MVC With and Without a Framework

Nolan Erck

About Me

- Owner / Director, South of Shasta
- ColdFusion Certification Team
- Manager, Sac Interactive Tech Meetup
- Reformed Video Game Developer - Grim Fandango, SimPark, StarWars Rogue Squadron, etc...
- Music Junkie

I'M REAL PROUD OF MY RECORD COLLECTION

Code and Slides

- github.com/nolanerck
    - mvc-with-and-without-a-framework

Today's Agenda

- Some prerequisites
- Intro to Model-View-Controller pattern
- Some pros and cons
- All concepts are non-framework, non-*language* specific
- Some code samples

Before we get started...

- Everyone should know...
- CFComponent
  - How to create and use via new()
  - Some understanding of how a component works
- The code samples aren't “perfect OO”
  - Some shortcuts to make the concepts easier to learn.
- One more thing you need to know...

OO Programming is Hard

- Very different from a top-down procedural code
- Several new concepts and design considerations
- Some of it is very confusing at first
- That's NORMAL

...but this is the way development has moved on pretty much every platform

CFComponent

- ColdFusion's OO construct
- Same as “class” or “object” in Java, C++, etc
- Several calling conventions – cfinvoke, cfobject, CreateObject, new
  - we'll use new()
- *Typically* includes an “init” method for setup but that's not technically required
  - aka “constructor”

Musician.cfc

```
<cfcomponent>
	<cfset variables.name = “” />
	<cfset variables.instrument = “” />

	<cffunction name=”playInstrument”>
		<!--- code goes here --->
	</cffunction>
</cfcomponent>
```

Musician.cfc

- You use it like so:

```
<cfset mySinger = new (“Musician” ) />
<cfset mySinger.name = “John Lennon” />
<cfset mySinger.playInstrument() />
```

Model View Controller

- Not ColdFusion specific
- Common design pattern used in other OO languages
  - What's a design pattern?
    - $6 word for “a common problem solved by organizing objects in a certain way”.
    - Like for() loops and arrays but with objects and methods.

Model View Controller

- Used within CF frameworks (Framework-1, ColdBox, Model-Glue, Mach-ii, etc)
- Lots of this info/concepts transfers to other OO languages
- Basically a way of organizing and calling your code that “separates the concerns”
  - Display code, controlling flow, data/business logic.

View

- The part of the app that users, well, view
- HTML, CSS, JavaScript
- *Some* CF for display logic but that's it.
  - Toggle the “log in / log out” button, etc
  - Alternating page row colors in a table (but really, do that in CSS!)
- No business logic, no SQL code
- The menu at a restaurant

Model

- Short for “data model” (kind of)
- Where all (yes all) your SQL code lives
- Doesn't have to be a database
    - Whatever your storage medium is
        - Log files, XML, etc
- Business logic mostly lives here too
- The kitchen / chef at a restaurant

Controller

- Sits between the Model and the View
- No HTML output, no SQL
- Small bits of "logic" for *controlling* the flow of your application
  - User clicks "save" button, save action happens, user is then directed to the next page in the app.
- Like the waiter in a restaurant

Model View Controller

- View

  HTML

  JavaScript

  CSS

  UI-related logic, but no business logic

  No SQL!

- Controller

  Glues the View and Model together
- Logic for controlling flow of the app
- “Controls” where the user goes next

- Model

  SQL

  Business logic

  LDAP

  XML I/O

  etc

Demo – Old Spaghetti App

- Nothing is modular (CFInclude doesn't count, it leaks data)
- SQL, HTML and business logic are all mixed together
- No ability to build an API
- Lots of risk when making upgrades

Demo - Model

- Nothing new here
- Same CFQuery stuff we've used for years, just inside CFFunction tags
- “But that's a lot of typing”
  - One-time “pain” for developer is less important than better overall architecture
  - Various IDE tools, plugins, code generators, etc that will help.

Demo - View

- Nothing new here either
- Mostly plain HTML, JavaScript, CSS
- A *small* amount of CF for display logic
- No SQL, no real business logic
- Easy to swap out new UI, add Bootstrap, make the site responsive, etc
- No SQL or business logic to accidentally break

Demo - Controller

- Takes the info from the user
- Does some *minor* validation
- Hands the data off to the Model for all the heavy lifting
- *Controls* where the user goes next in the application
- No SQL or HTML, *very little* business logic

Demo - Controller

- Other benefits
  - access=”remote”
  - Methods can be called via HTTP
  - Automatically available for ajax, for mobile apps, as an API
    - (other design considerations for security, proper modularity, etc)

MVC - Pros

- Promotes code reuse
- Allows multiple people to work on code at the same time
  - 1 works on UI (View), 1 works on app flow, 1 on SQL queries, etc
- Non-framework, non-language specific
- Very common pattern/nomenclature
  - "Model" means the same thing in Java, .NET, Ruby, C++ and so on

MVC - Cons

- A change in style from spaghetti code, or even good procedural programming
  - May take time to “click”
- “More typing” to get the 3 layers up and running
  - But the code is more reusable
  - And seriously, this is lame excuse
    - IDE, plugins, various tools to help write code for you

When Is This Enough?

- Should I use a framework? When is this MVC pattern enough by itself?
- Open source projects
  - Personal preference
  - Works on old flavors of CF
- When I just need some organization
  - But won't use the “extras” from ColdBox, Framework-1, etc.

Using a Framework

- MVC frameworks all kind of work the same way
- ...because they're all using the same *design pattern!*
- Framework-1, ColdBox, all have places to put “views”, “controllers”, and “models”
- Only difference is a little syntax and calling convention stuff. Nothing crazy.
- Let's look at a Framework-1 app

A few last thoughts

- OO is hard!
- That's normal
- *Nobody* instantly knows this stuff the first time
- But it does make building large apps simpler
  - Keeps you organized
  - Common terminology
  - Separation of concerns, easy to update UI, update API, split up the work

Other Resources

- Book: Object Oriented Programming in ColdFusion – Matt Gifford
- Book: Head First Design Patterns
- CFML Slack
- southofshasta.com
- Ortus Solutions

Questions?

Email: nolan@southofshasta.com

Twitter: @southofshasta

Blog: southofshasta.com/blog/

(Slides are on GitHub.)

Thanks!