Seminar CDI

# Khái niệm CDI (Contexts and Dependency Injection)

CID cho Java EE platform là một trong những tính năng của Java EE 6 giúp kết nồi giữa tầng web và tầng giao dịch của nền tảng Java EE. CDI là một bộ các service được sử dụng cùng nhau để giúp người lập trình dễ dàng sử dụng enterprise beans cùng với công nghệ JavaServer Faces trong một ứng dụng web.

CDI được quy định trong JSR 299, trước đây được gọi là Web Beans.

Các định nghĩa liên quan đến CDI khác như:

* JSR 330, Dependency Injection for Java
* The Managed Beans specification, which is an offshoot of the Java EE 6 platform specification (JSR 316)

# Giới thiệu về JSF (Java Server Faces)

JSF là viết tắt của JavaServer Faces, một bản miêu tả kĩ thuật giúp đơn giản hóa việc phát triển giao diện cho các ứng dụng Web viết bằng Java bằng cách dùng các thành phần dùng lại được (reusable components).

JSF là một bộ khung (framework) phát triển các ứng dụng Web viết bằng Java nhằm làm đơn giản hóa quá trình phát triển giao diện người dùng cho các ứng dụng J2EE. Để tạo ra giao diện hiển thị, JSF dùng dạng cấu trúc cây của các thẻ, mỗi thẻ là một thành phần giao diện (component) và FacesServlet servlet sẽ thực hiện công đoạn chuyển đổi ra giao diện tương ứng cho người dùng với định dạng HTML. Ngoài ra, JSF cũng có thể sử dụng các kĩ thuật hiển thị khác, như XUL. JSF bao gồm:

Một tập các hàm API để biểu diễn các thành phần UI (giao diện người dùng) và quản lí thạng thái của chúng, xử lí các sự kiện và kiểm tra dữ liệu đầu vào, quy định việc di chuyển trang (page navigation), hỗ trợ tính đa ngôn ngữ và hỗ trợ cho người sử dụng (accessibility)

Một số thành phần (component) có sẵn

Hai thư viện chứa các thẻ tùy biến JSP (JavaServer Pages custom tag) để có thể biểu diễn một JavaServer Faces interface bên trong một trang JSP.

Mô hình sự kiện ở phía máy chủ (server-side event model)

Khả năng quản lí trạng thái

Managed Beans (JavaBeans được tạo bằng dependency injection - xem thêm Spring framework)

Bản miêu tả kĩ thuật JSF được quy định trong JSR 127 của Quá trình Cộng đồng Java.

Mục tiêu của JSF

Tạo ra một bộ khung gồm các thành phần giao diện người dùng chuẩn (standard GUI component framework) nhằm giúp cho các công cụ phát triển dễ dàng hơn cho người dùng trong việc tạo GUI chất lượng cao đồng thời quản lí các kết quả của GUI với xử lí thực thi của chương trình.

Định ra một tập các lớp cơ sở của Java (lightweight Java base classes) biểu diễn cho các thành phần UI, trạng thái mỗi thành phần, và các sự kiện đầu vào. Những lớp này sẽ xử lí những vấn đề liên quan đến chu kì sống của GUI, đặc biệt là quản lí trạng thái trong suốt chu trình sống của một trang của thành phần GUI đó.

Cung cấp một tập các thành phần GUI chung, bao gồm các thành phần HTML form input. Những thành phần này sẽ được dẫn xuất từ tập đơn giản các lớp cơ sở (đề cập ở #1) đề từ đó có thể định ra các thành phần mới.

Cung cấp một mô hình JavaBeans để có thể truyền đi (dispatch) các sự kiện từ các GUI controls phía máy khách đến các xử lí hiện thực cụ thể từ phía ứng dụng máy chủ.

Định ra các hàm APIs để kiểm chứng dữ liệu nhập, bao gồm hỗ trợ kiểm chứng từ phía máy chủ.

Chỉ định một mô hình để có thể đa ngôn ngữ hóa hay địa phương hóa các GUI.

Khởi tạo tự động dữ liệu ra phù hợp cho máy khách đích, dựa vào mọi dữ liệu cấu hình ở máy khách đó, bao gồm cả dựa vào phiên bản trình duyệt, ví dụ.

Việc khởi tạo tự động dữ liệu ra còn kèm theo các đòi hỏi về hỗ trợ người dùng (accessibility), được quy định bởi WAI.

