**Resource Injection**

The javax.annotation.Resource annotation is used to declare a reference to a resource.@Resource can decorate a class, a field, or a method. The container will inject the resource referred to by @Resource into the component either at runtime or when the component is initialized, depending on whether field/method injection or class injection is used. With field and method-based injection, the container will inject the resource when the application is initialized. For class-based injection, the resource is looked up by the application at runtime.

@Resource has the following elements:

* name: The JNDI name of the resource
* type: The Java language type of the resource
* authenticationType: The authentication type to use for the resource
* shareable: Indicates whether the resource can be shared
* mappedName: A non-portable, implementation-specific name to which the resource should be mapped
* description: The description of the resource

The name element is the JNDI name of the resource, and is optional for field- and method-based injection. For field-based injection, the default name is the field name qualified by the class name. For method-based injection, the default name is the JavaBeans property name based on the method qualified by the class name. The name element must be specified for class-based injection.

The type of resource is determined by one of the following:

* The type of the field the @Resource annotation is decorating for field-based injection
* The type of the JavaBeans property the @Resource annotation is decorating for method-based injection
* The type element of @Resource

For class-based injection, the type element is required.

The authenticationType element is used only for connection factory resources, and can be set to one of the javax.annotation.Resource.AuthenticationType enumerated type values:CONTAINER, the default, and APPLICATION.

The shareable element is used only for ORB instance resources or connection factory resource. It indicates whether the resource can be shared between this component and other components, and may be set to true, the default, or false.

The mappedName element is a non-portable, implementation-specific name that the resource should be mapped to. Because the name element, when specified or defaulted, is local only to the application, many Java EE servers provide a way of referring to resources across the application server. This is done by setting the mappedName element. Use of the mappedName element is non-portable across Java EE server implementations.

The description element is the description of the resource, typically in the default language of the system on which the application is deployed. It is used to help identify resources, and to help application developers choose the correct resource.

**Field-Based Injection**

To use field-based resource injection, declare a field and decorate it with the @Resource annotation. The container will infer the name and type of the resource if the name and type elements are not specified. If you do specify the type element, it must match the field’s type declaration.

package com.example;

public class SomeClass {

@Resource

private javax.sql.DataSource myDB;

...

}

In the code above, the container infers the name of the resource based on the class name and the field name: com.example.SomeClass/myDB. The inferred type isjavax.sql.DataSource.class.

package com.example;

public class SomeClass {

@Resource(name="customerDB")

private javax.sql.DataSource myDB;

...

}

In the code above, the JNDI name is customerDB, and the inferred type isjavax.sql.DataSource.class.

**Method-Based Injection**

To use method-based injection, declare a setter method and decorate it with the @Resourceannotation. The container will infer the name and type of the resource if the name and type elements are not specified. The setter method must follow the JavaBeans conventions for property names: the method name must begin with set, have a void return type, and only one parameter. If you do specify the type element, it must match the field’s type declaration.

package com.example;

public class SomeClass {

private javax.sql.DataSource myDB;

...

@Resource

private void setMyDB(javax.sql.DataSource ds) {

myDB = ds;

}

...

}

In the code above, the container infers the name of the resource based on the class name and the field name: com.example.SomeClass/myDB. The inferred type isjavax.sql.DataSource.class.

package com.example;

public class SomeClass {

private javax.sql.DataSource myDB;

...

@Resource(name="customerDB")

private void setMyDB(javax.sql.DataSource ds) {

myDB = ds;

}

...

}

In the code above, the JNDI name is customerDB, and the inferred type isjavax.sql.DataSource.class.

**Class-Based Injection**

To use class-based injection, decorate the class with a @Resource annotation, and set the requiredname and type elements.

@Resource(name="myMessageQueue",

type="javax.jms.ConnectionFactory")

public class SomeMessageBean {

...

}

**Declaring Multiple Resources**

The @Resources annotation is used to group together multiple @Resource declarations for class-based injection.

@Resources({

@Resource(name="myMessageQueue",

type="javax.jms.ConnectionFactory"),

@Resource(name="myMailSession",

type="javax.mail.Session")

})

public class SomeMessageBean {

...

}

The code above shows the @Resources annotation containing two @Resource declarations. One is a JMS message queue, and the other is a JavaMail session.