upstate.edu

Lesson 2: Java and Java Objects

19-24 minutes

Learning Objectives

- Understand:
- The nature of objects in the Velocity environment
- Why the Java API documentation is relevant to Velocity
- Learn how to:
- Expose information of objects
- Read the Java API documentation
- Use Java objects

Collapse all

Block and Format Contents in Regions

- Check the page named `hello-world`, and make sure that a format and an XML block, both named `hello-world`, are attached to the same region of the page
- Remove all the contents from the format; put in ## instead
- Now check the page again and it should be blank, even though there is an XML block attached to the same region
- If we detach the format from the page region, then the XML contents of the block will show up on the page
- This means that if a page region is attached with both a block and a format, then the format determines what is displayed in the region, irrespective of the block content

The Relevance of Java

- Velocity is Java-based
- In the Velocity environment, almost everything are Java objects
- A variable, when assigned a valid value, is normally associated with a Java object
- Every Java object has type information (e.g., the name of the class associated with the object)
- For example, when we assign the String "Hello, World!" to a variable `$greetings`, the type of the variable is `java.lang.String`
- We can invoke object methods or static class methods through a Java object to manipulate the associated data:
  ```
  #set( $greetings = "Hello, World!" )
  $greetings.charAt( 0 ) ##=> H
  ```
- We can retrieve a property of a Java object:
  ```
  $greetings.Class ##=> class java.lang.String
  ```
- This piece of information is particularly important if we want to find out what methods are defined in a Java class
- We use Java syntax to invoke methods or retrieve properties (using the dot operator)
- Special points about Java method calls:
- An object of the relevant class is required even when a static method of that class is invoked, unless reflection is used
- Commas are required to separate argument values passed into method calls
  ```
  ## create a LinkedHashMap object
  #set( $states = {} )
  ## add an entry by invoking the put method
  #set( $void = $states.put( 'NY', 'New York' ) )
  ```
- A parameter value cannot be the result of an operation
  ```
  ## this will cause an error
  $greetings.charAt( 1 + 2 )
  ```

References

Properties and Methods

- In the context of Velocity, a Java property is of the form: `$var.PropertyName`
- In the User Guide of Apache, a property is always in Pascal case, though camelCase is also acceptable; example: `$greetings.Class`
- A property is normally a shorthand form of a method call of a getter
- For example, `$greetings.Class` is an abbreviated way of writing `$greetings.getClass()`, and `$greetings.Class.Name` is equivalent to `$greetings.getClass().getName()`
- Even though the first character of a property name can be in either uppercase or lowercase, the rest of the name is case-sensitive
- A method call is of the following form: `$x.methodName( argList )`
- The value(s) or object(s) passed into a method call are **argument(s)**
- `$greetings.getClass()` is an example of a method call through the `java.lang.String` object
- Note that both property names and method names are case sensitive: `$d.getDisplayName()` can be abbreviated to `$d.DisplayName` or `$d.displayName` but not `$d.displayname` ('N' must be in uppercase)
- According to the recommendation of Apache, whenever an object has a property, the property should be used without calling the corresponding method

Exposing Object Information: Using `$x.Class.Name`

- To expose the information of the object associated with a variable `$x`, use `$x.Class.Name` (or `$x.class.name`), where `$x` is the variable created by the `#set` directive

```
#set( $greetings = "Hello, World!" )
$greetings.Class.Name
```

- The previous code snippet outputs `java.lang.String`, showing that the value associated with the variable `$greetings` is a `java.lang.String` object
- This technique works for any variable assigned with a valid value
- If a variable, like `$var`, is not defined, then we will see an output like `$var.Class.Name` instead
- In general, if a name (of a variable, of a macro, etc.) is not associated with a valid value or code, then when the name is referenced, the name is output as is
- In later lessons, we will see that `$x.Class.Name` can be used to test whether a variable is defined

When the Value of `$x` Is Not an Object

- In Java, almost everything are objects
- There are a few exceptions: there are eight primitive types, including `int`, `double`, and so on, and there are Java arrays
- We normally don't deal with primitive values; they are always wrapped inside wrapper objects
- For example, an `int` value is normally wrapped inside a `java.lang.Integer` object
- In `#set( $int = 1 )`, the variable `$int` is associated with a `java.lang.Integer` object
- However, some Java methods return arrays
- For example, `com.hannonhill.cascade.api.adapters.PageAPIAdapter.getStructur` returns an array of `com.hannonhill.cascade.api.asset.common.StructuredDataNode` objects
- When a variable like `$x` is associated with an array, `$x.Class.Name` will be a String of the form `[L`*`X`*`;`
- There are three parts in this String: `[L`, `X`, and `;`, where `X` is the class name of the objects

