

Model View Controllers

Nolan Erck
Saccfug, November 2008

What We'll Cover Today

- An introduction to the “Model View Controller” pattern.
- Some pros and cons.
- All info is non-framework specific.

About Me

- Independent contractor (southofshasta.com).
- Saccfug Co-Manager.
- CF Development since 4.5.
- Software development for 12 years.

Model View Controller

- Not ColdFusion specific.
- Common design pattern used in other languages.
- Used within CF Frameworks (e.g. Model Glue).

Definition – Model

- Short for “database model”.
- Where all (yes all) of your SQL code lives.
- DAOs, Gateways, etc, are part of the model.
- Doesn't have to be a database.
 - Whatever your storage medium is for data.
 - Log files, XML, etc.
- Some business logic may live here too.

Definition -- View

- The part of the app that users, well, view.
- All HTML/Javascript/CSS for the visible output.
- Some CF code for display logic, but that's it.
 - e.g. Code to toggle background colors of rows in a table.
- No business logic.
- No SQL code.
- Similar to the “dsp_” files from Fusebox apps.

Definition -- Controller

- Sits between the Model and the View.
- Some business logic lives here.
- No HTML output is done here.
- Just calls to `<cflocation>`, “controlling” where the user goes next.

Model View Controller

- Model
 - DAO
 - Gateway
 - All SQL.
 - Business logic.
- Controller
 - Glues the View and Model together.
 - Business logic
 - “Controls” where the user goes next.
- View
 - HTML
 - JavaScript
 - CSS
 - UI related CF, but no business logic.
 - No SQL!

customer_old.cfm

- Let's look at the customer_old.cfm sample.
 - Original CFM template.
 - Looks a lot like code we've all seen (and written) at some point.
 - Everything is together.
 - No reuse, no modularity.
 - Only 1 programmer can work on it at a time.
 - No ability to swap out anything.
 - What if we want to experiment with a Flex UI?
Everything has to be rewritten.

customer.cfm

- Now let's look at customer.cfm
 - A refactored version.
 - This is the “view”.
 - Note the `<form post="">` posts to a “Controller” CFC.
 - Also note the `<input name="method">` field.
 - Name of the method in our controller we're calling.
 - Now, only the HTML is in this file.
 - Easy to swap in a different UI without touching the business logic, or SQL code.
 - What if we want to try a Flex UI?
 - Easy! Since this file only contains UI code and the business logic is all reusable (and callable from Flex!)

CustomerController.cfc

- Not much going on here.
- `saveCustomer()` method.
 - Takes the fields from the form post.
 - Passes them to the Model code for saving.
 - Then “controls” where the users goes next (back to the Customer.cfm page, with an “saved successfully” message.
 - (Could also check for errors and direct User to an “errors” page if need be.)

Our “Model” Code.

- Nothing new or different here.
- CustomerBean.cfc
 - Bean is the same as a struct.
 - Just “wraps” the data.
 - No moving parts.
- CustomerDAO.cfc
 - Handles saving our Customer Bean to the database.
- CustomerGateway.cfc
 - Handles some other SQL functionality for us (things meant for multiple records, whereas the DAO is for just 1 record).

Our “Model” Code (cont)

- Illudium PU36 Code Generator is your friend.
- Note: 99% of the “Model” code in this app was auto-generated!
 - I made a few hand edits, just for simplicity of the demo.
 - Even that could have been automated if necessary (just modify the template in PU36).
- Barely any typing was needed for the “Model” code.

Pros

- Promotes code reuse.
- Allows multiple people to work on the same “screen” in an app at the same time.
 - One works on UI (the View).
 - One works on business logic (the Controller).
 - Etc.
- Non framework/language specific.
- Very common pattern/nomenclature.
 - E.g. “model” means the same thing regardless of language or framework, etc.

Cons

- A change in style from other programming methods.
 - May take a little time to “click”.
- “More typing” to get the 3 layers up and running.
 - But the code is more reusable.
 - And as we've seen, PU36 can cut down that typing to where it's really a non-issue in the long run.

Other Resources

- Book: “Head First Design Patterns”
- CFCDev Mailing list
- Fullasagoog.com
- Coldfusionbloggers.org
- Stannard.net.au/blog
 - Article: “Writing Your Own Simple MVC Framework in ColdFusion”.
- Benorama.com (use Archive.org to see the CF info).

Thanks!

- nolan.erck@gmail.com
- southofshasta.com/blog
- nolanmusic.com
(shameless plug – show on Nov 15th!)