Introduction to Spring Framework and Dependency Injection

Aaron Zeckoski
azeckoski@gmail.com

Sakai Montreal CRIM Workshop

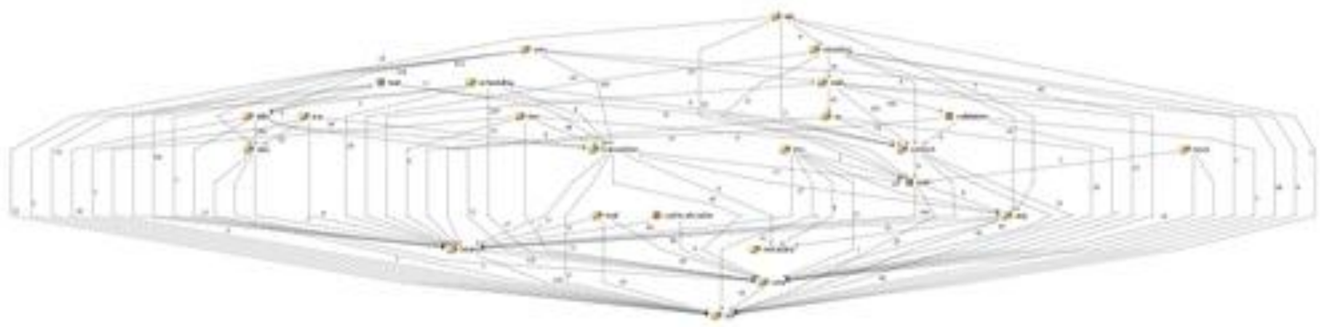
Sakai Programmer's Café

Spring Framework

- A popular and stable Java application framework for enterprise development
  - Ubiquitous for Java development
  - Well established in enterprise Java apps
  - Time tested and proven reliable
- A primary purpose is to reduce dependencies and even introduce negative dependencies
  - Different from almost every other framework out there
  - Part of the reason it has been adopted so quickly

URL: http://www.springframework.org/

Spring code structure



- Spring code base is proven to be well structured (possibly the best)
  - http://chris.headwaysoftware.com/2006/07/springs_structu.html
    - Analysis using Structure 101
- 139 packages
- No dependency cycles

More Spring

- Considered an alternative / replacement for the Enterprise JavaBean (EJB) model
- Flexible
  - Programmers decide how to program
- Not exclusive to Java (e.g. .NET)
- Solutions to typical coding busywork
  - JDBC
  - LDAP
  - Web Services

URL: http://en.wikipedia.org/wiki/Spring_framework

What does Spring offer?

- Dependency Injection
  - Also known as IoC (Inversion of Control)
- Aspect Oriented Programming
  - Runtime injection-based
- Portable Service Abstractions
  - The rest of spring
    - ORM, DAO, Web MVC, Web, etc.
    - Allows access to these without knowing how they actually work

Dependency Injection defined

- Method to create needed dependencies or look them up somehow without doing it in the dependent code
  - Often called Inversion of Control (IoC)
- IoC injects needed dependencies into the object instead
  - Setters or Contructor
- Primary goal is reduction of dependencies in code
  - an excellent goal in any case
  - This is the central part of Spring

URL: http://en.wikipedia.org/wiki/Inversion_of_Control

Aspect Oriented Programming defined

- Attempts to separate concerns, increase modularity, and decrease redundancy
  - Separation of Concerns (SoC)
    - Break up features to minimize overlap
  - Don't Repeat Yourself (DRY)
    - Minimize code duplication
  - Cross-Cutting Concerns
    - Program aspects that affect many others (e.g. logging)
- AspectJ is the top AOP package
  - Java like syntax, IDE integration

URL: http://en.wikipedia.org/wiki/Aspect-oriented_programming

Portable Service Abstractions defined

- Services that easily move between systems without heavy reworking
  - Ideally easy to run on any system
  - Abstraction without exposing service dependencies
    - LDAP access without knowing what LDAP is
    - Database access without typical JDBC hoops
- Basically everything in Spring that is not IoC or AOP

Sakai

What is a bean?

- Typical java bean with a unique id
- In spring there are basically two types
  - Singleton
    - One instance of the bean created and referenced each time it is requested
  - Prototype (non-singleton)
    - New bean created each time
    - Same as **new** ClassName()
- Beans are normally created by Spring as late as possible

What is a bean definition?

