Seasar2.3

Yasuo Higa

Seasar Foundation/Chief Committer

http://www.seasar.org/en

Translated by:H.Ozawa

I want to ask this question to
every developer in the world,

DO YOU REALLY WANT TO WRITE CONFIGURATION FILES?

Seasar2 saves you from the XML hell

Translated by:H.Ozawa

Agenda

- Why DI was conceived
- Problem with the current DI implementation (Spring)
- Path to the next generation DI(Seasar2)
- Seasar2 VS EJB3

Translated by:H.Ozawa

Before DI

- Unfulfilled dream of components
- The promise of a component
  - enable developers to just combine “black box” components to build an application

Translated by:H.Ozawa

Merit and Demerit of a Component

- Merit
  - Components can be reused
  - Compenents can be easily combined
- Demerit
  - Components must be developed in conformance to on “special” rules (implement set of APIs)
    - Lock in from using “special” APIs
    - Components conforming to different APIs may not be able to work together

Translated by:H.Ozawa

Light and Darkness of Components

- Light
  - ActiveX is seeing success in the GUI world
- Darkness
  - Not too successful in application development world
    - Not many components can be used
    - Cost of favouring one API implementation is too high
    - Need to connect differing systems together

Translated by:H.Ozawa

Example of a Failure

EJB

Translated by:H.Ozawa

Problems with EJB (SessionBean)

- Too many files are necessary to build just one component
  - 2 interfaces
  - 1 implementation class
  - Configuration files
- Gathering everything and deploying to an application server is too much of a hassle
- Redeploying each time there is a modification is very tiring

Translated by:H.Ozawa

Problems with EJB (SessionBean)

- Testing is difficult because a component must run on an application server
- API is difficult and requires too much time to learn

Translated by:H.Ozawa

Disastrous State of EJB

- Many developers tried using it because it is a “standard”

  But, most did not overcome the hardship and abandoned it

Translated by:H.Ozawa

Why DI was Conceived

- As a replacement of EJB
- Resolves following EJB problems
  - No need to implement proprietary APIs
  - No need to deploy
  - Able to run without an application server

Translated by:H.Ozawa

Concept of DI

- POJO(Plain Old Java Object)
  - Not dependent of APIs
    - Improved reusability
    - No need to learn APIs
    - Able to run without an application server
    - Testing is easier

Concept of DI

- DIContainer resolves dependencies between objects
  - Each object defines interface of type deploy and does not depend of a class implementation
    - Objects are more decoupled resulting in better maintainability and reusability
    - Easier testing because implementation can easily be exchanged with a Mock
  - DIContainer instantiates objects and resolves dependencies during runtime
    - Dependecies are often defined in a XML file

Sample to Demonstrate Concept of DI

- Greeting
  - Returns greeting String
- Greeting client class
  - Output message from a Greeting class
- Greeting execution class
  - Combines Greeting class with Greeting client class

Translated by:H.Ozawa

Greeting.java

```
package examples.di;

public interface Greeting {
    String greet();
}
```


Translated by:H.Ozawa

GreetingImpl.java

```
package examples.di.impl;

import examples.di.Greeting;

public class GreetingImpl implements Greeting {

    public String greet() {
        return "Hello World!";
    }
}
```


Translated by:H.Ozawa

GreetingClient.java

```
package examples.di;

public interface GreetingClient {

    void execute();
}
```


Translated by:H.Ozawa

GreetingClientImpl.java

```
package examples.di.impl;

import examples.di.Greeting;
import examples.di.GreetingClient;

public class GreetingClientImpl implements GreetingClient {

    private Greeting greeting;

    public void setGreeting(Greeting greeting) {
        this.greeting = greeting;
    }

    public void execute() {
        System.out.println(greeting.greet());
    }
}
```


Translated by:H.Ozawa

beans.xml (Spring)

```
<beans>
    <bean id="greeting"
        class="examples.di.impl.GreetingImpl"/>
    <bean id="greetingClient"
        class="examples.di.impl.GreetingClientImpl">
        <property name="greeting">
            <ref bean="greeting"/>
        </property>
    </bean>
</beans>
```


Translated by:H.Ozawa

GreetingMain.java (Spring)

```
package examples.di.main;

import …;

public class GreetingMain {

    public static void main(String[] args) {
        ClassPathResource res =
            new ClassPathResource("beans.xml");
        XmlBeanFactory factory = new XmlBeanFactory(res);
        GreetingClient greetingClient = (GreetingClient)
            factory.getBean("greetingClient");
        greetingClient.execute();
    }
}
```

Points to Remember from this Sample

- Class that uses the function (GreetingClientImpl)
  - deployment type is declared in the interface (Greeting) of a class that provides the function
  - is not dependent on implementation of a class (GreetingImpl)
- DI Configuration File (beans.xml)
  - hads component declaration and DI information

DI FAQ:Question 1

- Is interface a necessity?
  - No. DI does not require developers to create an interface. Using an interface, however, is strongly recommended.

