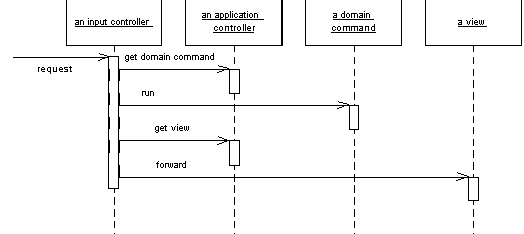
**http://www.martinfowler.com/eaaCatalog/applicationController.html**

**Application Controller**

*A centralized point for handling screen navigation and the flow of an application.*

For a full description see [P of EAA](http://www.martinfowler.com/books/eaa.html) page **379**



Some applications contain a significant amount of logic about the screens to use at different points, which may involve invoking certain screens at certain times in an application. This is the wizard style of interaction, where the user is led through a series of screens in a certain order. In other cases we may see screens that are only brought in under certain conditions, or choices between different screens that depend on earlier input.

To some degree the various Model View Controller (330) input controllers can make some of these decisions, but as an application gets more complex this can lead to duplicated code as several controllers for different screens need to know what to do in a certain situation.

You can remove this duplication by placing all the flow logic in an Applica-tion Controller. Input controllers then ask the Application Controller for the appropriate commands for execution against a model and the correct view to use depending on the application context.

http://www.corej2eepatterns.com/Patterns2ndEd/ApplicationController.htm

**Application Controller**

See [Core J2EE Patterns, 2nd Edition](http://www.corej2eepatterns.com/AboutTheBook.htm) for full description of this pattern and its strategies.

**Problem**

*You want to centralize and modularize action and view management.*

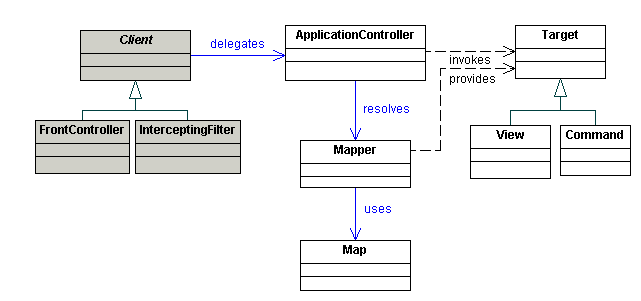
**Forces**

* You want to reuse action and view-management code.
* You want to improve request-handling extensibility, such as adding use case functionality to an application incrementally.
* You want to improve code modularity and maintainability, making it easier to extend the application and easier to test discrete parts of your request-handling code independent of a web container.

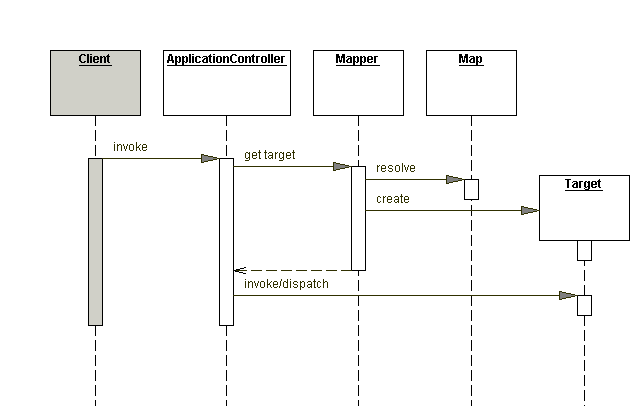
**Solution**

**Use an Application Controller to centralize retrieval and invocation of request-processing components, such as commands and views.**

Class Diagram



Sequence Diagram



**Strategies**

* Command Handler Strategy
* View Handler Strategy
* Transform Handler Strategy
* Navigation and Flow Control Strategy
* Message Handling Strategies
* Custom SOAP Message Handling Strategy
* JAX RPC Message Handling Strategy