- Example: the returned type of `com.hannonhill.cascade.api.adapters.PageAPIAdapter.getStructur` is `[Lcom.hannonhill.cascade.api.asset.common.StructuredDataNode;`
- This means that the `getStructuredData` method returns an array of objects of type `com.hannonhill.cascade.api.asset.common.StructuredDataNode`

The Velocity Context

- The Velocity context is a 'carrier' of data between the Java layer and the template layer (see Developer Guide)
- It is represented by a class
- It only contains objects
- Objects in the Cascade Velocity context:
- `$contentRoot`: representing the root element of the block XML when there is a block attached to the same region, which is an `org.jdom.Element` object

- For a data definition block, the name of the root element is `system-data-structure`
- For an index block, the name of the root element is `system-index-block`
- For a feed block, the root element is the same root element of the feed
- For an xml block, the root element is the same root element of the block XML
- When the source XML has no root element, like the textual content of a text block, the XML is wrapped up by the root element `system-xml`
- `$currentPage` representing the rendered current page (could be a draft), which is a `com.hannonhill.cascade.api.adapters.PageAPIAdapter` object
- `$_DateTool`
- `$_XPathTool`
- `$_SerializerTool`
- `$_SortTool`
- `$_StringTool`
- `$_DisplayTool`
- `$_EscapeTool`
- `$_MathTool`
- `$_NumberTool`
- `$_FieldTool`
- `$_` (Locator Tool)
- `$_PropertyTool`
- `$currentPagePath`: a `java.lang.String` object containing the path of the currently rendered page
- `$currentPageSiteName`: a `java.lang.String` object containing the name of the currently rendered page's site

- Other Java objects from various packages

Exposing Object Information: Introducing `$_PropertyTool`

- Update the format `hello-world` with the following content:

```
<pre>
$_PropertyTool.outputProperties( $greetings )
</pre>
```

- The output should show the following:

```
Object type: java.lang.String
Properties:
 - empty: boolean
 - getChars(intintchar[]int): void
 - bytes: byte[]
 - getBytes(String): byte[]
 - getBytes(intintbyte[]int): void
 - getBytes(Charset): byte[]
 - class: Class
```

- `$_PropertyTool` is a Java object of the type `com.hannonhill.cascade.velocity.PropertyTool`
- How do we find out information about `$_PropertyTool`? Try `$_PropertyTool.Class.Name`
- This object is made available to Velocity by Cascade
- `$_PropertyTool.outputProperties( $greetings )` is a method call through the `$_PropertyTool` object
- The `outputProperties` method of the `$_PropertyTool` object can be used to display object information of the argument object (in this case, a `java.lang.String` object)
- The displayed result shows some of the properties and methods defined in the class for the argument object
- `$_PropertyTool.outputProperties` can be used to display object information of any object; it can be used, for example, to display information of itself:

```
<pre>
```

```
$_PropertyTool.outputProperties( $_PropertyTool )
</pre>
```

```
Object type:
com.hannonhill.cascade.velocity.PropertyTool
Properties:
 - isEmpty(Object): boolean
 - class: Class
 - isNull(Object): boolean
```

- Note that `outputProperties` is not listed here
- From this output, we know that there is a `isNull` method associated with the `$_PropertyTool` object
- The `isNull` method can be used to test whether the argument object is `null`
- Example:
```
<pre>
$_PropertyTool.isNull( "Hello, World!" )
</pre>
```
```
false
```
- Note that the `java.lang.String` method `isEmpty()` can be abbreviated as `Empty` or `empty`, but the `isNull()` method of `$_PropertyTool` cannot be abbreviated as `Null` because it takes an argument
- `$_PropertyTool.outputProperties` is not a reliable tool because it can leave out critical information of an object
- For example, the `java.lang.String` class has dozens of methods, yet only seven are displayed when `$_PropertyTool.outputProperties( "Hello, World!" )` is called
- Although Hannon Hill always recommends the use of `$_PropertyTool` and `$_PropertyTool.isNull()` in your Velocity code, you can safely ignore this object and its methods
- We will see what can be used in its place later

Introducing the `java.lang.String` API

- Often we need to manipulate Strings retrieved from fields and nodes
- `$_PropertyTool` does not provide enough information to help us; we need to turn to the Java API
- Create a new format named `string` containing the following:
```
#set( $str1 = "Hello, World!" )
$str1.length()
```
- The page should show this:
```
13
```
  meaning that there are 13 characters in the String
- Here we use the `length()` method of the String object to get its length
- The `length()` method is not listed in the output by `$_PropertyTool`; how do we know that there is such a method available to us?
- To find out more about the `java.lang.String` class, we need to look at the java.lang.String

