

BUILDING MODERN DSLS IN GROOVY

by Cédric Champeau (@CedricChampeau)

WHO AM I

```
speaker {  
    name 'Cédric Champeau'  
    company 'Gradle Inc'  
    oss 'Apache Groovy committer',  
    successes 'Static type checker',  
              'Static compilation',  
              'Traits',  
              'Markup template engine',  
              'DSLs'  
    failures Stream.of(bugs),  
    twitter '@CedricChampeau',  
    github 'melix',  
    extraDescription '''Groovy in Action 2 co-author  
Misc OSS contribs (Gradle plugins, deck2pdf, jlangdetect, ...)'''  
}
```



DISCLAIMER

This is an *opinited* talk about how a DSL designed with Apache Groovy should look like.



DOMAIN SPECIFIC LANGUAGES

- Focused
- Readable
- Practical
- (usually) embeddable
- Examples: SQL, HTML, XSLT, Ant, ...

APACHE GROOVY FOR DSLS

- Concise, clean syntax
- Supports scripting
- Supports metaprogramming
- Embeddable
- Mature tooling: Eclipse, IntelliJ, Netbeans...

Lots of docs available at <http://docs.groovy-lang.org/latest/html/documentation/core-domain-specific-languages.html>

SOME OLD GROOVY DSLS

INTELLIJ IDEA'S GDSL

```
contributor(context()) {  
    if (classType?.hasAnnotation("groovy.lang.Singleton")) {  
        property name: "instance",  
                 type: classType?.getQualifiedName() ?: "java.lang.Object",  
                 isStatic: true  
    }  
}
```

GRAILS DYNAMIC FINDERS

```
def persons = Person.findByName('Stark')
assert persons.findAll {
    it.alive
}.isEmpty()
```

GRADLE TASK EXECUTION

```
task(hello) << {  
    println "hello"  
}
```

VS

```
task(hello) {  
    println "hello"  
}
```

SOME THOUGHTS

- removing *semicolons* is not designing a DSL
- removing *parenthesis* is not designing a DSL
- **user experience** is important
- **consistency** is important
- Try to be idiomatic

MODERN APACHE GROOVY DSLs

SPOCK

```
given:  
def shell = new GroovyShell()  
  
when:  
def sum = shell.evaluate(  
    @groovyx.ast.bytecode.Bytecode  
    int sum(int limit) {  
        l0:  
        iconst_0  
        istore_2  
        ...  
    }  
)  
  
then:  
    sum(i) == reference  
where:  
    i | reference  
    0 | 0  
    1 | 0  
    2 | 1  
    3 | 3  
    4 | 6
```

GRAILS 3 WHERE QUERIES

```
assert Person.findAll {  
    lastName == 'Stark' && alive  
}.isEmpty()
```

GRADLE NEW MODEL

```
model {
    components {
        main(JvmLibrarySpec) {
            sources {
                java {
                    dependencies {
                        library 'myLib'
                    }
                }
            }
            targetPlatform 'java8'
        }
        myLib(JvmLibrarySpec) {
            api {
                exports 'compile,test'
            }
            targetPlatform 'java5'
            targetPlatform 'java6'
        }
    }
}
```

RATPACK

```
ratpack {  
    handlers {  
        get {  
            render "Hello World!"  
        }  
        get(":name") {  
            render "Hello ${pathTokens.name}!"  
        }  
    }  
}
```

JENKINS JOB DSL

```
job {  
    using 'TMPL-test'  
    name 'PROJ-integ-tests'  
    scm {  
        git(gitUrl)  
    }  
    triggers {  
        cron('15 1,13 * * *')  
    }  
    steps {  
        maven('-e clean integTest')  
    }  
}
```

MARKUPTEMPLATEENGINE

```
modelTypes = {
    List<String> persons
}

html {
    body {
        ul {
            persons.each { p ->
                li p.name
            }
        }
    }
}
```

See [docs for the template engine](#)

IMPLEMENTING MODERN DSLs

THE TOOLS

- Closures with support annotations (@DelegatesTo, ...)
- Compilation customizers
- AST transformations
- Type checking extensions
- Groovy Shell / Groovy Console

CLOSURES

- Still at the core of most DSLs
- delegate is very important:

```
['Paris', 'Washington', 'Berlin'].collect {  
    it.length() == 5  
}
```

- do we really need it?

