WebSystique

learn together

Spring Dependency Injection Annotation Example, Beans Auto-wiring using @Autowired, @Qualifier & @Resource Annotations Configuration

August 25, 2014 websystiqueadmin

Spring Dependency Injection Annotation example + Spring Auto-wiring Annotation example.
This article explains Spring Dependency Injection and Beans auto-wiring using Spring `@Autowired` annotation. @Autowired can be applied on a bean's constructor, field, setter method or a config method to autowire the dependency using Spring's dependency injection.

@Autowired wires the dependency using `bean datatype`. If you are looking for bean wiring using bean name (somewhat analogues to byName in XML), you can use standard `@Resource` annotation with 'name' attribute. `@Qualifier` annotation is often used in conjunction with @Autowired to resolve ambiguity in case more that one bean of injected type exist in application context.

Other interesting posts you may like

- Spring MVC 4+AngularJS Example

Recent Posts

- Spring MVC 4+Hibernate 4 Many-to-many JSP Example
- Spring MVC 4+Hibernate 4+MySQL+Maven integration example using annotations
- Spring MVC4 FileUpload-Download Hibernate+MySQL Example
- TestNG Mockito Integration Example Stubbing Void Methods
- Maven surefire plugin and TestNG Example
- Spring MVC 4 Form Validation and Resource Handling

Let's explore practical usage of each of them in detail via examples:

1. @Resource Example (autowiring by name)

Define Beans to work with

```
package com.websystique.spring.domain;

import javax.annotation.Resource;

import org.springframework.stereotype.Component;

@Component("application")
public class Application {

    @Resource(name="applicationUser")
    private ApplicationUser user;

    @Override
    public String toString() {
        return "Application [user=" + user + "]";
    }
}
```

```
package com.websystique.spring.domain;

import org.springframework.stereotype.Component;

@Component("applicationUser")
public class ApplicationUser {

    private String name = "defaultName";

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    @Override
    public String toString() {
        return "ApplicationUser [name=" + name + "]";
    }
}
```

Standard `@Resource` annotation marks a resource that is needed by the application. It is analogous to @Autowired in that both injects beans by type when no attribute provided. But with name attribute, @Resource allows you to inject a bean by it's name, which @Autowired does not.

In above code, Application's user property is annotated with @Resource(name="applicationUser"). In this case, a bean with name 'applicationUser' found in applicationContext will be injected here.

Spring Configuration Class

```
package com.websystique.spring.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan("com.websystique.spring")
public class AppConfig {

}
```

Notice `@ComponentScan` which will make Spring auto detect the annotated beans via scanning the specified package and wire them wherever needed (using @Resource or @Autowired).

Above configuration is analogues to following in 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:cor
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/context http://www.spring

    <context:component-scan base-package="com.websystique.spring" /

</beans>
```

Run Application

Load the context and run it.

```
package com.websystique.spring;

import org.springframework.context.annotation.AnnotationConfigAppli
```

```
import org.springframework.context.support.AbstractApplicationConte

import com.websystique.spring.config.AppConfig;
import com.websystique.spring.domain.Application;

public class AppMain {

    public static void main(String args[]){
        AbstractApplicationContext  context = new AnnotationConfigA

        //Byname Autowiring
        Application application = (Application)context.getBean("app
        System.out.println("Application Details : "+application);

    }

}
```

Following will be the output.

```
Application Details : Application [user=ApplicationUser [name=defau
```

2. @Autowired Example

Define Beans to work with

```
package com.websystique.spring.domain;

import org.springframework.stereotype.Component;

@Component
public class License {

    private String number="123456ABC";

    @Override
    public String toString() {
        return "License [number=" + number + "]";
    }
    //setters, getters
}
```

@Autowired on Setter method

```
package com.websystique.spring.domain;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("driver")
public class Driver {

    private License license;
```

```
    @Autowired
    public void setLicense(License license) {
        this.license = license;
    }

    @Override
    public String toString() {
        return "Driver [license=" + license + "]";
    }
    //getter
}
```

@Autowired on Field

```
package com.websystique.spring.domain;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("driver")
public class Driver {
    @Autowired
    private License license;

    //getter,setter

    @Override
    public String toString() {
        return "Driver [license=" + license + "]";
    }
}
```

@Autowired on Constructor

```
package com.websystique.spring.domain;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("driver")
public class Driver {

    private License license;

    @Autowired
    public Driver(License license){
        this.license = license;
    }

    @Override
    public String toString() {
        return "Driver [license=" + license + "]";
    }
}
```

Run Application

Load the context and Run it.