API

- Check the API to find useful methods like `contains`, `indexOf`, and `trim`; none of these are listed in the output produced by `$_PropertyTool.outputProperties()`
- This is the reason why we need to get ourselves familiarized with the `java.lang.String` API
- To be more general, we need to consult various API documentation:
- To find out what methods are available (i.e., defined) in a class
- Equally important, to find out what are NOT defined in a class (see locatorTool problem

)

- A note about the `length()` method: because the name of this method does not contain the `get` or `is` preface, there is no `Length` property in the `java.lang.String` class

Introducing `org.jdom.Element`

- Create a new format `element` containing the following:
```
<pre>
$_PropertyTool.outputProperties( $contentRoot )
</pre>
```
- Attach the format to a page, and also attach the XML block named `hello-world` to the same region of the page
- The page should show a long list of properties of the [`org.jdom.Element`]

object `$contentRoot`

- Change the character `R` in `$contentRoot` to lowercase `r`
- What happens to the page? Why?
- `$contentRoot` is a Java object representing the root element of the XML stored in a block
- To access any part of the XML, we must start with this object
- That is to say, whenever a format is used to transform block data, we will need this `$contentRoot` object
- Since all `org.jdom.Element` objects have names, we can output the name of `$contentRoot` (`$contentRoot.Name` or `$contentRoot.getName()`), and what do we see?
- We see `greetings`
- Since we know that this element also has a value, we can output the value by using `$contentRoot.Value` or `$contentRoot.getValue()`
- Use `$_PropertyTool.outputProperties( $contentRoot )` again to output the list of properties, and make sure that you can

find both `name` and `value` in the list

- If you discover that a property you think should work does not work, then use the getter instead, and make sure that the getter is defined in the class and you can find it in the API
- This is particularly true if the getter requires a parameter
- Whenever a property is available, use the property instead of the getter (recommended by Apache)
- As pointed out before, `$_PropertyTool.outputProperties()` is not reliable
- We should look at the API documentation of a class instead

Reading the API Documentation

- Look at the `java.lang.String` API again
- Make sure the `All Methods` tab in `Method Summary` is lit up
- The first method we see is `charAt` and here is the summary:
  `char   charAt(int index)`

```
      Returns the char value at the specified
index.
```

- `char` is the return type, meaning the type of the returned value
- `charAt` is the method name
- `int index` indicates that this method takes an integer (not an `Integer` object) as the single argument
- The summary describes what this method does
- Below the summary, we can find detailed description of each method
- Check the documentation of `org.jdom.Element`

Manipulating `String` and `Element` Objects

- Why we want to manipulate Strings:
- When we process a data definition block, for example, we may want to process elements whose names have a common prefix or suffix
- Data or metadata may contain some kind of CSV String, and we

want to find out if the String contains a certain value before we proceed

- The whole idea of bricks bases upon String replacement
- See Formatting a submitted URL

  for Arrington's comments
- Before we proceed with more exercises, let us set up another set of format, XML block, and page, all named `bricks`, and attach both the block and format to the page
- We want to play with idea of bricks
- Put the following content in the XML block:

```
<system-data-structure>
    <brick-group>
        <brick-identifier>president</brick-
identifier>
        <brick-value>John Doe</brick-value>
    </brick-group>
</system-data-structure>
```

- Examine the XML structure of this code:

```
system-data-structure
    brick-group
        brick-identifier
        brick-value
```

- Using XML jargon, `system-data-structure` is the parent of `brick-group`, and both `brick-identifier` and `brick-value` are children of `brick-group`
- Note that only `brick-identifier` and `brick-value` have text values
- Recall that `$contentRoot` is an object of type `org.jdom.Element`
- Important methods of `org.jdom.Element`:
- `Element getChild(String name)`
- `String getName()`
- `Element getParent()`
- `String getValue()`
- Before start calling methods, let us try `$contentRoot.Class.Name` to make sure that the block is indeed attached to the page
- When a block containing XML data is attached to a region, the root element of the XML is associated with the variable `$contentRoot`, which is given to us by Cascade, and there is no need to retrieve the root element
- Code of the following type is totally pointless:
  ```
  #set( $data = $_XPathTool.selectSingleNode(
  $contentRoot, "/system-data-structure" ) )
  ```
  because both `$data` and `$contentRoot` are associated with the same root element
- Since `$contentRoot`, in this case, the `system-data-structure` element, is given to us by Cascade, we can call the `getChild` method to get the `brick-group` element
- With the `brick-group` element, we can call `getChild` again, twice, to get the `brick-identifier` and `brick-value`

elements

- Let us add code to the format:

```
#set( $group = $contentRoot.getChild( "brick-
group" ) )

<pre>
$group.name
</pre>
```

  Here we are doing something unnecessary. We already know that the name of the element is `brick-group`. That is how we get it in the first place. But still, we output its name in the `pre` element to show that we have succeeded getting the element.

