Service Layer

Andy

1

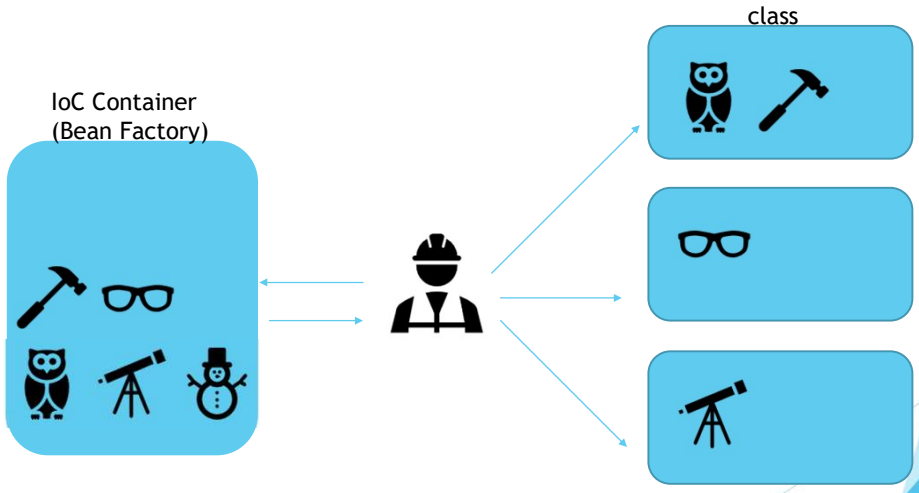
Different ways to get reference

```
public class Controller{

    public List<Student> getStudents(){
        StudentHelper helper = new StudentHelper();
        return helper.getAllStudents();
    }

    public void updateStudent(int id, String name){
        StudentHelper helper = new StudentHelper();
        helper.update(id, name);
    }

    public void addStudent(Student s){
        StudentHelper helper = new StudentHelper();
        helper.createNewStudent(s);
    }

    public void deleteStdent(int id){
        StudentHelper helper = new StudentHelper();
        helper.delete(id);
    }
}
```

```
public class Controller{

    @Autowired
    private StudentHelper helper;

    public List<Student> getStudents(){
        return helper.getAllStudents();
    }

    public void updateStudent(int id, String name){
        helper.update(id, name);
    }

    public void addStudent(Student s){
        helper.createNewStudent(s);
    }

    public void deleteStdent(int id){
        helper.delete(id);
    }
}
```

2

Spring IoC (Inversion of Control)



3

Spring Dependency Injection

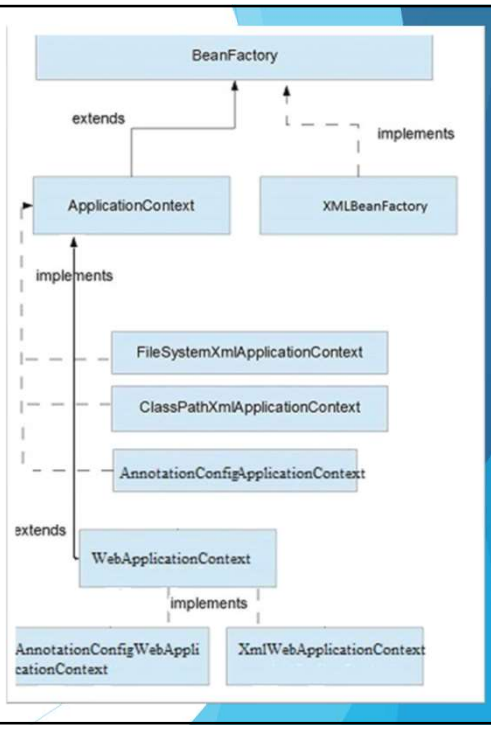
- Bean Creation
  - Container: ApplicationContext vs BeanFactory
  - Bean Initialization --- XML vs Annotation
  - Bean Scope
- Bean Injection
  - Bean Injection --- XML vs Annotation

4

ApplicationContext vs BeanFactory

- **BeanFactory**
  - Bean instantiation/wiring
- **ApplicationContext**
  - Bean instantiation/wiring
  - Automatic BeanPostProcessor registration
  - Automatic BeanFactoryPostProcessor registration
  - Convenient MessageSource access (for i18n)
  - ApplicationEvent publication

