WebSystique
learn together

Spring @PropertySource & @Value annotations example

August 19, 2014 websystiqueadmin

In this post we will see how to read values from properties files using Spring `@PropertySource` & `@Value` annotations. We will also discuss about Spring `Environment` interface. We will see corresponding XML configuration as well for side-by-side comparison.

Spring @PropertySource annotations is mainly used to read from properties file using Spring's Environment interface. This annotation is in practice, placed on @Configuration classes. Spring @Value annotation can be used to specify expression on field or methods. Common use case is to specify the property from a .properties file along with default value. Let's see full example below.

Other interesting posts you may like

- Spring MVC 4+AngularJS Example
- 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

WebSyst
G+ Foll
+ 127

Recent Posts

Spring 4 MVC+AngularJS CRUD Application using ngResource

Angularjs Server Communication using ngResource-CRUD Application

AngularJS Custom-Directives controllers, require option guide

AngularJS Custom-Directives transclude, ngTransclude guide

AngularJS Custom-Directives replace option guide

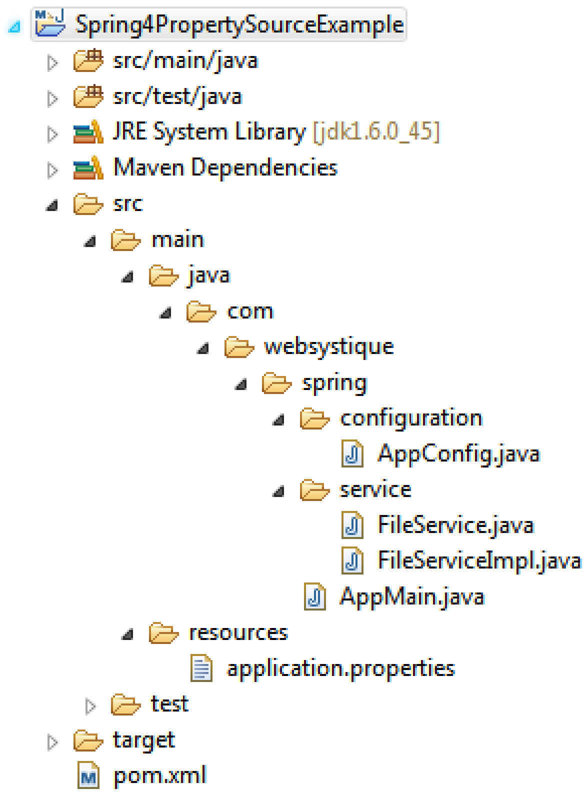
- Maven surefire plugin and TestNG Example
- Spring MVC 4 Form Validation and Resource Handling

Following technologies being used:

- Spring 4.0.6.RELEASE
- Maven 3
- JDK 1.6
- Eclipse JUNO Service Release 2

Project directory structure

Following will be the final project directory structure for this example:



```
Spring4PropertySourceExample
    src/main/java
    src/test/java
    JRE System Library [jdk1.6.0_45]
    Maven Dependencies
    src
        main
            java
                com
                    websystique
                        spring
                            configuration
                                AppConfig.java
                            service
                                FileService.java
                                FileServiceImpl.java
                            AppMain.java
            resources
                application.properties
        test
    target
    pom.xml
```

Let's add the content mentioned in above directory structure.

Step 1: Provide Spring dependencies in Maven pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http:
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://ma
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.websystique.spring</groupId>
    <artifactId>Spring4PropertySourceExample</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <name>Spring4PropertySourceExample</name>

    <properties>
        <springframework.version>4.0.6.RELEASE</springframework.ver
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${springframework.version}</version>
        </dependency>
    </dependencies>

</project>
```

Step 2: Create Spring Configuration Class

Spring configuration class are the ones annotated with `@Configuration`. These classes contains methods annotated with `@Bean`. These @Bean annotated methods generates beans managed by Spring container.

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

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.context.support.PropertySourcesPlaceholc

@Configuration
@ComponentScan(basePackages = "com.websystique.spring")
@PropertySource(value = { "classpath:application.properties" })
public class AppConfig {

    /*
     * PropertySourcesPlaceHolderConfigurer Bean only required for
     * Remove this bean if you are not using @Value annotations for
     */
    @Bean
    public static PropertySourcesPlaceholderConfigurer propertySour
        return new PropertySourcesPlaceholderConfigurer();
    }
}
```

@PropertySource(value = { “classpath:application.properties” }) annotation makes the properties available from named property file[s] (referred by value attribute) to Spring `Environment`. Environment interface provides getter methods to read the individual property in application.

Notice the `PropertySourcesPlaceholderConfigurer` bean method.

This bean is required only for resolving ${…} placeholders in @Value annotations. In case you don't use ${…} placeholders, you can remove this bean altogether.

