Configuration, Logging, and Plugins

Section goals

After this section, you will be able to

- Centrally configure your Grails application
- Enhance your application's functionality with Grails plugins
- Add Java and Groovy libraries to your Grails application

Config.groovy

grails-app/conf/Config.groovy

- Centralized location for application configuration

- Types of configuration
  - Enabling or disabling functionality (ex: email sending)
  - Specifying feature configuration values (ex: security 'user' class name)

http://grails.org/doc/latest/guide/conf.html#config

DataSource.groovy

grails-app/conf/DataSource.groovy

Specifies your database connection information

- Init operation ("create-drop", "update", "none", etc.)
- Database driver class
- JDBC connection URL
- DB username/password

DataSource example

```
dataSource {
    dbCreate = "update"
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://localhost/my_app"
    username = "root"
    password = ""
}
```

Grails environments

development

- Default when running 'grails run-app'

test

- Active when running integration and functional tests

production

- Default when running from a .war file

Environment - specific configuration

```
environments {
  development {
    grails.serverURL = "http://localhost:8080/myapp"
    ...
  }
  test {
    grails.serverURL = "http://localhost:8080/myapp"
    ...
  }
  production {
    grails.serverURL = "http://myapp.com"
    ...
  }
}
```

External config files

```
grails.config.locations = [
  "file:${userHome}/.grails/${appName}.Config.groovy",
  "file:${userHome}/.grails/${appName}.DataSource.groovy"
]

if (System.properties["${appName}.config.location"]) {
  grails.config.locations << "file:" + System.properties["${appName}.
config.location"]
}

if (System.properties["${appName}.datasource.location"]) {
  grails.config.locations << "file:" + System.properties["${appName}.
datasource.location"]
}
```

Custom config values

Config.groovy:

```
my.custom.value = "hello world"
my {
  value = "closure value"
}
```

Access custom value:

```
class MyController {
  def grailsApplication

  def myAction() {
   def customValue = grailsApplication.config.my.custom.value
   ...
  }
}
```

Command line properties

Certain properties can be set via the Grails command line grails -D<param_name>=value command

```
> grails -Dgrails.serverURL=http://localhost:8081/myapp run-app
```

Logging configuration

Config.groovy:

```
log4j = {
  appenders {
    ...
  }

  error ...
  warn ...
  info ...
  debug ...
}
```

Logging output - console

Console output:

```
appenders {
  console name:'stdout'
}
```

Log4J

http://logging.apache.org/log4j/

Grails logging documentation

http://grails.org/doc/latest/guide/conf.html#logging

Logging output - file

File output:

```
appenders {
  console name:'stdout'
  file name: 'file', file: "application.log"
}

root {
  error 'stdout', 'file'
}
```

Logging levels

Debug logging for all controllers, conf (Bootstrap, Config, etc.), and services:

```
debug 'grails.app.controllers',
      'grails.app.conf',
      'grails.app.services'
```

Debug logging for controllers in a specific package:

```
debug 'grails.app.controllers.com.myapp.package'
```

Log messages

Logger instance automatically available in controllers, services, taglibs, etc.

- Roughly everything under /grails-app/

```
class MyController {
  def myAction() {
    log.debug("Executing myAction")

    try {
      ...
    } catch (Exception e) {
       log.error("Triggered exception trying my action.", e)
    }
  }
}
```

Logging workshop

- Configure console logging
- Configure logging to file 'application.log'
- Enable 'debug' log level for all controllers, configuration, and services
- Add 'debug' logging message to the end of Bootstrap.groovy
- Run application with: `grails run-app`
  - Verify log output

Workshop results

```
appenders {
  ...
  console name: "stdout", layout: pattern
  file name: "file", file: "application.log", layout: pattern
}

root {
  error "stdout", "file"
}

debug "grails.app.controllers",
      "grails.app.conf",
      "grails.app.services"
```

Plugins

- Easily add into your application a piece functionality that is not included in the core Grails framework
- Over 1100 plugins on Grails Plugins Portal
  - http://grails.org/plugins
- Some of the most popular plugins (Hibernate, Mail, Spring Security, etc.) are supported by SpringSource.
- Most plugins have open-source licenses
  - Commonly Apache 2 license (same license as Grails)

Plugin structure

Plugins have same internal structure as Grails applications

- Controllers, services, views, etc.