# Tại sao phải sử dụng CDI

* Tích hợp với các Expression Language (EL) tạo điều kiện cho việc sử dụng các component trên trang JavaServer Faces hoặc JavaServer Pages.
* Khả năng decorate cho các thành phần đã inject. (giải thích bên dưới).
* Khả năng kết hợp các interceptor với component sử dụng các interceptor bindings một cách an toàn.
* Là một mô hình event-notification.
* Là một scope bổ sung cho các scope có sẵn (request, session, and application) được định nghĩa bới java servlet.
* Là một Service Provider Interface (SPI) hoàn chỉnh, cho phép các framework bên ngoài có thể tích hợp với môi trường Java EE 6 một cách dễ dàng.

Advance:

* Tách riêng giữa server và client bằng các type và qualifier để có thể dễ dàng thay đổi các implement trên server.
* Tách riêng vòng đời của các component có liên quan bằng cách:
* Tạo component theo ngữ cảnh (contextual) với quản lý vòng đời tự động.
* Cho phép các stateful component giao tiếp với nhau tương tự như service, thông qua messages.
* Hoàn toàn tách riêng thành phần tạo ra message và thành phần nhận message thông qua event.
* Tách biệt các giao thức trực tiếp bằng interceptor.

# Liệt kê các annotation

CDI Features

• Basic dependency injection: @Inject, @Qualifier, @Stereotype, @Alternative, @All, @Any.

• Component naming: @Named

• Custom Object Factories: @Produces, @Disposes, InjectionPoint

• Context management @RequestScoped, @SessionScoped, @ConversationScoped, @ApplicationScoped, @Scope

• Lightweight Events: @Event, @Observes

• Interceptors/Decorators: @Interceptor, @InterceptorBinding,

@AroundInvoke, @InvocationContext, @Decorator, @Delegate

• Portable extensions SPI!

# @Decorator

Một decorator là một class được đánh dấu javax.decorator.Decorator và có thành phần decorators tương ứng trong file beans.xml đồng thời còn phải có một delegate injection javax.decorator.Delegate.

Decorator khá giống so với interceptor, tuy nhiên decorator thực hiện các tasks một cách khác biệt so với interceptor. Interceptor không thể thực hiện các business logic trong khi đó decorator thì có thể bằng cách can thiệp vào các business logic của bean. Nghĩa là thay vì sử dụng lại các ứng dụng như interceptor, decorator có các logic cho một ứng dụng cụ thể.

Vd:

@Decorator

public abstract class CoderDecorator implements Coder {

@Inject

@Delegate

@Any

Coder coder;

public String codeString(String s, int tval) {

int len = s.length();

return "\"" + s + "\" becomes " + "\"" + coder.codeString(s, tval)

+ "\", " + len + " characters in length";

}

}

Trong fle beans.xml

<decorators>

<class>decorators.CoderDecorator</class>

</decorators>

Nếu một ứng dụng có cả interceptor và decorator thì interceptor sẽ được gọi trước,

# @Stereotype

Stereotype là một dạng annotation được sử dụng trên bean để kết hợp với các annotation khác. Stereotype khá hữu dụng trong các ứng dụng có nhiều bean thực hiện các chức năng giống nhau. Setreotype quy định các vấn đề sau:

* Một scope mặc định.
* Một hoặc nhiều interceptor binding.
* Một @Name mặc định sử dụng trên EL (không bắt buộc).
* Một @Alternative quy định tất cả bean có stereotype này là alternative (không bắt buộc).

Một bean có stereotype sẽ luôn sử dụng các annotation đã quy định cho stereotype đó, vì thế không cần phải định nghĩa lại các annotation giống nhau cho nhiều bean.

VD: tạo một stereotype có tên là Action:

@RequestScoped

@Secure

@Transactional

@Named

@Stereotype

@Target(TYPE)

@Retention(RUNTIME)

public @interface Action {}

Sau đó sử dụng trên một bean khác như sau:

@Action

public class LoginAction { ... }

Có thể ghi đè một annotation nào đó đã quy định sẵn bởi stereotype:

@SessionScoped

@Action

public class LoginAction { ... }

CDI còn quy định sẵn một stereotype gọi là Model:

@Named

@RequestScoped

@Stereotype

@Target({TYPE, METHOD, FIELD})

@Retention(RUNTIME)

public @interface Model {}