Annotation based Dependency Injection

So imagine a clean slate where we make a new IoC mechanism. We'd mostly just use a few annotations to declare the IoC contract - reusing standards where possible such as JSR 250 for the annotations and JNDI as the access mechanism to the container.

General guidelines for IoC

- the container is invisible and no container specific APIs are required by the component developers or users of the container
- regular Java code can be the container - or some scripting language etc
- allows configuration through some XML marshalling layer; where JAXB 2 could be the default XML configuration mechanism

Mandatory Annotations

These annotations MUST be adhered to by a container.

@Resource (from JSR 250)

Indicate a place in a naming system, such as JNDI where the resource should be fetched from. We assume here that using @Resource marks stuff as mandatory

@PostConstruct (from JSR 250)

This method MUST be called after the constructors and property setters have been called. May throw any exception to indicate that the bean could not be configured properly.

@PreDestroy (from JSR 250)

Called when a bean is no longer required and being destroyed by the container. This method MUST be called by all containers.

Other Annotations

See Other Annotations for more ideas of optional annotations.

Contract summary

The following pseudocode illustrates the containers contract

- initialse the POJO
- call any setters
- check that all the @Resource setters are called - if not fail with an exception
- call the @PostConstruct method
- before termination of the container, call any @PreDestroy method

Example POJO

```
public class Cheese {
  private DataSource dataSource;
  private int timeout = 100;

  @Resource
  public void setDataSource(DataSource ds) {
     this.dataSource = ds;
  }

  public void setTimeout(int timeout) {
     this.timeout = timeout;
  }

  @PostConstruct
  public void start() throws Exception {
     ...

  }
}
```

Example Containers

Java code

```
Cheese c = new Cheese()
c.setDataSource(foo);
c.setTimeout(123); // optional
c.start();
```

JAXB 2

Note that we'd have added an annotation or two from JAXB 2 to achieve the following. Namely adding @XmlIDREF to the setDataSource property

```
<cheese dataSource="customerDb" timeout="456"/>
```

Integrating with existing IoC containers

We should be able to add the lifecycle annotations to any existing lifecycle interfaces we have. e.g.

- in Spring it'd be the InitializingBean and DisposableBean.
- in many libraries there is a Service interface of some kind with start/stop in it.

So by adapting the existing lifecycle interfaces folks have to the DI containers we'd be able to move to ANDI while still supporting existing IoC containers like Spring POJOs etc.

Dealing with legacy code

Lots of code today uses old lifecycle interfaces. Lots of this code has been around for a while and will not be moving to exclusive Java 5 only any

time soon. So supporting a simple way to wire in lifecycle interfaces to AnDI containers would be useful. Here's one suggestion for how we can do it.

- if a POJO has no lifecycle annotations, look on the classpath for text files **META-INF/services/jsr-250/lifecycle** files. If there are any files parse them. They should be of the form

```
interfaceName#methodName = PostConstruct
interfaceName#methodName = PreDestroy
```

e.g. to support any Spring POJO just ensure the following is on the classpath.

```
org.springframework.beans.factory.InitializingBean#afterPropertiesSet = PostConstruct
org.springframework.beans.factory.DisposableBean#destroy = PreDestroy
```

Exposing the IoC container to Java code

A natural way to expose the IoC container to Java code is via a JNDI provider. That way folks can write to the JNDI standard to look up POJOs in the initial context, nor navigate to child contexts without using an IoC container-specific API.

Issues with JSR 250

- currently JSR 250 says that @PostConstruct and @PreDestroy cannot throw exceptions. Pretty much every example we could find do throw checked exceptions. e.g. Spring, Pico, ActiveMQ, GBeans etc. So we'd like that restriction removed so that the methods can throw any exceptions they wish. Afterall - its easy for the container to catch any exceptions (and they usually do anyway) - its more work for the component developer to catch & wrap as runtime exceptions.
- we are currently using @Resource to indicate a kind of *@Mandatory*; namely that anything marked with @Resource is a mandatory property/field. Ideally we'd have preferred @Mandatory as this purely denotes mandatory versus optional semantics and does not imply looking up of resources in a naming context, but I guess @Resource is an OK substitute for now?

Note that lifecycle annotations like @PostConstruct and @PreDestroy can be used on methods on an interface (such as in Spring's InitializingBean and DisposableBean) and then be inherited on any POJO.

Credits

Many thanks to Hani Suleiman for help creating this document and giving feedback from the JSR 250 expert group.