GR8Conf EUROPE 2015

June 2nd - 4th 2015
Copenhagen, Denmark

GRAILS Groovy gradle

Event-driven plugins with Grails 3

Göran Ehrsson, Technipelago AB

Technipelago AB

Göran Ehrsson, @goeh

- Grails enthusiast
- Founded Technipelago 2006
- Custom business applications
- 90% of customer base running Grails apps
- Main contributor to GR8 CRM plugin collection

Custom business applications (web & mobile)

- Different industries but common requirements
  - Customers
  - Projects
  - Tasks / Calendar
  - Documents
  - …

DID YOU SAY
COPY & PASTE?!
memegenerator.net

Grails Plugins

- Plugins extend the platform. A plugin can:
    - extend the data model
    - add services
    - provide static resources
    - add command line scripts
    - do a lot more…
- The plugin framework provides lots of extension points

Separation of Concern

- Each plugin should focus on one task or domain
- A plugin should be tested isolated from others
- Grails plugins make the boundaries strong and well defined.
- They force the developer to stay inside the box

Keep services and UI in separate plugins

- Put logic in the service layer, not in view controllers
- You may want to have different user interface plugins for different requirements
- The same service plugin can be used in both the web-front application and in back-office without exposing admin UI to web-front
- You can use the same service plugin in rich client or micro service style applications

Communicate with events

- Spring has built-in support for both synchronous and asynchronous application events
- Spring Integration includes advanced event support
- Grails 2 platform-core plugin includes great event handling (synchronous, asynchronous, event reply)
- Grails 3 includes event support based on the Reactor library

plugin-grails-events

- Core plugin included in the Grails 3 distribution
- Based on the Reactor library
- Not the same as the grails-events plugin by @smaldini

Reactor

- Reactor is a foundational library for building reactive fast-data applications on the JVM.
- You can use Reactor to power an application that has a low tolerance for latency and demands extremely high throughput.
- It’s really fast. On a recent laptop with a dual-core processor, it's possible to process over 25,000,000 events per second in a single thread.
- It is an implementation of the Reactive Streams Specification.

Reactive Streams

- Reactive Streams is a standard and specification for Stream-oriented libraries for the JVM that;
  - process a potentially unbounded number of elements in sequence
  - asynchronously passing elements between components
  - with mandatory non-blocking backpressure

Grails 3 Events API

- Grails events plugin statically injects Events API methods in Grails Controllers and Services using a trait called Events. You can implement this trait in other artefacts.
- Consume events
  - on(key) { /* handle event */ }
- Publish events
  - notify(key, data)
  - sendAndReceive(key, data) { reply -> /* handle reply */ }
  - def event = eventFor(Map headers, Object data)
    - notify(key, event)

Spring Reactor Support

- @reactor.spring.context.annotation.Consumer
- @reactor.spring.context.annotation.Selector
- reactor.spring.context.annotation.SelectorType
  - SelectorType.OBJECT (default)
  - SelectorType.REGEXP
  - SelectorType.URI
  - SelectorType.JSON_PATH

```
class WebOrderService {
    @Transactional
    void confirm(Long id) {
        def orderInstance = WebOrder.get(id)
        if(orderInstance) {
            orderInstance.status = OrderStatus.CONFIRMED
            if(orderInstance.save()) {
                notify(”order.confirmed”,
                [order: id, email: orderInstance.customerEmail])
                render "Thank you for ordering!"
            }
        }
    }
}
```

```
@Consumer
class MyApplicationService {

    @Selector("order.confirmed")
    void sendConfirmationEmail(Event<Map> event) {
        Map data = event.getData()
        String subj = "Order confirmation"
        String text = parseTemplate("order.template", data)
        sendMail {
            to: data.email
            from: "info@company.com"
            subject: subj
            body: text
        }
    }
}
```

Selecting events based on regular expression

```
@Selector(value = /(.+)\.created/, type = SelectorType.REGEX)
void somethingWasCreated(Event<Object> event) {
    println "${event.headers.group1} was just created"
}
```

Event replies

Consumers should return void and use event.reply(data) to reply results back to the sender.

```
def userList(String department) {
    List result = []
    CountDownLatch latch = new CountDownLatch(1)
    sendAndReceive("user.list", department) { Event reply ->
        result = reply.data
        latch.countDown()
    }

    latch.await(5, TimeUnit.SECONDS)

    [list: result]
}
```

Transactions

```
class WebOrderService {
    @Transactional
    void confirm(Long id) {
        def orderInstance = WebOrder.get(id)
        if(orderInstance) {
            orderInstance.status = OrderStatus.CONFIRMED
            if(orderInstance.save()) {
                notify(”order.confirmed”,
                [order: id, email: orderInstance.customerEmail])
                render "Thank you for ordering!"
            }
        }
    }
}
```

Transaction bound events

Spring 4.2 (planned release July 2015) will have the ability to bind event listeners to a phase of the transaction.

