is SFig language syntax efficient and clear (and better than Spring-Framework's XML DSL)?

Asked 16 years ago Modified 6 years, 7 months ago Viewed 600 times

0

ADDENDUM EDIT:

> Have not accepted an answer to this as there has not been any feedback from experienced Spring Framework developers.

I've been working on a replacement DSL to use for Spring-Framework applicationContext.xml files (where bean initialization and dependency relationships are described for loading up into the Spring bean factory).

My motivation is that I just flat out don't like Spring's use of XML for this purpose nor do I really like any of the alternatives that have been devised so far. For various reasons that I won't go into, I want to stay with a declarative language and not some imperative scripting language such as Groovy.

So I grabbed the ANTLR parser tool and have been devising a new bean factory DSL that I've dubbed SFig. Here's a link that talks more about that:

SFig™ - alternative metadata config language for Spring-Framework

And here is the source code repository site:

http://code.google.com/p/sfig/

I'm interested to know how I'm doing on the language syntax so far. Do you think SFig is both efficient and clear to understand? (I'm particularly concerned right now with the mulit-line text string):

```
properties_include "classpath:application.properties";


org.apache.commons.dbcp.BasicDataSource dataSource {
    @scope = singleton;
    @destroy-method = close;
    driverClassName = "${jdbc.driverClassName}";
    url = "${jdbc.url}";
    username = "${jdbc.username}";
    password = "${jdbc.password}";
    defaultAutoCommit = true;
}
```

```
org.springframework.orm.ibatis.SqlMapClientFactoryBean sqlMapClient {
    @scope = singleton;
    @init-method = afterPropertiesSet;
    @factory-method = getObject;
    configLocation = "classpath:sqlmap-config.xml";
    dataSource = $dataSource;
}


/* this string will have Java unescape encoding applied */
STRING str = "\tA test\u0020string with \\ escaped character encodings\r\n";


/* this string will remain literal - with escape characters remaining in place
*/
STRING regexp = @"(\$\{([a-zA-Z][a-zA-Z0-9._]*)\})";


/* multi-line text block - equates to a java.lang.String instance */
TEXT my_multi_line_text = ///
Here is a line of text.
This is yet another. Here is a blank line:

Now picks up again.
///;


/* forward use of 'props' bean */
java.util.HashMap map {
    this( $props );
}


/* equates to a java.util.Propertis instance */
PROPERTIES props {
    "James Ward" = "Adobe Flex evangelist";
    "Stu Stern" = "Gorilla Logic - Flex Monkey test automation";
    Dilbert = "character in popular comic strip of same title";
    "App Title Display" = "Application: ${app.name}";
    "${app.desc}" = "JFig processes text-format Java configuration data";
}


/* equates to a java.util.ArrayList instance */
LIST list {
    this( ["dusty", "moldy", "${app.version}", $str] );
    [234, 9798.76, -98, .05, "numbers", $props, ["red", "green", "blue"]];
}
```

inversion-of-control antlr spring dsl parsing

Share

Improve this question

Follow

edited Feb 2, 2009 at 2:03

asked Dec 21, 2008 at 20:11

RogerV

3,856 ● 4 ● 30 ● 32

I had to edit my post to change 'JFig' to 'SFig'. Turns out there's already another Java-centric programming tool called JFig and it's been around for a number of years. I chose SFig to imply the intended relationship of configuration for Spring-Framework. – RogerV Jan 1, 2009 at 20:52

2 Answers

Sorted by: Highest score (default)

0

I'll provide a bit of background on Spring and it's applicationContext.xml file - that will lend clarity to some of the things going on in SFig syntax.

The applicationContext.xml file is used to express bean initialization for beans that will be managed by the Spring bean factory. So given the example of beans seen in my SFig version of this file, in Java application code one might request the bean factory to make an instance of a bean like so:

```
SqlMapClient sqlMapClient = getBean("sqlMapClient");
```

The bean factory takes care of any instantiation and initialization that the bean requires - even to the point of injecting dependencies. In this case, a SqlMapClient bean needs an instance of a *dataSource* bean (which is also described and referenced in the SFig example).

A bean descriptor relays the following information to the bean factory:

- the bean's Java class name
- a bean ID by which to request or reference it
- bean definition meta attributes (optional)
- constructor initialization arguments (optional)
- and/or property initializers

The '@' prefixes bean definition meta attributes. These are attributes that are used by the bean factory to manage the bean. For instance, @scope = singleton, would inform the bean factory to make a single instance of the bean, cache it, and hand out references to it when it is requested. The ones that can be set are the same ones defined by the Spring-Framework.