Above Configuration can be expressed in XML based approach as follows (let's name it app-config.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/contex
        xsi:schemaLocation="http://www.springframework.org/schema/b
    http://www.springframework.org/schema/context   http://www.spri

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

    <bean class="org.springframework.context.support.PropertySource
        <property name="ignoreUnresolvablePlaceholders" value="true
        <property name="locations">
          <list>
            <value>classpath:application.properties</value>
          </list>
        </property>
      </bean>
</beans>
```

Step 3: Create Sample properties file

```
jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://localhost:3306/websystique
jdbc.username = myuser
jdbc.password = mypassword
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = false
hibernate.format_sql = false
sourceLocation = /dev/input
```

We will read the properties from this file using above mentioned configuration in our sample service class.

Step 4: Create Sample service class

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

public interface FileService {

    void readValues();
}
```

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

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.env.Environment;
import org.springframework.stereotype.Service;

@Service("fileService")
public class FileServiceImpl implements FileService {

    @Value("${sourceLocation:c:/temp/input}")
    private String source;

    @Value("${destinationLocation:c:/temp/output}")
    private String destination;

    @Autowired
    private Environment environment;

    public void readValues() {
        System.out.println("Getting property via Spring Environment
                + environment.getProperty("jdbc.driverClassName"));

        System.out.println("Source Location : " + source);
        System.out.println("Destination Location : " + destination)

    }

}
```

First point to notice is Environment got auto-wired by Spring. Thanks to @PropertySoruce annotation , this Environment will get access to all the properties declared in specified .properties file. You can get the value of specif property using getProperty method. Several methods are defined in Environment interface.

Other interesting point here is `@Value` annotation. Format of value annotation is

```
@value("${key:default")
private String var;
```

Above declaration instruct Spring to find a property with key named 'key' (from .properties file e.g.) and assign it's value to variable var.In case property 'key' not found, assign value 'default' to variable var.

Note that above ${...} placeholder will only be resolved when we have registered `PropertySourcesPlaceholderConfigurer` bean (which we have already done above) else the @Value annotation will always assign default values to variable var.

Step 5: Create Main to run as Java Application

```
package com.websystique.spring;

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

import com.websystique.spring.configuration.AppConfig;
import com.websystique.spring.service.FileService;

public class AppMain {

    public static void main(String args[]){
        AbstractApplicationContext  context = new AnnotationConfigA
        FileService service = (FileService) context.getBean("fileSe

        service.readValues();
        context.close();
    }

}
```

Run above program , you will see following output:

```
Getting property via Spring Environment :com.mysql.jdbc.Driver
Source Location : /dev/input
Destination Location : c:/temp/output
```

Since destinationLocation property was not found in application.properties, it's got the default value.

That's it.

For **XML based configuration** , replace

```
AbstractApplicationContext  context = new AnnotationConfigApplicati
```

with

```
AbstractApplicationContext context = new ClassPathXmlApplicationCor
```

in above main, no other changes. Run the program and you will see same output.

Download Source Code

Download Now!

References

- Spring framework
- @Value
- @PropertySource
- Spring Environment

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 4 + Hibernate 4 + MySQL+ Maven Integration example (Annotations+XML)**
2. **Spring @Profile Example**
3. **Spring Dependency Injection Annotation Example, Beans Auto-wiring using @Autowired, @Qualifier & @Resource Annotations Configuration**
4. **Spring Auto-detection autowire & Component-scanning Example With Annotations**

spring. permalink.

← Spring Auto-detection autowire & Component-scanning Example With Annotations

Spring @Profile Example →

5 Comments websystique 1 Login

Recommend Share Sort by Best

Join the discussion…

Jesus Moscoso • 4 months ago

Thanks for the post. It's clear and i could follow pretty well but after trying it is not working to me.

I created a post in Stack overflow http://stackoverflow.com/quest... where i reference to this one.

Any ideas what could be wrong in my project.

• Reply • Share ›

websystique Mod Jesus Moscoso • 4 months ago

Hi,
Could you please mention the exact version of spring you are using?

• Reply • Share ›

Jesus Moscoso websystique • 4 months ago

Sure.

```
<spring.version>3.2.6.RELEASE</spring.version>

<dependency>
<groupid>org.springframework</groupid>
<artifactid>spring-web</artifactid>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupid>org.springframework</groupid>
<artifactid>spring-webmvc</artifactid>
<version>${spring.version}</version>
</dependency>
<dependency>
<groupid>org.springframework</groupid>
<artifactid>spring-context</artifactid>
<version>${spring.version}</version>
</dependency>
```

• Reply • Share ›

websystique Mod Jesus Moscoso • 4 months ago

Hi Jesus,

I tested this example with 3.2.6.RELEASE without any issue. Anyway, Is there a way you could share your minimal runnable code you are trying to run (through github for example)? That would help me to pinpoint particular issue you are getting. You can also use 'contact us' page of this site to send the details if you prefer.

• Reply • Share ›

Jesus Moscoso websystique
• 4 months ago

Yes, you are right. I found there is something else.

I can execute this in @Controller classes and in @Services classes, but i also have a @Service class that implements UserDetailsService class.

Using this solution provided by spring, i'm overriding the method:

```
@Override
public UserDetails
loadUserByUsername(String username)
throws UsernameNotFoundException {
```

and here is where I can not use property source values. I still need to investigate this further to see if there is something wrong in my side.

Thanks for all your assistance.

• Reply • Share ›

ALSO ON **WEBSYSTIQUE** WHAT'S THIS?

Copyright © 2014-2016 WebSystique.com. All rights reserved.