Objects and the Web

陳建村

2002/06/17

ctchen@canthink.com.tw
Outline

- The problem and solution
- Definitions
- Motivations
- MVC concepts & MVC for the Web
- The Web Actions framework
- Conclusion
- Q&A
The problems on the Web today

- Scripting and server page technologies can encourage copy-and-paste reuse, direct-to-database coding, and poor factoring.

- Scripting languages, are often optimized for rapidly creating simple functionality rather than for modular construction of large programs.
The solution

- The authors describe a layered, OO architecture, based on the Model-View-Controller (MVC) patterns, which to produce large, well-structured Web systems.
Definitions
Definitions

- Layered architecture
- Input
- Application logic
- Business logic
- Presentation
- Scripts
- Server pages
Layered architecture

- A layered architecture is a system containing multiple, strongly separated layers, with minimal dependencies and interactions between the layers
  - Good separation of concerns
- The layers we are concerned here:
  - (1) Input
  - (2) Application logic
  - (3) Business logic
  - (4) Presentation
Input

- The input layer contains the code concerned with processing and syntactically validating input
- In the MVC framework, this corresponds to the input controller
Application logic

- The application logic code is concerned with the application’s overall flow
  - As the glue layer, separating business logic from input and presentation logic and managing the interface between the two
  - Manage a multipage Web interaction as a sequence of steps (maintain state)
- In the MVC framework, this corresponds to the application controller
Business logic

- The business logic code is concerned only with the underlying functionality
  - Entirely unaware of the presentation layer
  - Could be the largest component and include code that accesses external systems such as database, external components, and other services
- In the MVC framework, this corresponds to the model
Presentation

- This layer contains code and noncode (such as HTML text and images) resource used to present the application
  - Typically contains little code - code concerned only with formatting and presenting data
- In the MVC framework, this corresponds to the view
Scripts

- **Scripts**- small programs that perform HTTP processing
  - Compiled CGI, Perl, Python, Ruby, VBScript, Servlet
  - Accept an HTTP request and send back a response
  - In basic usage, each script is stateless and independent of all others
- **An important distinction**- whether scripts can share memory with other scripts or are entirely independent
Server pages

- Servers pages are HTML pages annotated with small amounts of code
  - An alternative mode of HTML scripting
  - JSP, ASP, PHP, Zope

- Variations:
  - Annotated with application-specific HTML or XML tags
  - Developers can specify code to be run when these tags are encountered
Motivations (1/2)

- From an OO and layering perspective, the problem is:
  - A single script has responsibilities spanning several layers
- The script must:
  - Accept input
  - Handle application logic
  - Handle business logic
  - Generate output (presentation logic)
Motivations (2/2)

- Couples all the layers together, making it harder to modify or test any particular aspect in isolation.
- Mixing code with the HTML also presents code management and debugging issues.
Accepting input

- **Passing parameters:**
  - URL encoding, Query values, Form data, cookie

- **Script must know the parameter-passing mechanism:**
  - convert the parameters to appropriate type
  - Validate

→ code duplication between scripts
Handling application logic (1/2)

- Another issue affects both input and application logic—lack of information hiding when accessing request and session data:
  - Retrieve data by name (dictionary-like fashion)
    - Must be aware of the naming conventions
  - Programming mechanisms for controlling variable access do not apply
  - Potential source of errors:
    - spelling error
    - Different scripts using the same name for different purposes
Handling application logic (2/2)

- When using server pages for application logic:
  - Code management techniques are not usually available for code inside server pages
  - Debugging code inside the server pages is difficult

→ To minimize the amount of code in server pages and to keep application logic out of the pages
Handling business logic

- Difficult to isolate the business logic from the other layers, particularly application logic
- The same issues for application logic
Generating output

- Simple scripts mix the HTML encoding of the result with the dynamic data
  - Couples the page’s look and feel with the other layers
  - Changing the Web site’s look or adapting the application to multiple output devices becomes difficult

→ web designers design and maintain the pages and by letting programmers provide annotations
Model-View-Controller
MVC concepts (1/2)

- **Model** - handles application and business logic
- **View** - handles presentation logic
- **Controller** - accepts and interprets keyboard and mouse input
MVC concepts (2/2)

- Intention:
  - To separate the model (nonGUI) code from its presentation

- Two weaknesses:
  - It had a simplistic view of the model and did not account for any difference between application logic and business logic
  - Most GUI libraries and windowing systems combined the view and controller functions in a single widget, making the logical separation into view and controller less useful
The revised MVC
MVC for the Web

- Model - business object
- View - View
- Controller:
  - Input controller
  - Application controller
- Authors define a framework called Web Actions
The Web Actions framework

- WAP
- HTTP
- object
- server page
- script
- command
- business objects

Diagram showing the flow of data and control within the framework.
Input controller (1/2)

- Implement the input controller as a script
  1. Parses input
  2. Determines the parameter-passing mechanisms
  3. Extracts any necessary information from the request
  4. Cooperates with the application controller to determine the next action
  5. Invokes that action in the correct context
Input controller (2/2)

There is a single input controller for all pages in a Web application (?)

