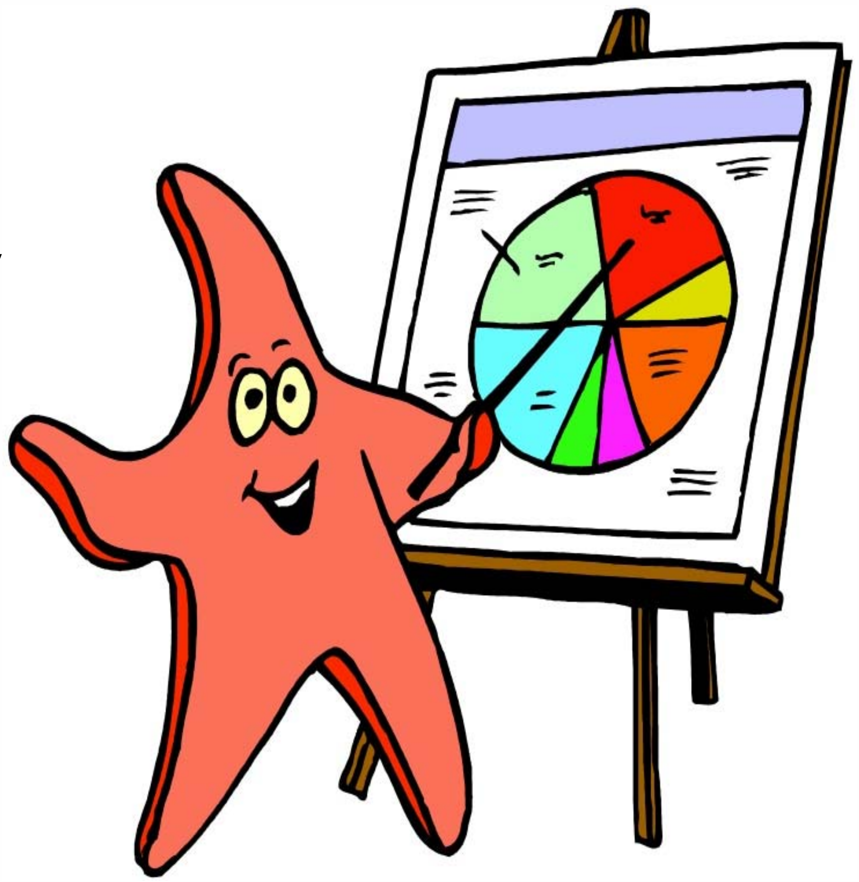
Spring Namespaces

Sydney Spring User Group
7 August 2006

Last update: 15-Oct-06

About Me

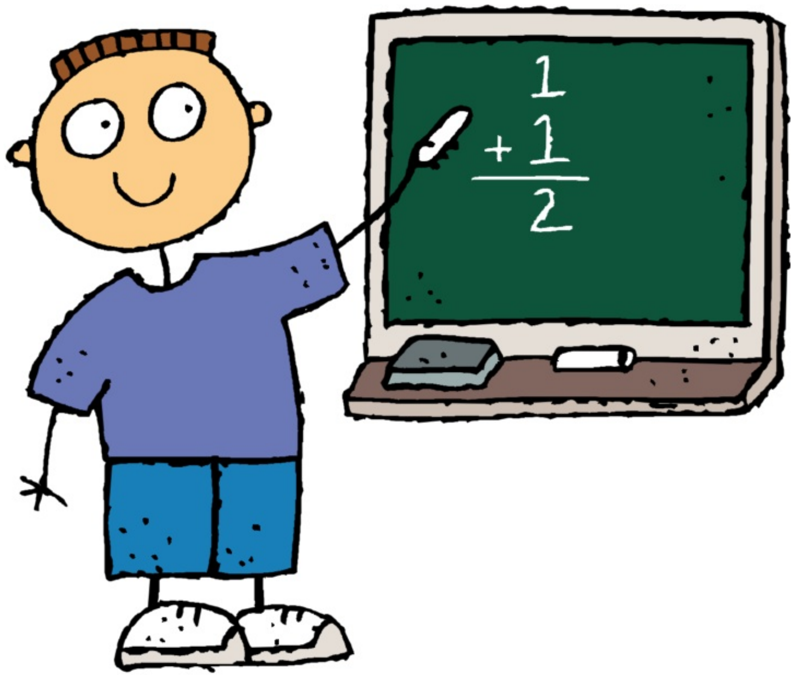
- Ben Alex
- Director - Interface21
- OSS Projects
  - Lead - Acegi Security
  - Dev – Spring RCP
- Author
  - Professional Java Development w/ Spring Framework
- Using Spring since 03
- Teach Spring since 04



Agenda

- Two quick demos
- Getting our hands dirty!