Example: https://gist.github.com/bigme666/5210042



5

Bean Creation – XML vs Annotation

```
<bean id="studentHelper"
        name="myStudentHelperBean"
        class="com.demo.service.StudentHelper">
 </bean>

<alias name="studentHelper"  alias="studentHelperAlias"> </alias>
```

```
@Configuration
public class HelperManagement {

        @Bean
        public StudentHelper studentHelper() {
                return new StudentHelper();
        }
}
```

6

Bean Creation – XML vs Annotation

```java
@Configuration
public class MigratedConfiguration {

    @Bean
    public JButton button() {
        return new JButton("Hello World");
    }

    @Bean
    public JButton anotherButton(Icon icon) {
        return new JButton(icon);
    }

    @Bean
    public Icon icon() throws MalformedURLException {
        URL url = new URL("http://morevaadin.com/assets/images/learning_vaadin_cover.png");
        return new ImageIcon(url);
    }
}
```

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">
    <bean id="button" class="javax.swing.JButton">
        <constructor-arg value="Hello World" />
    </bean>
    <bean id="anotherButton" class="javax.swing.JButton">
        <property name="icon" ref="icon" />
    </bean>
    <bean id="icon" class="javax.swing.ImageIcon">
        <constructor-arg>
            <bean class="java.net.URL">
                <constructor-arg value="http://morevaadin.com/assets/images/learning_
            </bean>
        </constructor-arg>
    </bean>
</beans>
```

Question: XML or Annotation?

7

A special bean

- Data Source

```xml
<bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <!-- results in a setDriverClassName(String) call -->
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost:3306/mydb"/>
    <property name="username" value="root"/>
    <property name="password" value="masterkaoli"/>
</bean>
```

8

Bean Scope

Scope	Description
singleton	(Default) Scopes a single bean definition to a single object instance for each Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request. That is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring `ApplicationContext`.
session	Scopes a single bean definition to the lifecycle of an HTTP `Session`. Only valid in the context of a web-aware Spring `ApplicationContext`.
application	Scopes a single bean definition to the lifecycle of a `ServletContext`. Only valid in the context of a web-aware Spring `ApplicationContext`.
websocket	Scopes a single bean definition to the lifecycle of a `WebSocket`. Only valid in the context of a web-aware Spring `ApplicationContext`.

- ** The **Delegate** Mechanism
- Question: Is Spring Singleton Bean thread-safe?

9

Bean Scope

- Implementation

Annotation:

@Scope("prototype")

@Bean

public User getUser(){}

XML:

<bean id="user" class="example.User" scope="prototype"/>

10

Dependency Injection : **Constructor-based**

- Annotation:

```
public class StudentManagement{

    private StudentHelper helper;

    @Autowired
    public StudentManagement(StudentHelper helper, String flow){
        this.helper = helper;
        this.flow = flow;
    }
}
```

- XML:

```
<beans>
    <bean id="studentManagement" class="com.demo.controller.StudentManagement">
        <constructor-arg ref="helper"/>
        <constructor-arg type="java.lang.String" value="new"/>
    </bean>

    <bean id="helper" class="com.demo.service.StudentHelper">
    </bean>
</beans>
```

11

Dependency Injection : **Setter-based**

- Annotation:

```
public class StudentController{

    private StudentHelper helper;

    @Autowired
    public void setStudentHelper(StudentHelper helper){
        this.helper = helper;
    }

    public StudentHelper getStudentHelper(){ //NOT on Getter
        return this.helper;
    }
}
```

- XML:

```
<beans>
    <bean id="studentController" class="com.demo.controller.StudentController">
        <property name="helper" ref="studentHelper"/>
    </bean>

    <bean id="studentHelper" class="com.demo.service.StudentHelper">
    </bean>
</beans>
```

12

Dependency Injection : **Field-based**

- Annotation:

```
public class StudentController{

    @Autowired
    private StudentHelper helper;

    public void setStudentHelper(StudentHelper helper){
        this.helper = helper;
    }

    public StudentHelper getStudentHelper(){ //NOT on Getter
        return this.helper;
    }
}
```

- XML: Not Supported

**Note: Field-based is based on Java reflection, not Spring supported DI type*

13

Constructor-based vs Setter-based

Circular Dependency

```
public class Circular1{

    @Autowired
    public Circular1(Circular2 c2){}
}


public class Circular2{

    @Autowired
    public Circular2(Circular1 c1){}
}
```

Constructor-based

BeanCurrentlyInCreationException

