Spring Container

CS544: Enterprise Architecture

Spring Basics

- In this module we are going start by looking at a basic hello world spring application. Then we will look into the details of:
  - The Spring Application Context
  - Spring Bean initialization
  - Spring Bean lifecycle methods
- We are going to take a look at the basic outer layer of spring, the application context, and the configuration of beans. (Life is found in layers).

A basic Spring application

```
package module2.helloworld;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Application {
  public static void main(String[] args) {
    ApplicationContext context = new
                    ClassPathXmlApplicationContext("module2/helloworld/springconfig.xml");
    CustomerService customerService = context.getBean("customerService", CustomerService.class);
    customerService.sayHello();
  }
}
```

Create an ApplicationContext based on springconfig.xml

Get the bean with id="customerService" from the ApplicationContext

```
package module2.helloworld;

public class CustomerService {
 public void sayHello(){
   System.out.println("Hello from CustomerService");
 }
}
```

springconfig.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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <bean id="customerService" class="module2.helloworld.CustomerService" />
</beans>
```

Bean declaration

Spring Container

THE APPLICATION CONTEXT

The spring ApplicationContext

- Reads the Spring XML configuration file
- Instantiates objects declared in the Spring configuration file
- Wires objects together with dependency injection
- Creates proxy objects when needed

Creation of the ApplicationContext

```
ApplicationContext context = new ClassPathXmlApplicationContext("springconfig.xml");
```

Resource on the classpath

```
ApplicationContext context = new FileSystemXmlApplicationContext("C:\springconfig.xml");
```

Resource on the filesystem

Spring Container

SPRING BEANS

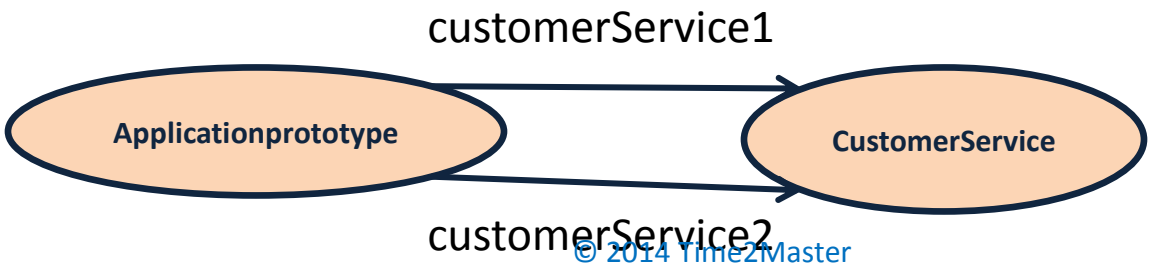
Spring beans are default singletons

```
public class Application{
  public static void main(String[] args) {
    ApplicationContext context = new
                        ClassPathXmlApplicationContext("module2/singleton/springconfig.xml");
    CustomerService customerService1 = context.getBean("customerService", CustomerService.class);
    CustomerService customerService2 = context.getBean("customerService", CustomerService.class);
    System.out.println("customerService1 ="+ customerService1);
    System.out.println("customerService2 ="+ customerService2);
  }
}
```

```
public class CustomerService {
  public CustomerService() {
  }
}
```

```
<bean id="customerService" class="module2.singleton.CustomerService" />
```

```
customerService1 =module2.singleton.CustomerService@29e357
customerService2 =module2.singleton.CustomerService@29e357
```



Prototype beans

```
public class Application{
  public static void main(String[] args) {
    ApplicationContext context =
                  new ClassPathXmlApplicationContext("module2/prototype/springconfig.xml");
    CustomerService customerService1 = context.getBean("customerService", CustomerService.class);
    CustomerService customerService2 = context.getBean("customerService", CustomerService.class);
    System.out.println("customerService1 ="+ customerService1);
    System.out.println("customerService2 ="+ customerService2);
  }
}
```

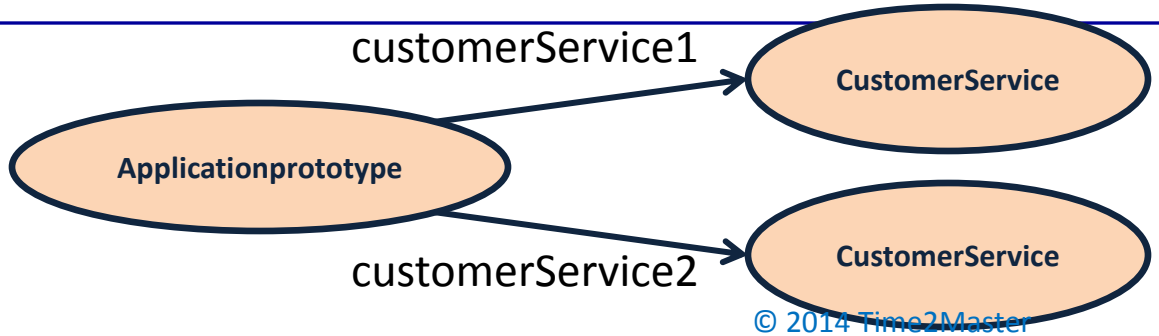
```
public class CustomerService {
    public CustomerService() {
    }
}
```

difference between lazy and prototype

```
<bean id="customerService" class="module2.prototype.CustomerService" scope="prototype" />
```

prototype

```
customerService1 =module2.prototype.CustomerService@1632847
customerService2 =module2.prototype.CustomerService@e95a56
```



Eager-instantiation of beans

