maioria das aplicações usam algum framework especializado em injeção de dependência (IoC/DI) para fazer a ponte entre eles, como CDI ou Spring.

Ao adotar o CDI no projeto é muito tentador sair anotando as classes do nosso modelo como DAOs, Services, validadores etc para logo em seguida injetá-las nos managed beans do JSF através da anotação `@Inject` do CDI, como no código a seguir:

```
@ManagedBean
public class AlunosBean {

    @Inject
    private AlunosDao dao;

}
```

Em TEORIA a anotação `@Inject` do CDI funciona em "qualquer lugar", inclusive nos managed beans gerenciados pelo JSF. O problema é que isso ainda é **muito falho** na PRÁTICA já que depende do servidor de aplicação e implementação CDI que estivermos trabalhando, como JBoss Weld ou Apache OpenWebBeans.

Há relatos em fóruns e listas de discussão onde ora a injeção funciona em um Wildfly e Glassfish, ora quebra num Tomcat 7 e Jetty. O problema se agrava quando misturamos escopos existentes somente no CDI como `@Dependent` com managed beans anotados pelo JSF.

Para você ter idéia, eu tive exatamente este problema algumas semanas atrás...

Ao customizar o nosso curso de Persistência com JPA 2 para trabalhar com CDI eu quebrei a cabeça por algumas horas com isso. O container do JSF se perdeu com relação aos escopos ao processar uma EL na página. Em vez de injetar o bean do CDI ele estava injetando `null` no managed bean gerenciado pelo JSF e algumas vezes registrava o managed bean em um escopo diferente.

A coisa estava tão feia pro meu lado que fui pedir ajuda a amigos que sacam mais de CDI do que eu, como o Raphael Lacerda e Daniel Cunha. Daí não deu outra: eles me ajudaram a perceber que **misturar as anotações NÃO era uma boa prática**.

Mas de quem é a culpa? O problema está na implementação do CDI ou no servidor de aplicação?

A verdade é que estamos trabalhando com 2 containers de IoC/DI ao mesmo tempo, que tem escopos e ciclo de vida similares na teoria mas com implementações completamente diferentes na prática. Neste momento, o ideal é decidir qual deles instanciará e gerenciará seus objetos daqui para frente...

Qual usar: CDI ou JSF?

Nós já havíamos comentado em um post anterior que as anotações do JSF começaram a ser **depreciadas** e, em futuras versões da Java EE, recomenda-se utilizar apenas as anotações do CDI.

Esse tipo de recomendação não é feita à toa, pois o **micro**-container do JSF possui diversas limitações e resolve apenas o propósito do framework e de seus componentes. Ele se limita a camada de apresentação, enquanto o CDI vai mais longe…

O modelo de componentes do CDI é muito, mas muito superior ao container do JSF. Ele vai além da injeção de dependência, ele resolve de maneira elegante o gerenciamento de escopos e configuração de interceptadores dentro da aplicação; também possui um barramento de eventos simples e eficiente; fora isso podemos criar extensões para plugar em seu container.

O expert group se esforçou no JSF 2.2 e está se esforçando ainda mais no JSF 2.3 para melhorar e estender a integração com o CDI. Dessa forma, nós poderemos trabalhar melhor em cima das anotações do CDI e sem perceber iremos abandonando as anotações do JSF. Isso está ocorrendo com os novos escopos, navegação entre páginas, injeção em conversores e validadores e até na forma como injetamos objetos do contexto do JSF.

A tendência é o CDI se tornar a principal tecnologia para gerenciar objetos na plataforma Java EE, onde praticamente TUDO será gerenciado por seu container e com o tempo substituirá outras especificações que também possuem um container, como EJB e JSF. Dessa forma, teremos uma melhor integração com toda stack Java EE. No fim, todos os objetos no seu servidor de aplicação serão CDI beans…

Por todos estes motivos, se o CDI foi adotado no seu projeto então nada mais sensato do que deixar o próprio **CDI gerenciar todos os objetos** - inclusive os managed beans - através de suas anotações.

Mais como migrar os managed beans para CDI?

Migrando beans do JSF para CDI

Migrar os beans anotados do JSF para CDI é uma tarefa extremamente simples. Basicamente teremos que substituir algumas anotações…

Pegando o managed bean anterior como exemplo, poderíamos migrá-lo facilmente para um bean do CDI com algumas poucas linhas de código:

```
@Named
@RequestScoped // javax.enterprise.context.RequestScoped
public class AlunosBean {

    @Inject
    private AlunosDao dao;

}
```

O que precisamos fazer são 3 coisas:

- Substituir a anotação `@ManagedBean("xxx")` por `@Named("xxx")`, dessa forma conseguimos acessar os objetos via EL (Expression Language) na páginas;
- Mudar as anotações de escopos por anotações equivalentes no CDI. De maneira simples precisamos alterar somente os pacotes: de `javax.faces.bean.*` para `javax.enterprise.context.*`

; A exceção é a anotação `@ViewScoped` que está no pacote `javax.faces.view`;

- Substituir a anotação `@ManagedProperty` por `@Inject`;

Bem simples, não é?

Continuando no ritmo de migrar as classes para CDI, a classe `AlunosDao` também sofreria algumas mudanças, como abaixo:

```
@ApplicationScoped // javax.enterprise.context.ApplicationScoped
public class AlunosDao {

}
```

Se você estiver trabalhando com CDI 1.1 você nem mesmo precisaria anotar a classe `AlunosDao` pois por padrão todas as classes na sua aplicação podem ser injetadas pelo CDI sem que você precise anotá-las.

Favoreça o CDI... ou Spring

Embora a especificação do CDI nos permita misturar as anotações do CDI com as anotações JSF, vimos que isso não acaba muito bem. São problemas difíceis de prever e resolver, por esse motivo e muitos outros, de acordo com o Gavin King (líder da especificação CDI), devemos evitar tal prática.

Favoreça o container mais completo e robusto no seu projeto, que normalmente é o CDI ou Spring. Ambos estão passando por constantes melhorias e correções, enquanto os demais containers estão "ficando para trás".

O que tenho observado é que qualquer novidade oriunda de novos frameworks auxiliares e extensões serão dirigidas exclusivamente para o CDI e não mais o JSF. Um bom exemplo são os escopos de conversação auxiliares do DeltaSpike. Embora eles existam para resolver problemas pertinentes ao JSF eles são implementados em cima do CDI.

Enfim, quanto mais tempo postergamos essa migração para CDI ou Spring mais difícil fica. Adotar uma tecnologia para IoC/DI em um novo projeto é uma decisão importante que normalmente dificulta o início do projeto, mas que traz ganhos de produtividade a médio-longo prazo, pois qualquer modificação torna-se barata. Levamos isso tão a sério que nossos cursos mais avançados possuem capítulos tratando do assunto.

E você, utiliza CDI no seu projeto? Já migrou ou pensa em migrar seus managed beans do JSF para CDI?

Rafael Ponte

Desenvolvedor e instrutor na TriadWorks

Posted in: java ee java cdi jsf spring ioc di