Translated by:H.Ozawa

DI FAQ:Question 2

- Why is it better to use an interface?
  - Because if specification (interface) is decided on, it is not necessary to be consciousness about the actual implementation
    - Unit test can be easily done by exchanging the implementation with a Mock
    - Concurrent development can be done more smoothly because Mock can be used instead of classes that is not yet developed

Translated by:H.Ozawa

DI FAQ:Question 3

- Isn't it a hassle to think about the interface from the start?
  - Implementation should not begin before specification is set
  - It's like wandering without knowing where to go. Many problems are caused by this
  - If specification is decided on, it shouldn't be too difficult to decide on interfaces

Translated by:H.Ozawa

DI FAQ:Question 4

- Doesn't it conflict with XP YAGNI to first decide on a specification?
  - Should take caution against extensive design to avoid YAGNI (You Arent Gonna Need It)
  - Specification to be decided should be based on what is **currently required**, not on what may be required

Translated by:H.Ozawa

DI FAQ:Question 5

- Isn't it OK to simply just implement the classes if the specification is decided first?
  - As answered in question 2, there are merits to using interfaces
  - If specification is decided on, creating interfaces do not require too much time.
  - Unless one person develops everything and that person do all the maintenance, there is more benefit to gain by taking a little bit of time to create interfaces

Translated by:H.Ozawa

DI FAQ:Question 6

- Doesn't it cause more complication because it becomes more difficult to trace class implementation from the source code?
  - If the specifications are clear, this shouldn't be a problem.
  - Knowing **what** the class do is what's important – it's not **how** that's important
  - It's important to decouple component independent of implementation. Benefits include better maintainability and reusablity

Translated by:H.Ozawa

DI FAQ:Question 7

- It's a hassle to write DI configuration in XML files?
  - You're right
  - This is the main problem with current DI implementation (Spring)

Translated by:H.Ozawa

Problem with the Current DI Implementation (Spring)

- XML Hell
  - As the number of components increase, number of XML files also increase leading to the entrance of XML Hell

Translated by:H.Ozawa

- Less Configuration
  - Decrease number of necessary configuration files
  - But how?

Less Configuration – Point 1

- Convention over Configuration
  - Develop according to a convention, and let the framework will do most of the configuration

Translated by:H.Ozawa

Convention over Configuration – Example 1

- Convention
  - Define property type in an interface
- Auto Configuration by the S2 framework
  - If property type is an interface and there is an object that implements this interface, dependecy is automatically configured
  - Trying to automatically configure every type is dangerous but by limiting automatic configuration to just an interface, it works in **most circumstances**

Translated by:H.Ozawa

Convention over Configuration – Example 2

- Convention
  - Name implementation class **Xxx**Impl when interface name is **Xxx**
- Auto Configuration by the S2 framework
  - Recursively search within a package for class names ending with string “Impl” and automatically register all such classes in a S2Container

Translated by:H.Ozawa

Result of Convention over Configuration

- Component definition is unnecessary
- DI configuration is unnecessary

Translated by:H.Ozawa

beans.dicon (Seasar2)

```
<components>
    <component
      class="...FileSystemComponentAutoRegister">
        <initMethod name="addClassPattern">
            <arg>"examples.di.impl"</arg>
            <arg>".*Impl"</arg>
        </initMethod>
        <initMethod name="registAll"/>
    </component>
</components>
```


Translated by:H.Ozawa

Less Configuration – Point 2

- Configuration by Exception
  - Decide on a default value. Use this value when value is not specified
  - If the default value is not appropriate, explicitly set a value
  - Use the principle of Convention over Configuration and avoid explicitly specifying a value as much as possible

Translated by:H.Ozawa

Less Configuration – Point 2

- Configuration by Exception
  - Use annotation to configure
  - Annotation is seen to be easier than XML because it is nearer to the source code

Translated by:H.Ozawa

Example of Configuration by Exception

```
// Explicit specifying “hoge2”
@Binding("hoge2")
public void setHoge(Hoge hoge) {
    this.hoge = hoge;
}

//Specifying not to automatically bind
@Binding(bindingType=BindingType.NONE)
public void setHoge(Hoge hoge) {
    …;
}
```


Translated by:H.Ozawa

3 Types of Annotation

```
//Tiger annotation
@Binding("hoge2")
public void setHoge(Hoge hoge) {
    this.hoge = hoge;
}

//bacckport175 annotation(works with JDK1.4K)
/**
  * @...backport175.Binding("hoge2")
  */
public void setHoge(Hoge hoge) {
    this.hoge = hoge;
}

//constant annotation
public static final String hoge_BINDING = "hoge2";
```


Translated by:H.Ozawa

What is Less Configured?

- Component declaration and DI configuration
  - Writing configuration files is a hassle and liable to produce an error
  - So, try to avoid writing configuration file as much as possible
- Parameters dependent on an environment
  - Parameter like database connection string is dependent on an environment
  - So, they should be specified in a configuration file

Translated by:H.Ozawa

Improvements in EJB3

- It's POJO based
- Configuration can be done by annotation
  - Supports Configuration by Exception

Translated by:H.Ozawa

Weakness of EJB3 over Seasar2

- Concept of Convention over Configuration is not supported so some annotation are still required
- Hassle to deploy
- Testing without a Mock can only be done on an application server after deploying to it
- AOP support is weak

Translated by:H.Ozawa

Highlight of Seasar2.4

- Capabilities to monitor automatic and manual S2Container configuration from a web

Translated by:H.Ozawa

Summary

- EJB failed in building components
- DI is more friendly because it is based on POJO
- Current DI implementation (Spring) leads developers to gates of XML Hell as the application becomes larger
- Next generation DI implementation (Seasar2) is available to avoid XML Hell by using Less Configuration concept

Translated by:H.Ozawa