If a bean is to be initialized via a constructor, then that is expressed in SFig by a syntax that appears to be invoking *this* with arguments in parenthesis.

Or a bean can be initialized by setting its properties. Identifiers that are assigned to and not prefixed by '@' are bean properties.

When referencing a bean that is a required dependency, then it can be referred to by prefixing it's bean ID with '$'. Several examples of this appear in the SFig example.

The ${foo.bar} syle of variable appearing in string literals will be replaced by a Java property value. In this case, properties are loaded from the file application.properties via this line:

```
properties_include "classpath:application.properties";
```

Java System properties will be looked to next if not found in any included properties. This is a widely followed practices in many Java frameworks. The current XML-based applicationContext.xml file has a way of permitting this usage too.

Because java.util.Properties are often used to initialize beans, SFig provides the PROPERTIES as a special convenient syntax for declaring a Properties object. Likewise for java.util.List, which has the corresponding SFig LIST. Also, arrays of values can be declared within square brackets [...].

Additionally there is TEXT for declaring blocks of multi-line text. The '@' prefixing a string literal means to turn off escape encoding - a language syntax borrowed from C#.

One of the primary design objectives of the SFig DSL is to remain declarative in nature. I purposely am refraining from adding any imperative scripting features. The complexity of programming logic embedded in a text configuration file will imply possibility of having to debug it. Don't want to open yet another dimension of code debugging.

Share

Improve this answer

Follow

edited Jan 1, 2009 at 20:48

answered Dec 22, 2008 at 4:18

RogerV

3,856 ● 4 ● 30 ● 32

0

I haven't much experience with the Spring XML you refer, so you should take the following feedback with a pinch of salt.

As a second and third caveat:

- providing a snippet of code will give a *flavour* of what the language and its semantics are. It is difficult to completely understand some of the choices you have already made (and with good reason), so any feedback here may be completely contradictory or impossible in the light of those choices.
- language design is as much an art as a science, and so at this stage, any feedback you may get is likely to be quite subjective.

A larger, meta-, question: as a DSL are you trying to do configuration of Spring, or as a more general class of frameworks?

There: caveat emptor. Now my subjective and incomplete feedback ;)

- I'm not sure I understand the reason why you have the `@` **prefix** for `scope` and `destroy-method`, but not `driverClassName`. Also the mix of **both xml-case and camelCase** isn't completely apparent to start with. Is the `@` prefix a type modifier, or are these keywords in the language?
- I'm not completely sure of your intentions about the block header format. You have class name, then a function of that class; is the intention to specify what class your are going to use for a particular function?

e.g.

```
sqlMapClient: org.springframework.orm.ibatis.SqlMapClientFactoryBean {
    # body.
}
```

or even:

```
sqlMapClient {
    @class = org.springframework.orm.ibatis.SqlMapClientFactoryBean;
    # is there a sensible (perhaps built-in) default if this is missing?
}
```

- I like the **variable substitution**; I presume the values will come from System properties?
- I like being able to specify **string literals** (without escaping), especially for the regular expressions you've shown. However, having multi-character quote or quote modifier seems a little alien. I guess you considered the single-quote (shell and Perl use single-quotes for literal strings).
- On the other hand, I think the triple forward slash for **multi-line** `TEXT` is the right approach, but two reminiscent of comments in C-style languages. Python uses a triple `"` for this purpose. Some shell idioms have a multi-line text convention I would not copy.
- I very much like the look of properties and config location, using what looks like a **URI notion of addressing**. If this is a URI, `classpath://file.xml` may be clearer. I may have the wrong end of the stick here, however.
- I also very much like the notion of list and map literals you have, though I'm not sure where:
    - `this` comes into it (I guess a call to a Java constructor)

- why some types are capitalized, and others are not. Do I take it that there is a default `MAP` type, which you can be more specific type if you wish to?
- is Dilbert an unquoted string literal?

Finally, I'd point you to another configuration DSL, though perhaps more for sysadmin usage: Puppet.

Go well.

Share

Improve this answer

Follow

edited May 6, 2018 at 18:02

Dez

5,838 ● 8 ● 45 ● 52

answered Dec 22, 2008 at 0:55

jamesh

20.1k ● 14 ● 58 ● 96