Versus Jar dependencies, which only contain standard class files

Installing plugins

grails-app/conf/BuildConfig.groovy

Can control scope of plugin ('compile', 'test', etc.)

```
plugins {
    runtime ":database-migration:1.4.0"
}
```

Can use variables in plugin dependency:

```
plugins {
  compile ":hibernate:${grailsVersion}"
}
```

Upgrading plugins

Manually edit plugin version in BuildConfig.groovy

- Next time execute Grails command, Grails asks to upgrade plugin

To list plugins that have newer versions, run command:

```
> grails list-plugin-updates
```

Removing plugins

Remove plugin definition from BuildConfig.groovy

Next app run, Grails will ask if you want to remove the plugin

Database Migration Plugin

- Check BuildConfig.groovy for the ‘database-migration’ plugin.
- See plugin sources downloaded to

  `questionApp/target/work/plugins/`
- This plugin uses Liquidbase to help you manage changes to your domain model

Migrating DB Changes - Liquibase

Change Datasource.groovy

```
dbCreate = "create"
```

Make H2 a file based database

```
url = "jdbc:h2:file:devDb;MVCC=TRUE;LOCK_TIMEOUT=10000;
DB_CLOSE_ON_EXIT=FALSE"
```

Generating the Change log

Generate the initial changelog before making changes:

```
> grails dbm-generate-changelog changelog.groovy
```

Change Datasource.groovy

- comment out dbCreate in the development environment

```
// dbCreate = "create-drop"
```

Mark Existing Schema as Sync'ed

Since no changes have been made, mark it as synchronized:

```
> grails dbm-changelog-sync
```

Record Schema Changes

Add new property to Answer

```
boolean accepted
```

Use database-migrations to view the changes

Output to the screen:

```
> grails dbm-gorm-diff
```

Output to file:

```
> grails dbm-gorm-diff add-accept-answer.groovy --add
```

Apply Schema Changes

```
> grails dbm-update
```

Applies all changes since last sync.

Using DB Migrations/Liquibase

- In a greenfield app, don't start too early, wait until your domain model is somewhat stable.
- Does NOT work well with H2 in memory database. It would be best to work with the database you will be using in production, MySQL, Postgres, etc.

Dependencies - BuildConfig.groovy

- Add dependencies from Maven repositories
- Since Grails 2.3 uses Aether, the dependency resolution library used by Maven.
- Earlier version use Ivy-style syntax. If you prefer the Ivy syntax you can still use it.

BuildConfig.groovy

```
grails.project.dependency.resolver = "maven" // or ivy
```

Dependencies

- Specify dependencies under either ‘build’, ‘compile’, ‘runtime’, ‘test’, or ‘provided’
- Aether uses the Maven pattern of

```
<groupId>:<artifactId>[:<extension>[:<classifier>]]:<version>
```

```
runtime 'mysql:mysql-connector-java:5.1.29'
```

Specify Maven repositories

BuildConfig.groovy:

```
repositories {
  // Default repository locations
  grailsPlugins()
  grailsHome()
  grailsCentral()
  mavenCentral()

  // Custom repository locations
  mavenRepo "http://myrepo.com:8080"
}
```

Exclude transitive dependencies

Example: GMock has a transitive dependency on an older version of JUnit

- Would rather exclude that transitive dependency and use the JUnit that comes with Grails

```
test ('org.gmock:gmock:0.8.2') {
  excludes 'junit'
}
```

Dependencies - common version

Want to keep the versions of several dependencies in sync?

- For example, Selenium drivers

Declare a variable in BuildConfig.groovy

```
selenium.version="2.15.0"

dependencies {
  test "org.seleniumhq.selenium:selenium-support:${selenium.version}"
  test "org.seleniumhq.selenium:selenium-chrome-driver:${selenium.version}"
  test "org.seleniumhq.selenium:selenium-firefox-driver:${selenium.version}"
}
```

Dependency report

- Grails can generate project dependency report

```
> grails dependency-report
```

- Report written to `target/dependency-report/index.html`
- Ivy dependency information for all dependencies in project

Summary

- Application configuration: Config.groovy
- Database connection configuration: DataSource.groovy
- Install/upgrade plugins: BuildConfig.groovy
- Add .jar libraries: BuildConfig.groovy