# Controllers 2

# Section goals

After this section, you will be able to:

- Use the two techniques for binding data in controller actions
  - Named parameters and command objects
- Render JSON and XML using built-in Grails converters
- Execute different controller code based on the requested content type
  - HTML, JSON, XML

# Data binding - named parameters

Incoming parameters parsed and bound to action method parameters of same name

- Ex. Fields on a form submission

Good for cases when only a few parameters

- Can grow unwieldy if many parameters

# Named parameter example

## GSP form

```
<g:form controller="question" action="comment" id="${question.id}">
  <g:textArea name="comment" rows="15" cols="40" class="span8"/>
  <g:submitButton name="submit" value="Submit"/>
</g:form>
```

## Controller action

```
def comment(String comment) {
  Question question = Question.get(params.id)
  if (question){
    Comment newComment = new Comment(comment: comment, user: getUser())
    question.addToComments(newComment)
    question.save()
    redirect( action: 'show', id: question?.id)
  }
}
```

# Data binding - command objects

Command object is instance of helper class that handles binding and parsing of data

- Helper class is usually defined in controller class file

Parses dates, booleans, decimal numbers, etc.

Even retrieves object instances from the database when the param is an object id

# Command object example

```
<g:form name="commandForm" action="myAction">
  <g:hiddenField name="myObject.id" value="${object.id}"/>
  <g:textField name="username"/>
  <g:submitButton name="submitBtn" value="Submit" />
</g:form>

class MyController {
  def myAction(MyCommand cmd) {
    cmd.myObject?.username = cmd.username
  }
}

class MyCommand {
  MyObject myObject
  String username
}
```

# Command object - validation

Command objects support validation just like domain objects

```
class MyCommand {
  MyObject myObject
  String name

  static constraints = {
    myObject (nullable: false)
    name (nullable: false, blank: false)
  }
}
```

# Command objects - useful DTOs

A command object can also be a useful Data Transfer Object (DTO) for passing data

Ex: passing search data to service

Simpler than passing many search parameters to service

# Command object - DTO example

```
class SearchCommand {
  String comment
  String name
  String number
  String user
}

def search(String comment, String name, String number, String user) {
  def results = searchService.doSearch(comment, name, number, user)
  [results: results]
}

def search = { SearchCommand cmd ->
  def results = searchService.doSearch(cmd)
  [results: results]
}
```

# Workshop - command objects

Check out the `advancedControllerStart` branch from git.

Convert the controller action `AnswerController.answer()` to use a command object

# Workshop Solution - 1

```
def answer(AnswerCommand cmd) {
    if (cmd.validate()) {
        def question = cmd.id
        User user = User.get(1)

        def answer = new Answer(text: params.answer, author: user)
        question.addToAnswers(answer)
        try {
            answer.save(failOnError: true)
            question.save(failOnError: true)
        } catch (ValidationException ex) {
            flash.message = "There was an issue adding your answer.
Please try again"
        }
        redirect controller: "question", action: "show", id: question.id
    }
```

# Workshop Solution - 2

```
else {
    if (cmd.id) {
        redirect(controller: 'question', action: 'show', id: cmd.id)
    } else {
        redirect(controller: 'question', action: 'list')
    }
}
```

# Workshop Solution - 3

```
package com.opi

import grails.validation.Validateable

@Validateable
class AnswerCommand {
    Question id
    String answer

    static constraints = {
        id nullable: false, blank: false
        answer nullable: false, blank: false
    }
}
```

# Controller converters

Easily render objects in different formats

Two converters included with Grails

- grails.converters.JSON
- grails.converters.XML
- Converters in the grails.converters.deep package are deprecated - don't use them

# Converter examples

```
import grails.converters.XML
import grails.converters.JSON

class MyController {
  def myActionJson() {
    MyObject myObject = MyObject.get(params.id)
    render myObject as JSON
  }

  def myActionXml() {
    MyObject myObject = MyObject.get(params.id)
    render myObject as XML
  }
}
```

# Converters - fine-grained output control

Can set format-specific output fields

Put code in BootStrap.groovy

Can exclude object fields, create new fields

```
JSON.registerObjectMarshaller(MyObject) {
  def jsonMap = [:]
  jsonMap['id'] = it.id
  jsonMap['name'] = it.name
  jsonMap['simpleName'] = it.toString()
  return jsonMap
}
```