**Consequences**

* Improves modularity
* Improves reusability
* Improves extensibility

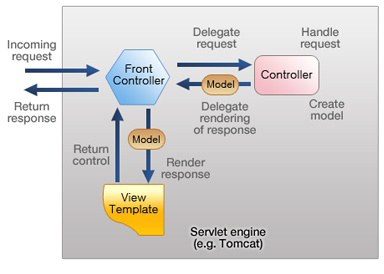
**Related Patterns**

* [Front Controller](http://www.corej2eepatterns.com/Patterns2ndEd/FrontController.htm)   
  A Front Controller uses an Application Controller to perform action and view management.
* [Service Locator](http://www.corej2eepatterns.com/Patterns2ndEd/ServiceLocator.htm)   
  A Service Locator performs service location and retrieval. A Service Locator is a coarser object, often uses sophisticated infrastructure for lookup, and doesn’t manage routing. It also doesn’t address view management.
* Command Processor [[POSA1](http://www.amazon.com/exec/obidos/ASIN/0471606952)]  
  A Command Processor manages command invocations, providing invocation scheduling, logging, and undo/redo functionality.
* Command Pattern [[GoF](http://www.amazon.com/exec/obidos/ASIN/0201633612/corej2eepatte-20" \t "_blank)]  
  A Command encapsulates a request in an object, separating the request from its invocation.
* Composite Pattern [[GoF](http://www.amazon.com/exec/obidos/ASIN/0201633612/corej2eepatte-20" \t "_blank)]  
  A Composite represents objects as part-whole hierarchies, treating individual objects and compositions of objects uniformly.
* Application Controller [[PEAA](http://www.amazon.com/exec/obidos/ASIN/0321127420/corej2eepatte-20)]  
  Martin Fowler’s description of Application Controller [PEAA] seems to focus on controlling a user’s navigation through an application using a state machine, as described in the Navigation and Flow Control strategy. However, the Application Controller [PEAA] and our documentation of Application Controller have the same core intent.

<http://www.techzoo.org/java/lets-implement-j2ee-application-controller-design-pattern-today.html>

**Let’s Implement J2EE Application Controller Design pattern today**

* Tuesday, August 27, 2013, 19:18
* [Java](http://www.techzoo.org/category/java)
* 9,558 views

I was reading Santosh Sir’s article on application controller pattern last night. Those who have some understanding on J2EE design pattern know the different between Front Controller and Application controller pattern. Struts 1.x is a best example of both.

You can read the Santosh Sir’s article on [Front Controller](http://www.developersfusion.com/Articles/AD/F/26/Front-Controller-Design-Pattern.aspx) and [Application controller](http://www.developersfusion.com/Articles/AD/F/27/Application-Controller-Design-Pattern.aspx) pattern.

So Today I have decided to implement application controller design pattern which can use java’s properties file to read and configure controllers.

Start with Controller interface.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | package org.techzoo.aqsa;    import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    public interface Controller {        public View execute(HttpServletRequest request,          HttpServletResponse response) throws Exception;  } |

View.java is a wrapper view resolver return from Controller’s execute method.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | package org.techzoo.aqsa;    import java.util.Map;    public class View {        private Map model;      private String forward;        public View(String forward) {          this.setForward(forward);      }        public View(String forward, Map data) {          this.setForward(forward);          if (data != null) {              this.model = data;          }      }        public void clean() {          this.model = null;      }        public Map getModel() {          return model;      }        public void setModel(Map model) {          this.model = model;      }        public String getForward() {          return forward;      }        public void setForward(String forward) {          this.forward = forward;      }  } |

ControllerConfig is used to hold controller’s action with it’s class name.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | package org.techzoo.aqsa.config;    import static java.lang.Class.forName;  import org.techzoo.aqsa.Controller;    public class ControllerConfig {        private String action;      private String controllerClass;        public ControllerConfig(String action, String controllerClass) {          super();          this.action = action;          this.controllerClass = controllerClass;      }        public Controller invokeController() {          Controller controller = null;          if (controllerClass != null && !controllerClass.isEmpty()) {              try {                  controller = (Controller)forName(controllerClass).newInstance();              } catch (Exception e) {                  e.printStackTrace();              }          }          return controller;      }        public void setAction(String action) {          this.action = action;      }        public void setControllerClass(String controllerClass) {          this.controllerClass = controllerClass;      }        public String getAction() {          return action;      }        public String getControllerClass() {          return controllerClass;      }  } |

WebConfig is used to manage all controller configured in properties file. This class will read, and make available all controller Object to Main Servlet.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | package org.techzoo.aqsa.config;    import static java.lang.String.format;    import java.io.File;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  import java.util.HashMap;  import java.util.Iterator;  import java.util.Map;  import java.util.Properties;  import java.util.Set;    public final class WebConfig {        private static final String MAPPING = "mappings.properties";      private Map<String, ControllerConfig> controllers = null;        public WebConfig(String mappingFile) {          File mapping = new File(mappingFile);          System.out.println(mapping.getPath());          if (mapping.isFile() && mapping.exists()) {              String xmlFileName = mapping.getName();              if (!xmlFileName.equals(MAPPING))                  throw new UnsupportedOperationException(format(                          "mapping file name should be %s.", MAPPING));              controllers = new HashMap<String, ControllerConfig>();              readMappingFile(mapping);          } else {              System.err.println("Not a Mapping file or not exist.."                      + mapping.getPath());          }      }        private void readMappingFile(File propFile) {          Properties props = new Properties();          try {              props.load(new FileInputStream(propFile));              Set keys = props.keySet();              Iterator it = keys.iterator();              while(it.hasNext()){                  String key = (String)it.next();                  String className = (String)props.get(key);                  ControllerConfig cc = new ControllerConfig(key, className);                  addControllerConfig(key, cc);              }          } catch (FileNotFoundException e) {              e.printStackTrace();          } catch (IOException e) {              e.printStackTrace();          }      }        private void addControllerConfig(String actionUrl,              ControllerConfig controllerConf)      {          if (!controllers.containsKey(actionUrl)) {              controllers.put(actionUrl, controllerConf);          } else throw new UnsupportedOperationException(              format("Action %s is already exist. "                      + "Two controller can't map to one action.",                      actionUrl));      }        public Map<String, ControllerConfig> getControllers() {          return controllers;      }  } |