```
package com.websystique.spring;
```

```
import org.springframework.context.annotation.AnnotationConfigAppli
import org.springframework.context.support.AbstractApplicationConte

import com.websystique.spring.config.AppConfig;
import com.websystique.spring.domain.Driver;

public class AppMain {

    public static void main(String args[]) {
        AbstractApplicationContext context = new AnnotationConfigAp
                AppConfig.class);

        Driver driver = (Driver) context.getBean("driver");
        System.out.println("Driver Details : " + driver);
    }

}
```

Following will be the output

```
Driver Details : Driver [license=License [number=123456ABC]]
```

3. @Qualifier Example

`@Qualifier` is useful for the situation where you have more than one bean matching the type of dependency and thus resulting in ambiguity.

Define Beans to work with

```
package com.websystique.spring.domain;

public interface Car {

    public void getCarName();

}
```

```
package com.websystique.spring.domain;

import org.springframework.stereotype.Component;

@Component("Ferari")
public class Ferari implements Car{

    public void getCarName() {
        System.out.println("This is Ferari");
    }

}
```

```
package com.websystique.spring.domain;

import org.springframework.stereotype.Component;

@Component("Mustang")
public class Mustang implements Car{

    public void getCarName() {
        System.out.println("This is Mustang");
    }

}
```

```
package com.websystique.spring.domain;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component
public class Bond {

    @Autowired
    private Car car;

    public void showCar(){
        car.getCarName();
    }
}
```

Run Applicaion

Load context and Run it.

```
package com.websystique.spring;

import org.springframework.context.annotation.AnnotationConfigAppli
import org.springframework.context.support.AbstractApplicationConte

import com.websystique.spring.config.AppConfig;
import com.websystique.spring.domain.Bond;

public class AppMain {

    public static void main(String args[]) {
        AbstractApplicationContext context = new AnnotationConfigAp
                AppConfig.class);

        Bond bond = (Bond) context.getBean("bond");
        bond.showCar();
    }

}
```

On running, Spring throws following exception:

```
Caused by: org.springframework.beans.factory.NoUniqueBeanDefinitior
    at org.springframework.beans.factory.support.DefaultListableBea
    at org.springframework.beans.factory.support.DefaultListableBea
```

```
    at org.springframework.beans.factory.annotation.AutowiredAnnota
    ... 14 more
```

What happened is Spring was not able to decide which bean (Ferari or Mustang as both implements Car) to choose for auto-wiring ,it throws this exception.

Happily, @Qualifier saves the day.

Change the Bond class as shown below

```
package com.websystique.spring.domain;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component
public class Bond {

    @Autowired
    @Qualifier("Mustang")
    private Car car;

    public void showCar(){
        car.getCarName();
    }
}
```

Run Application

Following will be the output.

```
This is Mustang
```

Mark Autowiring optional with attribute required="false"

By default, @Autowored annotation makes sure that field is indeed autowired. In case autowiring is not successful, Spring will throw an exception. There are times however when you want to make autowiring optional. Setting @Autowired required attribute to 'false' will make this filed optional for autowiring and Spring will skip it(remain null) if dependency not found.

```
package com.websystique.spring.domain;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("driver")
```

```
public class Driver {
    @Autowired(required=false)
    private License license;

    //getter,setter

    @Override
    public String toString() {
        return "Driver [license=" + license + "]";
    }
}
```

In above example, if no bean of type License been found, it will remain null and no error will be thrown on context loading.

caveat :
Note that standard @Resource annotation does not have this flexibility. In case the dependency annotated with @Resource not found, Spring will throw an exception. Both @Resource and @Autowired have few differences : No optionality in @Resource and no autowiring by bean name in @Autowired.
All in all, @Autowired is the most widely used option compare to @Resource and autowire attribute in XML.

That's it.

Download Source Code

Download Now!

References

- Spring framework

websystiqueadmin

If you like tutorials on this site, why not take a step further and connect me on Facebook , Google Plus & Twitter as well? I would love to hear your thoughts on these articles, it will help me improve further our learning process.

If you appreciate the effort I have put in this learning site, help me improve the visibility of this site towards global audience by sharing and linking this site from within and beyond your network. You & your friends can always link my site from your site on www.websystique.com, and share the learning.

After all, we are here to learn together, aren't we?

Related Posts:

1. **Spring Beans Auto-wiring Example using XML Configuration**
2. **Spring Dependency Injection Example with Constructor and Property Setter (XML)**
3. **Spring Auto-detection autowire & Component-scanning Example With Annotations**
4. **Spring @PropertySource & @Value annotations example**

spring. permalink.

← Spring Beans Auto-wiring Example using XML Configuration

Spring Job Scheduling using TaskScheduler (XML Config) →