```
public class Circular1{

    @Autowired
    Circular2 c2;
}


public class Circular2{

    @Autowired
    Circular1 c1;
}
```

Setter-based

No exception

14

How does @Autowired work

MODE:

- No -- default
- byType
- byName
- Constructor

Setter

```
public class Example Bean {

    private AnotherBean beanOne;

    private int i;

    @Autowired
    public void setBeanOne(AnotherBean beanOne){
        this.beanOne = beanOne;
    }
}
```

Constructor

```
public class Example Bean {

    private AnotherBean beanOne;

    private int i;

    @Autowired
    public Example(AnotherBean beanOne){
        this.beanOne = beanOne;
    }
}
```

Field

```
public class Example Bean {


    @Autowired
    private AnotherBean beanOne;

    private int i;

    //other methods
}
```

15

@Autowired + @Qualifier

```
<bean id="CustomerBean" class="com.mkyong.common.Customer">
    <property name="action" value="buy" />
    <property name="type" value="1" />
</bean>

<bean id="PersonBean1" class="com.mkyong.common.Person">
    <property name="name" value="mkyong1" />
    <property name="address" value="address 1" />
    <property name="age" value="28" />
</bean>

<bean id="PersonBean2" class="com.mkyong.common.Person">
    <property name="name" value="mkyong2" />
    <property name="address" value="address 2" />
    <property name="age" value="28" />
</bean>
```

```
public class Customer
{
    @Autowired
    @Qualifier("PersonBean1")
    private Person person;
    private int type;
    private String action;
    //getter and setter methods
}
```

Fallback: When byType finds multiple match, it will by default use byName

16

@Component & ComponentScan

```java
@Component
public class FactoryMethodComponent {

    @Bean
    @Qualifier("public")
    public TestBean publicInstance() {
        return new TestBean("publicInstance");
    }

    public void doWork() {
        // Component method implementation omitted
    }
}
```

```java
@Repository
public class JpaMovieFinder implements MovieFinder {
    // implementation elided for clarity
}
```

```java
@Service
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Autowired
    public SimpleMovieLister(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <context:component-scan base-package="org.example"/>

</beans>
```

```java
@Configuration
@ComponentScan(basePackages = "org.example")
public class AppConfig  {
    ...
}
```

Question? @Component vs @Bean ?

17

@PropertySource

```java
@Configuration
@PropertySource("classpath:/com/${my.placeholder:default/path}/app.properties")
public class AppConfig {

    @Autowired
    Environment env;

    @Bean
    public TestBean testBean() {
        TestBean testBean = new TestBean();
        testBean.setName(env.getProperty("testbean.name")); ←
        return testBean;
    }
}
```

18

End of IoC Container

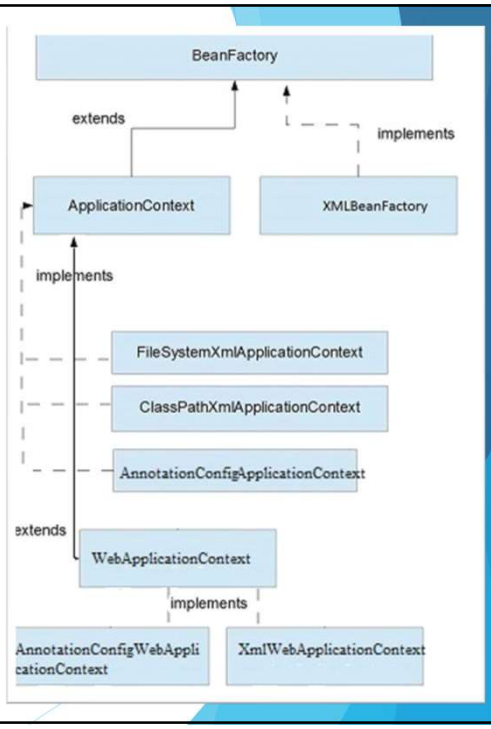
19

*ApplicationContext vs BeanFactory

- Example - How to Use

```
ApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml")
HelloBean hello = (HelloBean) ctx.getBean("hello");
hello.sayHello("John")
```

```
BeanFactory factory = new XmlBeanFactory(new ClassPathResource("beans.xml")
HelloBean hello = (HelloBean) factory.getBean("hello")
hello.getMessage();
```



20