- Next, we want to get the values of `brick-identifier` and `brick-value`:

```
#set( $group = $contentRoot.getChild( "brick-
group" ) )
#set( $id    = $group.getChild( "brick-identifier"
).Value )
#set( $value = $group.getChild( "brick-value"
).Value )

<pre>
$group.name
$id
$value
</pre>
```

  We should see the following on the page:

```
brick-group
president
John Doe
```

- This is the idea of bricks: we insert a pseudo-variable into text fields and WYSIWYGs, using something like "--president--", and we want to replace the pseudo-variable with a value that is maintained independently

- With the brick in place, we want to replace all instances of "--president--" in a String we make up (it could have come from a text field or a WYSIWYG in a block) with the brick value:

```
#set( $group = $contentRoot.getChild( "brick-group" ) )
#set( $id    = $group.getChild( "brick-identifier" ).Value )
#set( $value = $group.getChild( "brick-value" ).Value )
#set( $st    = "Today, President --president-- has come to New York city. " +
    "And the major gave President --president-- a warm welcome." )
#set( $var   = "--" + $id + "--" )

<pre>
$group.name
$id
$value
$var
$st.replaceAll( $var, $value )
</pre>
```

  And this is the result:

```
brick-group
president
John Doe
--president--
Today, President John Doe has come to New York city. And the major gave President John Doe a warm welcome.
```

- Comments on the code:
- Because `$group.getChild( "brick-identifier" )` returns an `org.jdom.Element` object, and such an object has a `Value` property, we can access the property by tagging `.Value` right after `$group.getChild( "brick-identifier" )`
- The + operator is used to concatenate Strings

- We call the `replaceAll` method of `java.lang.String` to replace all instances of the pseudo-variable with the brick value
- The first argument of `replaceAll` is a regex, hence we must be very careful about what to use for the pseudo-variable, and characters like "/" or "[" should not be used
- I output all the variables just to make sure everything is working OK

A Note About `getChild` and `getChildren`

- An element may have more than one child bearing a certain name
- When `getChild(name)` is called upon this element, only the first child bearing the name will be returned
- If we need to get all children bearing that name, then we need to use `getChildren(name)` instead
- To retrieve all children, regardless of names, use the property `.Children` (e.g., `$contentRoot.Children`)
- To access the first child, use code like `$contentRoot.Children[ 0 ]`

Introducing the `java.time.LocalDate`

- The `java.time` package was introduced in Java 1.8
- Objects instantiated from classes in this package, unlike objects of type `java.util.Calendar`, are immutable and thread-safe
- The Upstate Velocity library provides objects and macros to utilize the package
- We will look at `java.time.LocalDate`

briefly

- Invoke `#chanwLocalDateFromTimestampString` to turn a timestamp String into a `java.time.LocalDate` object
- Out of a `java.time.LocalDate` object, the `atTime` method can be called to create a `java.time.LocalDateTime` object
- The `format` method can be called to format `java.time.LocalDate` object
- `java.time.LocalDate` objects can be compared and differences can be computed
- Methods like `plus`, `minus`, and `with` can be used to create more `java.time.LocalDate` objects
- local-date.vm

About the Upstate Velocity Library

- Many macros defined in the library "return" useful values
- In Velocity, macros cannot really return anything
- Instead, I use a variable named after its corresponding macro to store the "returned" value
- For example, the macro `#chanwLocalDateFromTimestampString` "returns" a `java.time.LocalDate` object
- After the macro is invoked, the object is stored in a variable named `$chanwLocalDateFromTimestampString`
- We may want to invoke `#chanwLocalDateFromTimestampString` multiple times
- Every time when the macro is invoked, a new value is created, overwriting any old value stored
- Therefore, if the macro is invoked more than once, we want to store separate values in different variables for later consumption:

```
#chanwLocalDateFromTimestampString(
$_DateTool.SystemTime )
#set( $localDateNow =
$chanwLocalDateFromTimestampString )

#set( $timestamp = "1501466917465" )
#chanwLocalDateFromTimestampString( $timestamp )
#set( $localDateThen =
$chanwLocalDateFromTimestampString )
```

Examples

- introductory/02_java_and_java_objects

References

- java.lang.String

- org.jdom.Element
- java_objects.vm

- code_vs_String.vm

Challenges

- Can we create Java objects by invoking constructors?
- Can we list all methods defined in a class without using `$_PropertyTool`?