# Converters - control built-in object output

Can also control the output format of built-in object types, such as Date

```
JSON.registerObjectMarshaller(Date) {
  def dateFormat = new SimpleDateFormat("mm/DD/yyyy")
  dateFormat.setTimeZone(TimeZone.getTimeZone("UTC"))
  return dateFormat.format(it)
}
```

# Converter unit testing

Can access the rendered JSON or XML on the response object in a controller unit test

```
@TestFor(MyController)
class MyControllerTests {
  @Test
  void jsonRenderTest() {
    controller.myActionJson()

    assert response.json['name'] == "ExpectedObjectName"
  }
}
```

# Respond vs. withFormat

Respond attempts to return the most appropriate type for the requested content type.

withFormat lets you decide what to render based on the requested content type.

# Respond vs. withFormat

```
def save(Answer answerInstance) {
    if (answerInstance.hasErrors()) {
        respond answerInstance.errors, view: 'create'
        return
    }
    answerInstance.save flush: true

    request.withFormat {
        form multipartForm {
            ...
            redirect answerInstance
        }
        '*' { respond answerInstance, [status: CREATED] }
    }
}
```

# Code based on the format client accepts

Uses the HTTP 'Accept' header or file extension

- `headers['Accept'] = 'application/json'`
- `/question/show/1.json`

Determines format based on 'grails.mime.types' map in Config.groovy

- Default format map generated by 'grails create-app'

Can use same controller action for HTML form submissions, JSON API, and XML API

# Controller interceptors

Can run code before or after each action

Two types of interceptors

- `beforeInterceptor, afterInterceptor`

Defined as static fields on controller class

```
class MyController {
  static beforeInterceptor = ...

  static afterInterceptor = ...
}
```

# Controller beforeInterceptor

Can halt execution before action is executed

- Return `false` to halt execution
- Often used to redirect to different controller/action

```
class MyController {
  static beforeInterceptor = {
    if (!session.user) {
      redirect(controller: "login", action: "login")
      return false
    }
  }
}
```

# Controller afterInterceptor

Can add to model returned to view

```
class MyController {
  static afterInterceptor = { model ->
    model.extraValue = "extra value"
  }

  def myAction() {
    ['value': "first value"]
  }
}
```

# Interceptors - restricting scope

Can use a method as the interceptor

- Executes the method ‘theInterceptor’ only after the 'show' action

```
class MyController {
  static afterInterceptor = [action: this.&theInterceptor, only: ['show']]

  private theInterceptor(model) {
  }

  def show() { // Interceptor executed after this method
  }

  def list() { // Interceptor not executed after this method
  }
}
```

# Flash scope

Data in flash scope lives for the lifecycle of the request

- Ex: passing message from controller to view

Accessed as map in controller and view

Controller:

```
def myAction() {
  flash.message = "Message to user"
}
```

GSP:

```
<div>${flash.message}</div>
```

# Session scope

- Data stored in the HTTP session
- Lives across multiple requests
- Can access as map from controllers

```
def myAction(String incomingValue) {
  session["value"] = incomingValue

  def bar = session["bar"}

  ["bar": bar]
}
```

# Workshop - flash message

- Add a flash message to the `AnswerController.answer()` action saying whether the comment was added successfully or not

- Add a conditional display of flash message to the `/views/question/show.gsp`

# Workshop - flash message

```
if (cmd.validate()){
  ...
  flash.message = "Successfully added comment"
} else {
  flash.error = "Cannot save comment"
}
```

# Controller scopes

**Singleton (default since Grails 2.3)**

One instance per application context

Set it by adding to controller:

```
static scope = 'singleton'
```

(or remove the the line altogether)

# Controller scopes

**Prototype (default in Grails 2.2 and below)**

One instance per request

Set it by adding to controller:

```
static scope = 'prototype'
```

# Controller scopes

## Session

One instance per session

Set it by adding to controller:

```
static scope = 'session'
```

# Default controller scope

Can set the default scope globally via

`Config.groovy`

```
grails.controllers.defaultScope = "session"
```

# Specify action via HTTP method - parseRequest

- Call controller action based on HTTP method
- Controller action intuitive based on intent

```
UrlMappings.groovy
"/$controller"(parseRequest:true) {
  action = [GET: "list", POST: "save"]
}
```

GET: /question -> /question/list

POST: /question -> /question/save