- Defines a bean for Spring to manage
  - Key attributes
    - class (required): fully qualified java class name
    - id: the unique identifier for this bean
    - *configuration*: (singleton, init-method, etc.)
    - constructor-arg: arguments to pass to the constructor at creation time
    - property: arguments to pass to the bean setters at creation time
    - Collaborators: other beans needed in this bean (a.k.a dependencies), specified in property or constructor-arg
- Typically defined in an XML file

Sample bean definition

```
<bean id="exampleBean" class=”org.example.ExampleBean">
  <property name="beanOne"><ref bean="anotherExampleBean"/></property>
  <property name="beanTwo"><ref bean="yetAnotherBean"/></property>
  <property name="integerProperty"><value>1</value></property>
</bean>
```

```
public class ExampleBean {
        private AnotherBean beanOne;
        private YetAnotherBean beanTwo;
        private int i;
        public void setBeanOne(AnotherBean beanOne) {
                this.beanOne = beanOne; }
        public void setBeanTwo(YetAnotherBean beanTwo) {
                this.beanTwo = beanTwo; }
        public void setIntegerProperty(int i) {
                this.i = i; }
        …
}
```

What is a bean factory?

- Often seen as an ApplicationContext
  - BeanFactory is not used directly often
  - ApplicationContext is a complete superset of bean factory methods
    - Same interface implemented
    - Offers a richer set of features
- Spring uses a BeanFactory to create, manage and locate “beans” which are basically instances of a class
  - Typical usage is an XML bean factory which allows configuration via XML files

How are beans created?

- Beans are created in order based on the dependency graph
  - Often they are created when the factory loads the definitions
  - Can override this behavior in bean

```
<bean class="className" lazy-init="true" />
```

  - You can also override this in the factory or context but this is not recommended
- Spring will instantiate beans in the order required by their dependencies
  1. app scope singleton - eagerly instantiated at container startup
  2. lazy dependency - created when dependent bean created
  3. VERY lazy dependency - created when accessed in code

How are beans injected?

- A dependency graph is constructed based on the various bean definitions
- Beans are created using constructors (mostly no-arg) or factory methods
- Dependencies that were not injected via constructor are then injected using setters
- Any dependency that has not been created is created as needed

Multiple bean config files

- There are 3 ways to load multiple bean config files (allows for logical division of beans)
  - Load multiple config files from web.xml

```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>classpath:/WEB-INF/spring-config.xml,
   classpath:/WEB-INF/applicationContext.xml</param-value>
</context-param>
```

  - Use the import tag

```
<import resource="services.xml"/>
```

  - Load multiple config files using Resources in the application context constructor
    - Recommended by the spring team
    - Not always possible though

```
ClassPathXmlApplicationContext appContext = new
   ClassPathXmlApplicationContext( new String[]
   {"applicationContext.xml",
   "applicationContext-part2.xml"});
```

Sakai

Bean properties?

- The primary method of dependency injection
- Can be another bean, value, collection, etc.

```
<bean id="exampleBean" class="org.example.ExampleBean">
        <property name="anotherBean">
        <ref bean="someOtherBean" />
        </property>
</bean>
```

- This can be written in shorthand as follows

```
<bean id="exampleBean" class="org.example.ExampleBean">
        <property name="anotherBean" ref="someOtherBean" />
</bean>
```

Anonymous vs ID

- Beans that do not need to be referenced elsewhere can be defined anonymously
- This bean is identified (has an id) and can be accessed to inject it into another bean

```
<bean id="exampleBean" class="org.example.ExampleBean">
      <property name="anotherBean" ref="someOtherBean" />
</bean>
```

- This bean is anonymous (no id)

```
<bean class="org.example.ExampleBean">
      <property name="anotherBean" ref="someOtherBean" />
</bean>
```

What is an inner bean?

```
<bean id="outer" class="org.example.SomeBean">
    <property name="person">
        <bean class="org.example.PersonImpl">
            <property name="name"><value>Aaron</value></property>
            <property name="age"><value>31</value></property>
        </bean>
    </property>
</bean>
```

- It is a way to define a bean needed by another bean in a shorthand way
  - Always anonymous (id is ignored)
  - Always prototype (non-singleton)

Bean init-method

- The init method runs AFTER all bean dependencies are loaded
  - Constructor loads when the bean is first instantiated
  - Allows the programmer to execute code once all dependencies are present

```
<bean id="exampleBean" class=”org.example.ExampleBean"
        init-method=”init” />
```

```
public class ExampleBean {
        public void init() {
                // do something
        }
}
```

Bean values

- Spring can inject more than just other beans
- Values on beans can be of a few types
  - Direct value (string, int, etc.)
  - Collection (list, set, map, props)
  - Bean
  - Compound property

Example of injecting a string value

```
<bean class="org.example.ExampleBean">
    <property name="email">
        <value>azeckoski@gmail.com</value>
    </property>
</bean>
```

Abstract (parent) beans

- Allows definition of part of a bean which can be reused many times in other bean definitions

```
<bean id="abstractBean" abstract="true"
       class="org.example.ParentBean">
    <property name="name" value="parent-AZ"/>
    <property name="age" value="31"/>
</bean>

<bean id="childBean"
       class="org.example.ChildBean"
       parent="abstractBean" init-method="init">
    <property name="name" value="child-AZ"/>
</bean>
```

- *The parent bean defines 2 values (name, age)*
- *The child bean uses the parent age value (31)*
- *The child bean overrides the parent name value (from parent-AZ to child-AZ)*
- *Parent bean could not be injected, child could*