SETTING THE DELEGATE

See [extension modules docs](#).

```
class HelperExtension {  
    public static <T,U> List<U> myCollect(  
        List<T> items,  
        Closure<U> action) {  
        def clone = action.clone()  
        clone.resolveStrategy = Closure.DELEGATE_FIRST  
        def result = []  
        items.each {  
            clone.delegate = it  
            result << clone()  
        }  
        result  
    }  
}  
  
HelperExtension.myCollect(['Paris', 'Washington', 'Berlin']) {  
    length() == 5  
}
```

CONVERT IT TO AN EXTENSION MODULE

- META-INF
 - services
 - org.codehaus.groovy.runtime.ExtensionModule

```
moduleName=My extension module
moduleVersion=1.0
extensionClasses=path.to.HelperExtension
```

CONVERT IT TO AN EXTENSION MODULE

- Consume it as if it was a regular Groovy method

```
['Paris', 'Washington', 'Berlin'].myCollect {  
    length() == 5  
}
```

DECLARE THE DELEGATE TYPE

- Best IDE support
- **Only** way to have static type checking

```
public static <T,U> List<U> myCollect(  
    List<T> items,  
    @DelegatesTo(FirstParam.FirstGenericType)  
    Closure<U> action) {  
    ...  
}
```

REMOVING CEREMONY

- Is your DSL self-contained?
- If so
 - Embrace SAM types
 - Try to remove explicit imports
 - Avoid usage of the new keyword
 - Avoid usage of annotations

SAM WHAT?

This is ugly:

```
serve(new Handler() {  
    @Override  
    void handle(String message) {  
        println message  
    }  
})
```

SAM WHAT?

This is cool:

```
serve {  
    println message  
}
```

SAM type coercion works for both interfaces and abstract classes.

COMPIILATION CUSTOMIZERS

```
class WebServer {  
    static void serve(@DelegatesTo(ServerSpec) Closure cl) {  
        // ...  
    }  
}
```

COMPILATION CUSTOMIZERS

```
def importCustomizer = new ImportCustomizer()
importCustomizer.addStaticStars 'com.acme.WebServer'

def configuration = new CompilerConfiguration()
configuration.addCompilationCustomizers(importCustomizer)

def shell = new GroovyShell(configuration)
shell.evaluate '''
serve {
    port 80
    get('/foo') { ... }
}
'''
```

COMPIILATION CUSTOMIZERS

- `ImportCustomizer`: automatically add imports to your scripts
- `ASTTransformationCustomizer`: automatically apply AST transformations to your scripts
- `SecureASTCustomizer`: restrict the grammar of the language
- `SourceAwareCustomizer`: apply customizers based on the source file
- See [docs for customizers](#)

TYPE CHECKING EXTENSIONS

GOALS

- Provide **early** feedback to the user
- Type safety
- Help the compiler understand your DSL

TYPE CHECKING EXTENSIONS API

- Event-based API
- React to events such as *undefined variable* or *method not found*
- Developer instructs the type checker what to do

```
methodNotFound { receiver, name, argList, argTypes, call ->
    if (receiver==classNodeFor(String)
        && name=='longueur'
        && argList.size()==0) {
        handled = true
        return newMethod('longueur', classNodeFor(String))
    }
}
```

TYPE CHECKING EXTENSIONS

- Powerful tool but focused on the Groovy compiler
- See [documentation](#)

MARKUPTEMPLATEENGINE EXAMPLE

- Given the following template

```
pages.each { page ->
    p("Page title: $page.title")
    p(page.text)
}
```

- How do you make sure that pages is a valid model type?
- How do you notify the user that page doesn't have a text property?
- How to make it **fast**?

SOLUTION

- Declare the model types

```
modelTypes = {  
    List<Page> pages  
}  
  
pages.each { page ->  
    p("Page title: $page.title")  
    p(page.text)  
}
```

- Implement a *type checking extension*

MARKUPTEMPLATEENGINE EXTENSION

- Recognizes unresolved method calls
 - converts them into direct *methodMissing* calls
- Recognizes unresolved variables
 - checks if they are defined in the binding
 - if yes, instructs the type checker what the type is

MARKUPTEMPLATEENGINE EXTENSION

- Applies @CompileStatic transparently
- Performs post-type checking transformations
 - Don't do this at home!

(OPTIONAL) @CLOSUREPARAMS

- For type checking/static compilation

```
['a', 'b', 'c'].eachWithIndex { str, idx ->
  ...
}
```

(OPTIONAL) @CLOSUREPARAMS

```
public static <T> Collection<T> eachWithIndex(  
    Collection<T> self,  
    @ClosureParams(value=FromString.class, options="T, Integer")  
    Closure closure) {  
    ...  
}
```

Check out the documentation for more details.

WHAT WE LEARNT

- Leverage the lean syntax of Groovy
- Scoping improves readability
- Use the *delegate*
- Use @DelegatesTo and @ClosureParams for IDE/type checker support
- Use imperative style as last resort
- Help yourself (builders, specs vs impl, ...)

QUESTIONS



WE'RE HIRING!

<http://gradle.org/gradle-jobs/>



THANK YOU!

- Slides and code : <https://github.com/melix/javaone-groovy-dsls>
- Groovy documentation : <http://groovy-lang.org/documentation.html>
- Follow me: [@CedricChampeau](https://twitter.com/CedricChampeau)