BuildingASimpleCard

Matthew Bregg

December 6, 2015

Contents

Overall, these slides go over the heart of the script evaluator, which is the process of which a ScriptObject is constructed, which is the same for all builders.

1 ScriptObject slides

ScriptObject serves as the equivelent of Javas object, but for our language. contains an accept method for the builder visitor, which allows the things to be added to the builder to be added by visiting.

2 ScriptObjectBuilder

The Builder for ScriptObjects.

As the various types in the scripting language can have varying ways to be constructed, and returns varying representations, but use the same overall process, that is, choosing the representation to make, and then, for each argument, using the visitor pattern to have the argument added to the builder.

3 Language Slides

3.1 Slide 1

- Expression token
  - cons keyword Expression token has the cons keywork, so this builder will be retrieved from the map
  - Then, for each argument in the token, run doParse again on it.
    - These are both constants, with no arguments, so the result from the tokens builder is returned immediately.
  - These two script objects are added immediately to the cons.
  - The builder got what it expected, and can successfully make the and return cons when getResult() is called.

```
(cons 1 "foo")
(list 1 2 "foo" "bar")
```

3.2 Slide 2

- This is the same as before, but rather than both tokens being constant tokens, one is an expression token.
- Thus, doParse is ran on (cons 2 3) as above, and a ScriptObject is is returned. (A conspair). This conspair is added to the outer cons, with the script object holding
  1. (DoubleScriptWrapper)
- The builder can then make the outer cons containing these two objects.
- getResult() will then return that script object.

```
(cons 1 (cons "foo" 3))
```

3.3 Slide 3

- So far, we have only cared about getting something.
  - Cons just needs a ScriptObject.
  - layout, and image, are a bit pickier.
  - They use the visitor pattern to see what is given to them, and may return an error if they are not given what they expect.
  - This is a runtime error in our dynamic language.

```
(layout (image "foo.jpg"))
```

3.4 Slide 4

- More complex example, but still follows the same process..

```
(layout (image "foo.jpg"))

(layout
 (list (layout (color-decal "green")) whole-layout)
 (list fooLayout (position-scaled 50 50 50 50) (rectangle 1 1))
 )
```

3.5 Slide 5

- So far, everything written has been garbage collected right after creation. Define playes something into the environment as a variable, which can be used later.

```
(define fooLayout (layout (image "foo.jpg")))

(define barLayout
  (layout
   (list (layout (color-decal "green")) whole-layout)
   (list fooLayout (position-scaled 50 50 50 50) (rectangle 1 1))
   )
  )
```

3.6 Slide 6

Expanding on define, to show a function.

```
  (define (anything image-file)
    (layout
     (list (layout (color-decal "green")) whole card)
     (list (layout (image image-file))  (position-scaled 50 50 50 50))
     )
    )
(anything "foo.jpg")
(anything "bar.jpg")
```

3.7 Slide 7

More complex function, and showing the languages representation of a card.

```
(define (anything image-file)
  (layout
   (list (layout (color-decal "green")) whole card)
   (list (layout (image image-file))  (position-scaled 50 50 50 50))
   )
  )
  (card (size 200 100) name (anything "foo.jpg") (anything "bar.jpg") (rectangle 100 10
```

3.8 Slide 8

Finally, render. Render will cause a card given to it to be... rendered into images, and placed into a rendered list. This list will then be iterated through and output to disk by another part of the design. The card created by the function, and render itself can then be safely collected.

```
(define (make-foo-layout image-file)
  (layout
   (list (layout (color-decal "green")) whole card)
   (list (layout (image image-file))  (position-scaled 50 50 50 50))
   )
  )
(define (make-card image-file name)
  (card
   (size 200 100)
   name
   (make-foo-layout image-file)
   (make-foo-layout image-file)
   (rectangle 100 100))
)

(render (make-card "foo.jpg" "Foo"))
(render (make-card "bar.jpg" "bar"))
```