Thanks to Erik Wiersma (JTeam) for original presentation material



Two quick demos

Sample 1 - Webframework

- NAWF (Not Another Web Framework)
- Strikingly named “Webframework”
- Using IoC as that’s A Good Thing™
  - Thus, it’s very configurable
  - But... generally a lot of XML configuration

Let's take a look at the XML

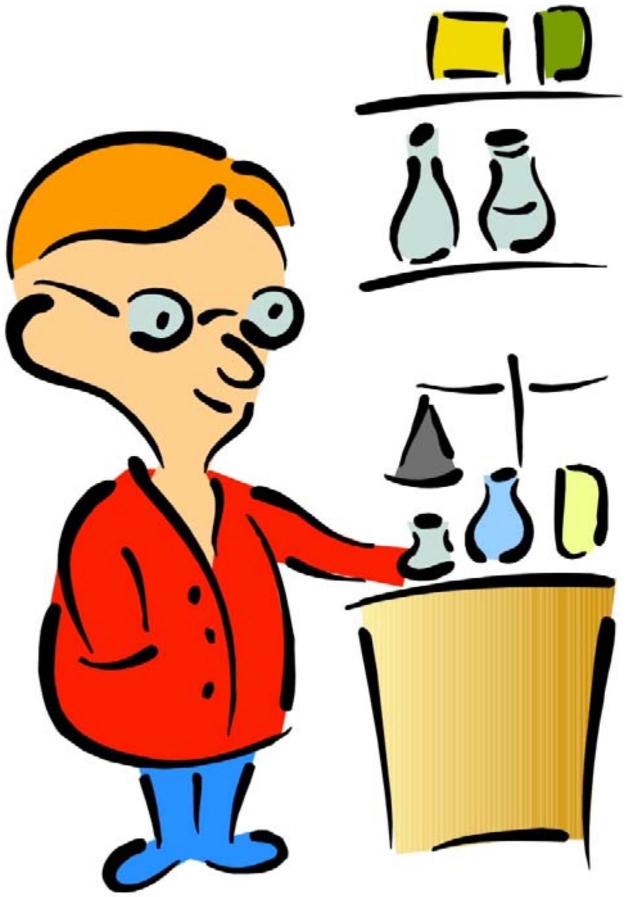
Sample 1 - Summarized

- Removed lots of XML
- Less error prone (XML validation)
- Easier to write (XML code completion)
- Encapsulated Java and Spring constructs
  - At last, XML and OO lifecycles independent
  - Great refactoring benefits

Sample 2 - DWR

- Direct Web Remoting (DWR):
  - Javascript to Java remoting framework
  - DWR is “Easy Ajax for Java”
- On a side track…
  - Don’t forget about JSON-RPC
  - Also check out Google Web Toolkit

Spring
from the source

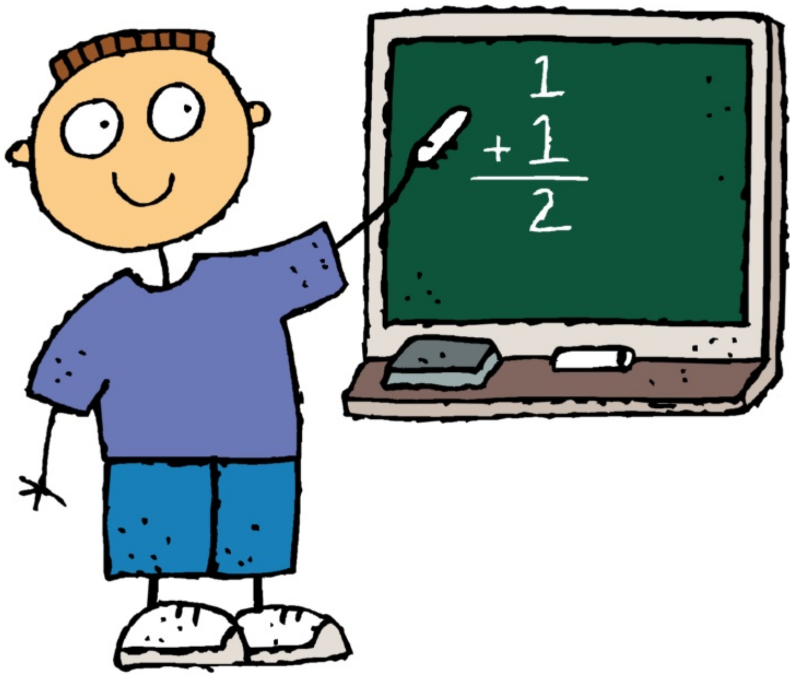


Let's take a look at the XML

INTERFACE21

Sample 2 - Summarized

- Same benefits as first example, plus…
- Easy to see which beans are remoted
- Notice we can work with both
  - New elements (BeanDefinitionParser)
  - Existing beans (BeanDefinitionDecorator)



Getting our hands dirty

How Do You Do It?