Sakai

AOP in Spring

- Provides way to create declarative services and custom aspects
- Transaction management is the most common aspect (or concern)
- Spring handles AOP via advisors or interceptors
  - Interception point is a *joinpoint*
  - A set of joinpoints are called a *pointcut*
    - pointcuts are key to Spring AOP, they allow intercepts without explicit knowledge of the OO hierarchy
  - Action taken by an interceptor is called *advice*

AOP advice types

- Around
  - Most common and powerful
  - Execute code before and after joinpoint
- Before
  - Executes before joinpoint, cannot stop execution
- Throws
  - Executes code if exception is thrown
- After return
  - Executes code after normal joinpoint execution

Spring AOP key points

- Pure java implementation
- Allows method interception
  - No field or property intercepts yet
- AOP advice is specified using typical bean definitions
  - Closely integrates with Spring IoC
- Proxy based AOP
  - J2SE dynamic proxies or CGLIB proxies
- Not a replacement for AspectJ

Example transaction proxy

- This wraps a transaction interceptor around a DAO

```
<bean id="daoBeanTarget" class="org.example.dao.impl.DaoImpl">
<property name="sessionFactory"><ref bean="mySessionFactory"/></property>
</bean>

<bean id="daoBean"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
   <property name="transactionManager" ref="transactionManager"/>
   <property name="target" ref="daoBeanTarget"/>
   <property name="transactionAttributes">
     <props>
        <prop key="*">PROPAGATION_REQUIRED</prop>
     </props>
   </property>
</bean>
```

Working example

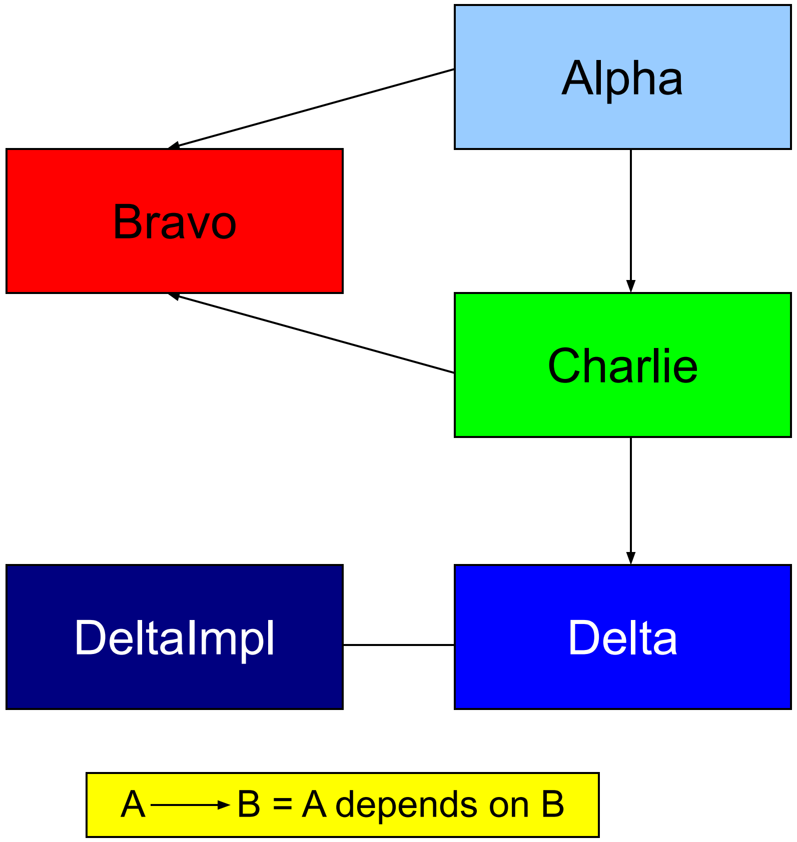
- Let's look at some example code pre and post spring
  - Simple application that allows a user to add, remove, and list a set of strings
- Pre spring code
  - Programmers Cafe - Example App
- Post spring code
  - Programmers Cafe - Example App Spring

Example App

- The example app is a simple command line Java app which is meant to demonstrate a reasonable dependency structure
- This app allows a user to save, delete, and list a set of strings associated with their username

Example App Structure

- Alpha is the main class
- Bravo handles user interaction
- Charlie handles application logic
- Delta handles data access
- Dependency graph is non-cyclical
  - No A => B => C => A



Non-spring version

- Involves using new to create needed dependencies
- Each class must know about the dependencies that it needs
- Singletons have to be created and handed to the classes that need them at the same time or you need a static way to access them (or a framework)
- Tightly coupled code structure

Spring version

- No more new use
- Classes only have to know about the interface
  - or class if no interface available
- Singletons easy to handle
- Loose coupling allows flexible changes

Questions?

- Spring framework
  - http://www.springframework.org/