Unfortunately Spring's application events are not based on Reactor, so we will still have two different event implementations in Grails.

Send event after commit

```
@Transactional
WebOrder createOrder() {
    final WebOrder orderInstance = new WebOrder()
    // populate order items and save.
    afterCommit {
        notify('order.created', orderInstance.id)
    }
    return orderInstance
}

private void afterCommit(final Closure task) {
    TransactionSynchronizationManager.registerSynchronization(
    new TransactionSynchronizationAdapter() {
        @Override
        void afterCommit() {
            task()
        }
    })
}
```

Migrating from Grails 2 to Grails 3

Grails 3.0 is a complete ground up rewrite of Grails and introduces new concepts and components for many parts of the framework.

- Based on Spring Boot
- Gradle is now used to build your Grails application
- Project structure differences
- File location differences
- Configuration differences
- Package name differences
- Legacy Gant Scripts
- Changes to Plugins

See https://grails.github.io/grails-doc/latest/guide/upgrading.html

Migrating plugins

- The plugin descriptor which was previously located in the root of the plugin directory should be moved to the "src/main/groovy" directory under an appropriate package

- Same file structure changes as with applications

- It's recommended to publish public/official plugins to Bintray

migrate2-grails3 plugin

- The migrate2-grails3 plugin performs a partial migration of a Grails 2 plugin or app to Grails 3
- gvm use grails (latest 3.x version)
- grails create-plugin myplugin
- gvm use grails (your 2.x plugin/application version)
- BuildConfig.groovy in the Grails 2.x project:
    - compile ”:migrate2-grails3:<latest version>"
- grails migrate ../../grails3/myplugin

Migrating from platform-core events to Grails 3 (reactor) events

```
// Publishing events with platform-core
event(for: "order", topic: ”confirmed",
  data: [order: order.id, email: customerEmail])


// Publishing events with Grails 3
notify(”order.confirmed",
  [order: order.id, email: customerEmail])
```

Migrating from platform-core events to Grails 3 (reactor) events

```
// Consuming events with platform-core
class FooService {
  @Listener(namespace = "order", topic = "confirmed")
  def sendConfirmationEmail(data) { ... }
}
// Consuming events with Grails 3
@Consumer
class FooService {
  @Selector("order.confirmed")
  def sendConfirmationEmail(Event<Map> event) { ... }
}
```

Plugin authors

Start your migration engines!

Misc caveats

’pluginExcludes’ does not work the same way in Grails 3.0.1. But you can add jar excludes in build.gradle instead.

```
jar {
  exclude "com/demo/**/**"
  exclude "demo/**"
}
```

Misc caveats

If domain mapping declares 'cache' options you must add cache region factory in application.yml.

```
static mapping = {
    cache 'nonstrict-read-write'
}
```

```
hibernate:
  cache:
    use_second_level_cache: true
    provider_class: net.sf.ehcache.hibernate.EhCacheProvider
    region:
       factory_class: org.hibernate.cache.ehcache.EhCacheRegionFactory
```

GR8 CRM

gr8crm.github.io

40+ Grails plugins for rapid development of customer relationship management applications

- crm-contact & crm-contact-ui
- crm-content & crm-content-ui
- crm-task & crm-task-ui
- crm-campaign & crm-campaign-ui
- crm-sales & crm-sales-ui
- crm-product & crm-product-ui
- crm-blog & crm-blog-ui

All GR8 CRM plugins are open source with the Apache 2.0 License

gr8crm.github.io
github.com/goeh
github.com/technipelago
grails.org/plugin/migrate2-grails3
projectreactor.io

@goeh
goran@technipelago.se
www.technipelago.se
linkedin.com/in/gehrsson