Building APIs with Grails

SpringOne2GX
September 12th, 2013
Santa Clara, CA

Bobby Warner

Who Am I

- Blog
  - http://bobbywarner.com
- Social Networks
  - @bobbywarner
  - https://www.facebook.com/bobbywarner
  - http://www.linkedin.com/in/bobbywarner
  - http://bobbywarner.com/googleplus

Where I Used To Work

My New Company

Groovy Users of Minnesota (GUM)

Question #1

- What is a modern API?

Purpose

- JavaScript clients (Backbone.js, Angular.js, Ember.js, etc.)
- Mobile clients (Android, iOS, etc.)
- Service Oriented Architecture
- Sharing data with other companies

Architecture

- Client/Server
- Stateless
- Caching
- Uniform Interface
- Layered System

Principles

- Endpoints
- Resources (hierarchy, namespaces, representation)
- Methods (GET, POST, PUT, DELETE)
- Representations (JSON, XML)

Status Codes: Success

- Fetching
  - 200: ok
  - 204: no content
  - 206: partial content
- Changing
  - 201: created
  - 202: accepted (later processing)
  - 204: no content (PUT or DELETE succeeded)

- GET returns 200, 204, 206
- HEAD returns 204
- POST returns 201, 202
- PUT returns 201, 202, 204
- DELETE returns 202, 204

Status Codes: Failures

- Client Failures:
  - 4xx
- Server Failures:
  - 5xx

Versioning

- URI component (http://myapi.com/v1/books)
- Query parameter (http://myapi.com/books?version=v1)
- Header (http://myapi.com/books + HTTP header)

System to System: Authentication

- Basic
- Digest
- API Keys
- Request Signing

User to System: OAuth Resource Server

- Gateway products (i.e. Apigee, Layer 7)
- OAuth provider libraries (i.e. Spring Security OAuth)

Question #2

- What are the different ways we can build APIs?

Groovy API Options

- JAX-RS (Jersey, RESTEasy, Apache CXF, Restlet, etc.)
  - Grails JAX-RS Plugin
- Spring (RestTemplate)
- Dropwizard (uses Jersey)
- Groovy directly w/o frameworks (JsonBuilder)
- ...and of course Grails!

Question #3

- Why should we use Grails to build APIs?

Grails 2.2 and below

- http://www.bobbywarner.com/2013/04/18/building-apis-with-grails/
  - ObjectMarshaller
  - GSON Plugin

Grails 2.3

- Grails has always supported building APIs, but now it's a lot better
- There have been many different plugins filling in the gaps
- Significant REST enhancements included in Grails 2.3

Benefits of Using Grails 2.3

- Support for resource mappings, nested resources and versioning
- Extensible response rendering and binding APIs
- Scaffolding for REST controllers
- Support for HAL, Atom and Hypermedia (HATEAOS)

Question #4

- How do I build an API with Grails 2.3?

Grails 2.3 APIs

- Option #1: The resource annotation on domains
- Option #2: Extend RestfulController and override methods
- Option #3: Start from scratch

Option #1: Resource Annotation

- grails create-domain-class book
- @Resource(uri='/books', formats=['json', 'xml'])

Option #1: Resource Annotation

- Demo!

Option #2: Extend RestfulController

```groovy
package apidemo

import grails.rest.*

class BookController extends RestfulController {
    static responseFormats = ['json', 'xml']

    BookController() {
        super(Book)
    }
}
```

Option #2: Extend RestfulController

- Demo!

Option #3: Start From Scratch

```groovy
package apidemo

import grails.transaction.*
import static org.springframework.http.HttpStatus.*
import static org.springframework.http.HttpMethod.*

@Transactional(readOnly = true)
class BookController {
    static responseFormats = ['json', 'xml']

    def index(Integer max) {
        params.max = Math.min(max ?: 10, 100)
        respond Book.list(params)
    }

    def show(Book book) {
        respond book
    }
}
```

Option #3: Start From Scratch

- The key to implementing REST actions is the respond method introduced in Grails 2.3.
- Respond method tries to produce the most appropriate response for the requested content type (JSON, XML, HTML etc.)

Option #3: Start From Scratch

- Demo!

URL Mappings

- grails url-mapping-report

Question #5

- How do I customize the XML or JSON response?

Customize the Response

- Option #1: Default Renderers (Include or Exclude specific properties)
- Option #2: Custom Renderers
- Option #3: Converters (Object Marshallers)
- Option #4: GSPs

Option #1: Default Renderers

```
import grails.rest.render.json.*

beans = {
    bookRenderer(JsonRenderer, Book) {
        excludes = ['class']
    }
}
```

Option #2: Custom Renderers

```groovy
import grails.rest.render.*
import org.codehaus.groovy.grails.web.mime.MimeType

class BookXmlRenderer extends AbstractRenderer<Book> {
    BookXmlRenderer() {
        super(Book, [MimeType.XML, MimeType.TEXT_XML] as MimeType[])
    }

    void render(Book object, RenderContext context) {
        context.contentType = MimeType.XML.name

        def xml = new groovy.xml.MarkupBuilder(context.writer)
        xml.book(isbn: object.isbn, title: object.title)
    }
}
```

Option #3: Converters

```
import grails.converters.JSON

class BookMarshaller {
    void register() {
        JSON.registerObjectMarshaller(Book) { Book book ->
            return [
                name: book.title,
                isbn: book.isbn
            ]
        }
    }
}
```

Option #4: GSPs

show.xml.gsp

```
<%@page contentType="application/xml"%>
<book isbn="${book.isbn}" title="${book.title}"/>
```

Question #6

- How do I version a Grails API?

Versioning

- Option #1: URI

```
"/v1/books"(resources:"book", namespace:'v1')
"/v2/books"(resources:"book", namespace:'v2')
```

- Option #2: Accept Header ("Accept-Version: 1.0")

```
"/books"(version:'1.0', resources:"book", namespace:'v1')
"/books"(version:'2.0', resources:"book", namespace:'v2')
```

Versioning

```
curl -i -H "Accept-Version: 1.0" -X GET http://localhost:8080/books
curl -i -H "Accept-Version: 2.0" -X GET http://localhost:8080/books
curl -i -H "Accept: application/json" http://localhost:8080/v1/books
curl -i -H "Accept: application/json" http://localhost:8080/v2/books
```

Hypermedia

- Hypermedia as the Engine of Application State (HATEOS)
- Independent evolution
- Decoupled Implementation
- Pick a hypermedia-aware data type (raw XML and JSON aren't)

Hypermedia in Grails

- HAL Support (standard exchange format)
- Versioning via MIME Types
- HAL Renderers (HalJsonRenderer or HalXmlRenderer)

Thoughts on Hypermedia APIs

- Theoretically, you will be able to change your URLs without needing to update clients, but this isn't practical
- Not every API client will come through the front door
- Most developers would rather look at docs as opposed to making many GET requests to understand the API

Thoughts on Hypermedia APIs

- Very interesting in concept, but still not convinced it's practical
- Don't worry so much about HATEOS and instead focus on these three things:
  - Don't change your API URLs
  - Document your API
  - Provide a custom client wrapper

What's Next

- Testing, finding / fixing bugs
- Documentation updates (need more examples)
- New screen-cast in the works demoing more 2.3 REST features!

Closing

- Thank you for attending this presentation!
- Thank you SpringOne2GX organizers for hosting a great conference!

- Hopefully you use Grails 2.3 to create some APIs!

- Additional Resources
  - http://grails.org/doc/latest/guide/single.html#REST

Q & A

- Are there any questions?