And Last is our Front Controller servlet (AqsaServlet). Aqsa Servlet is a Centralized control in application, a sigle entry point which manage Protocol Handling, Request Navigation and View Dispatching.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92 | package org.techzoo.aqsa;    import static java.lang.String.format;    import java.io.IOException;  import java.util.Map;  import java.util.Map.Entry;    import javax.servlet.RequestDispatcher;  import javax.servlet.ServletConfig;  import javax.servlet.ServletContext;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    import org.techzoo.aqsa.config.ControllerConfig;  import org.techzoo.aqsa.config.WebConfig;    final public class AqsaServlet extends HttpServlet {        private static final long serialVersionUID = 1L;      private static final String MAPPING\_FILE = "mapping";        private ServletContext context;      private WebConfig webConfig = null;        public void init(ServletConfig config) throws ServletException {          super.init(config);          this.context = config.getServletContext();          String mappingFile = null;          String controllersProps = config.getInitParameter(MAPPING\_FILE);          mappingFile = context.getRealPath(controllersProps);          webConfig = new WebConfig(mappingFile);      }        protected void doGet(HttpServletRequest request,              HttpServletResponse response) throws ServletException, IOException {          doProcess(request, response);      }        protected void doPost(HttpServletRequest request,              HttpServletResponse response) throws ServletException, IOException {          doProcess(request, response);      }        private void doProcess(HttpServletRequest request,              HttpServletResponse response) {          final String servletPath = request.getServletPath();          final String actionPath = servletPath.substring(1,servletPath.lastIndexOf("."));          final Map controllers = webConfig.getControllers();          final ControllerConfig ctrConfig = controllers.get(actionPath);            View view = null;          if (ctrConfig != null) {              String actionUrl = ctrConfig.getAction();              if (actionPath.equals(actionUrl)) {                  try {                      final Controller controller = ctrConfig.invokeController();                      view = controller.execute(request, response);                      prepareModelData(request, view);                  } catch (Exception e) {                      System.out.println(e.getMessage());                  }                  dispatchRequestToView(view, webConfig, request, response);              }          } else              throw new UnsupportedOperationException(format(                      "action %s is not found in mapping file.", actionPath)                  );      }        private void dispatchRequestToView(View view, WebConfig webConfig,              HttpServletRequest request, HttpServletResponse response) {          try {              String forward = view.getForward();              final RequestDispatcher rd = context.getRequestDispatcher(forward);              rd.forward(request, response);          } catch (Exception e) {              e.printStackTrace();          }      }        private void prepareModelData(HttpServletRequest request, View view) {          final Map model = view.getModel();          if (model != null) {              for (Entry data : model.entrySet()) {                  request.setAttribute(data.getKey(), data.getValue());              }          }      }  } |

web.xml and controller config properties file will Look similar below…



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <?xml version="1.0" encoding="UTF-8"?>  <web-app id="WebApp\_ID" version="2.5">        <display-name>AqsaFramework</display-name>        <servlet>          <servlet-name>action</servlet-name>          <servlet-class>org.techzoo.aqsa.AqsaServlet</servlet-class>          <init-param>              <param-name>mapping</param-name>              <param-value>/WEB-INF/mappings.properties</param-value>          </init-param>          <load-on-startup>1</load-on-startup>      </servlet>        <servlet-mapping>          <servlet-name>action</servlet-name>          <url-pattern>\*.zoo</url-pattern>      </servlet-mapping>        <welcome-file-list>          <welcome-file>index.jsp</welcome-file>      </welcome-file-list>    </web-app> |

and ….

#Mapping file contain key value pair  
login=com.xpert.controller.LoginController  
register=com.xpert.controller.RegistrationController

Now it’s time to create a simple controller, and view to test…



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package com.xpert.controller;    import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    import org.techzoo.aqsa.Controller;  import org.techzoo.aqsa.View;    public class LoginController implements Controller{        @Override      public View execute(HttpServletRequest request,              HttpServletResponse response) throws Exception {          return new View("/LoginUser.jsp");      }  } |