1. Write your preferred XML
2. Select an XML namespace string
3. Create supporting XML Schema
4. Write a `NamespaceHandler`
5. Create META-INF/spring.handlers
6. Create META-INF/spring.schemas

Write Your Preferred XML

Before:

```
<bean id="wordCounter“
   class="jteam.springone.WordCounterImpl">
  <property name="delimiter" value=","/>
  <property name="encoding" value="UTF8"/>
  <property name="maxWordCount” value="10"/>
</bean>
```

Write Your Preferred XML

After:

```
<wordCounter
     delimiter="comma"
     encoding="UTF-8"
     maxWordCount="100”/>
```

XML Namespace Background

- XML documents may have elements with the same name:

```
<table>
   <tr>
      <td>Pizza</td>
      <td>Sushi</td>
   </tr>
</table>
```

```
<table>
   <name>Office Table</name>
   <color>Cherry Wood</color>
</table>
```

HTML

Furniture

XML Namespace Background

- XML documents may have elements with the same name:

```
<table xmlns=“html”>
   <tr>
      <td>Pizza</td>
      <td>Sushi</td>
   </tr>
</table>
```

```
<table xmlns=“furniture”>
   <name>Office Table</name>
   <color>Cherry Wood</color>
</table>
```

HTML

Furniture

XML Namespace Background

- XML documents may have elements with the same name:

```
<h:table xmlns:h="html">
   <h:tr>
      <h:td>Pizza</h:td>
      <h:td>Sushi</h:td>
   </h:tr>
</h:table>
```

```
<f:table xmlns:f="furniture">
   <f:name>Office Table</f:name>
   <f:color>Cherry Wood</f:color>
</f:table>
```

HTML

Furniture

Select an XML Namespace String

- Need to pick a new namespace for our WordCounter
- Let’s call it:

http://www.jteam.nl/stringutils

- This is only a namespace, and does not specify the location of the *.xsd file

Create Supporting XML Schema

- Bound to your selected namespace
- Stored in an *.xsd file
- Defines and validates XML structure
  - Elements
  - Attributes
  - more...
- IDE/Tool support
- Powerful but complex

BeanDefinitions Matter

- Application contexts configurable via:
  - XML
  - Properties files
  - Several scripting languages
  - Java
- `BeanDefinitions` are abstraction of bean config
- Our `NamespaceHandler` will work with these `BeanDefinitions`

Write a NamespaceHandler

- Implement `NamespaceHandler` interface
- Or, simply extend `NamespaceHandlerSupport`

Using NamespaceHandlerSupport

```
public void init() {
   registerBeanDefinitionParser(
        “wordCounter",      Element name
        new WordCounterParser()
   );
}
          BeanDefinitionParser
```

BeanDefinitionParser Contract

```
BeanDefinition parse(
   Element element,
   ParserContext parserContext
);
```

Create META-INF/spring.handlers

Your namespace string

```
http\://www.jteam.nl/stringutils=\
jteam.springone.sample.StringUtilsNamespaceHandler
```

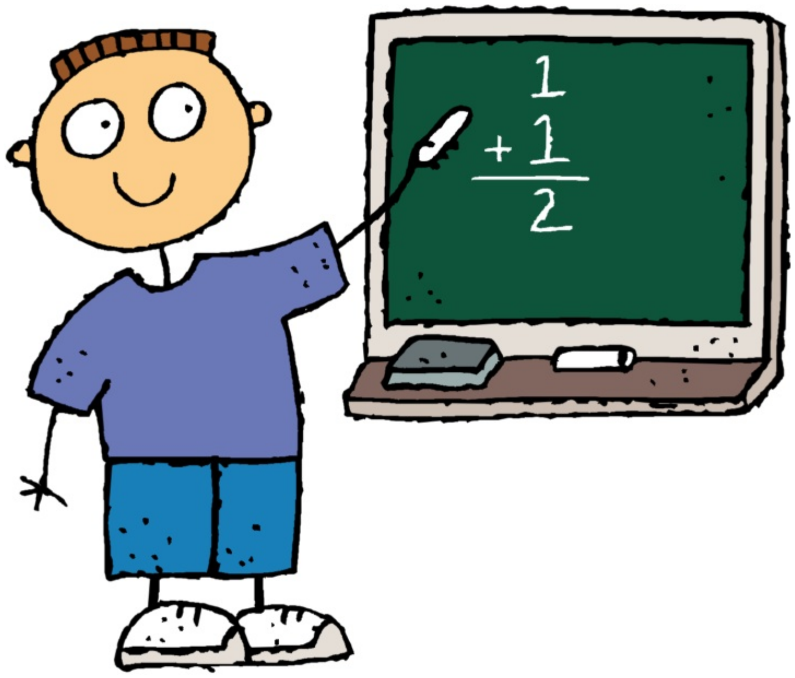
NamespaceHandler
implementation

Create META-INF/spring.schemas

Schema location in XML

```
http\://www.jteam.nl/sample/stringutils.xsd=\
  jteam/springone/sample/stringutils.xsd
```

Schema file location in classpath



WordCounter code review

Uncharted territory

- No best practices
- Few examples
- Limitations unknown

Example Usages

- Acegi Security ☺
- Direct Web Remoting (DWR)
- Spring 2.0
  - AOP
  - Jndi
  - Util
  - Tx
- Spring Modules Validation

Summary

- Domain driven configuration is better readable and easier to understand:
  - Configurations related to your domain
  - Sensible defaults (multiple approaches)
  - Encapsulation (via `NamespaceHandler`)
  - Stronger typing (via XSD)
  - “Compile time” checks (via XSD)
  - Documentation (`<xsd:documentation>`)

Questions?

Thank You!