- Localize any knowledge of HTTP or naming conventions at the request level
- Reduce code duplication and the total number of scripts
- Make it easier to modify any of the input processing
  - Single point of modification
**Application controller (1/2)**

- Implement the application controller as a regular object
  - Application flow
  - Handles errors
  - Maintains longer-term state (including references to the business objects)
  - Determines which view to display

- Storing it in the session using a key known to the input controller
By using a single, well-encapsulated object as the central point of reference of any persistent information, it resolves the issues of:

- Information hiding
- Naming conventions
- Rather than storing isolated pieces of information in the session, we store them in business object and access them using messages from application controller
Action

- Input controller will invoke one of many possible actions on each request
  - Determine which one by the input from the user and on the application’s current state
- The result of this determination as an Action object
  - An implementation of Command pattern
Business objects

- Implement the business objects as normal objects that contain only business logic
  - Have no knowledge of any other layers
  - The application controller is the only thing that manipulates the business objects
View

- Implement views as server pages, which can access the application controller and business objects
  - Should contain as little code as possible, delegation most functionality to the application controller or business object
  - Only code directly related to presentation in the current page should be used in a page
- Prefer to use a tag mechanism such as JSP custom tags to remove code from the pages altogether
public void service (HttpServletRequest req,
    HttpServletResponse res)
    throws ServletException, IOException {

    ApplicationController controller = this.appControllerFor (req);
    this.setValueIn (controller, req);
    String actionID = req.getPathInfo ();
    Action action = controller.actionFor (actionID);
    controller.performAction (action);
    View theView = this.viewFor (req);
    theView.forwardPage (controller.nextPage ());
}
http request

1. appCtl = appControllerFor

2. setValuesIn (appCtl, req)

3-1. actionID = getPathInfo (req)
3-2. action = actionFor (actionID)

4. performAction (action)

5-1. viewFor (req)
5-2. forwardPage (appCtl.NextPage())

run ()
http request

1. appCtl = appControllerFor

2. setValuesIn (appCtl, req)

3-1. actionID = getPathInfo (req)
3-2. action = actionFor (actionID)

4. performAction (action)

5-1. view = getView()
5-2. forwardPage ()

run ()
(1) - Find the application controller

- Input controller determine which application controller is responsible for the current request
  - Active application controller are stored in the session
  - Assume determine the application controller using a lookup based on the request’s path
(2) - Accept input (1/2)

- The input controller extracts the appropriate information from the request and transfers that information to the application controller.
  - Find any input parameters
  - Cooperation with the application controller, performs basic syntactic validation and informs the application controller of the values.
(2) - Accept input (2/2)

https://objectbank.com/ InputController/transfer?
form=123&to=321&amt=$50.00

action

https://objectbank.com/ InputController/updateAccountData

query parameters

submitted from an HTML form
(3) - Find the action

- The application controller keep track of a set of acceptable sequences of operations and the previous steps the user has taken.
  - We can determine which action to take and whether this action is legal in the current context.
  - This logic is particularly important because Web users can use the Back button to throw off the sequence of operations in a way that is not possible in a GUI.
(4) - Perform the action (1/2)

- We have determined which action to take, so execute it now
- Which implementation?
  - Heavyweight actions implemented as objects inheriting from a class Action and implementing a trigger method (command pattern, support undo*)
  - As an indicator of the application controller method to be performed (reflection) or function pointer (delegate)*
Public void trigger (ApplicationController aAppCcontroller) {
    BankController appCtrl = (BankController) aAppCcontroller;

    Account acc = appCtrl.readAccount (context.getAccountNo ());

    appCtrl.setCurrentAccount (acc);

    appCtrl.setNextPage ("account Summary");
}
(4) - Forward the request

- Once the action is complete, we determine the URL of the next page to be displayed and forward the request
  - In a simple system, the actions can directly determine the next page
  - In a more complex system, the application controller might coordinate this using internal state management such as state machine
Conclusion

- Using this framework, developer primarily focus on writing application code rather than dealing with servlets, requests, or session variables

- Problems:
  - Poor examples
  - Single input controller? How to implement?
  - How about client side script?
  - Cookie?
  - View with option?