```
public class Application {
  public static void main(String[] args) {
    System.out.println("1");
    ApplicationContext context = new
           ClassPathXmlApplicationContext("/module2/eagerinstantiation/springconfig.xml");
    System.out.println("2");
    CustomerService customerService = context.getBean("customerService", CustomerService.class);
    System.out.println("3");
    customerService.addCustomer("Frank Brown");
    System.out.println("4");
  }
}
```

```
public class CustomerServiceImpl implements CustomerService {
  public CustomerServiceImpl() {
    System.out.println("calling constructor of CustomerServiceImpl");
  }

  public void addCustomer(String customername) {
    System.out.println("calling addCustomer of CustomerServiceImpl");
  }
}
```

```
<bean id="customerService" class="module2.eagerinstantiation.CustomerServiceImpl" />
```

```
1
calling constructor of CustomerServiceImpl
2
3
calling addCustomer of CustomerServiceImpl
4
```

The CustomerService bean is eagerly instantiated

Lazy-instantiation of beans

```
public class Application {
  public static void main(String[] args) {
    System.out.println("1");
    ApplicationContext context = new
         ClassPathXmlApplicationContext("/module2/lazyinstantiation/springconfiglazy.xml");
    System.out.println("2");
    CustomerService customerService = context.getBean("customerService", CustomerService.class);
    System.out.println("3");
    customerService.addCustomer("Frank Brown");
    System.out.println("4");
  }
}
```

```
public class CustomerServiceImpl implements CustomerService {
  public CustomerServiceImpl() {
    System.out.println("calling constructor of CustomerServiceImpl");
  }

  public void addCustomer(String customername) {
    System.out.println("calling addCustomer of CustomerServiceImpl");
  }
}
```

```
<bean id="customerService" class="module2.lazyinstantiation.CustomerServiceImpl"
      lazy-init="true" />
```

Lazy instantiation

```
1
2
calling constructor of CustomerServiceImpl
3
calling addCustomer of CustomerServiceImpl
4
```

The CustomerService bean is lazy instantiated

Spring Beans

- Spring beans default to eagerly instantiated singletons, but can be configured to lazily instantiate or even not be a singleton at all.
- Spring beans are unaware of the fact that they are organized by Spring, regardless of what their configuration is.

Spring Container

LIFECYCLE METHODS

Lifecycle methods

```
public interface CustomerService {
  public void addCustomer(String customername);
  public void init();
  public void cleanup();
}
```

```
public class CustomerServiceImpl implements CustomerService {
  public CustomerServiceImpl() {
    System.out.println("calling constructor of CustomerServiceImpl");
  }
  public void addCustomer(String customername) {
    System.out.println("calling addCustomer of CustomerServiceImpl");
  }
  public void init() {
    System.out.println("calling init method of CustomerService");
  }
  public void cleanup() {
    System.out.println("calling cleanup method of CustomerService");
  }
}
```

```
<bean id="customerService" class="module2.xmllifecycle.CustomerServiceImpl"
      init-method="init" destroy-method="cleanup"/>
```

Method called just after the constructor

Method called when you close the ApplicationContext

Lifecycle methods example

```
public class Application {
  public static void main(String[] args) {
    System.out.println("1");
    ConfigurableApplicationContext  context = new
                    ClassPathXmlApplicationContext("/module2/xmllifecycle/springconfig.xml");
    System.out.println("2");
    CustomerService customerService = context.getBean("customerService", CustomerService.class);
    System.out.println("3");
    customerService.addCustomer("Frank Brown");
    System.out.println("4");
    context.close();
  }
}
```

ConfigurableApplicationContext

Close the ApplicationContext

```
1
calling constructor of CustomerServiceImpl
calling init method of CustomerService
2
3
calling addCustomer of CustomerServiceImpl
4
calling cleanup method of CustomerService
```

init method

cleanup method

Note: prototype beans are never destroyed

Lifecycle methods with annotations

```
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

public class CustomerServiceImpl implements CustomerService {
  public CustomerServiceImpl() {
    System.out.println("calling constructor of CustomerServiceImpl");
  }
  public void addCustomer(String customername) {
    System.out.println("calling addCustomer of CustomerServiceImpl");
  }
  @PostConstruct
  public void init() {
    System.out.println("calling init method of CustomerService");
  }
  @PreDestroy
  public void cleanup() {
    System.out.println("calling cleanup method of CustomerService");
  }
}
```

@PostConstruct

@PreDestroy

```
1
calling constructor of CustomerServiceImpl
calling init method of CustomerService
2
3
calling addCustomer of CustomerServiceImpl
4
calling cleanup method of CustomerService
```

init method

cleanup method

Lifecycle methods with annotations

```
<?xml version="1.0" encoding="UTF-8"?>
<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-3.0.xsd"
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>
   <bean id="customerService" class="module2.annotationslifecycle.CustomerServiceImpl"/>
</beans>
```

Tells Spring to check for annotation configuration in the Spring beans

Active Learning

- When a bean is declared with scope = prototype does it load eagerly or lazily?

always lazy because it is prototype, by default is eager

- Specifying a destroy method is not enough for Spring to use it, what else is needed?

call aplicationContext

Summary

- The Spring ApplicationContext instantiates all Spring beans declared in the Spring XML configuration file
- Spring beans are eagerly instantiated singletons by default
- Spring allows you to call your init methods and destroy methods anything you like.
  - Just tell spring what the name of the method is, and Spring takes care of the rest.

Main Point

- The Spring Container creates objects based on its configuration, default these are created only once at the start of the application
- *Science of Consciousness*: Spring's aim is to do less and accomplish more. By creating the objects only once at the start it never has to slow down execution time