JETServer Concepts

JETServer Concepts

Bradford M. Arant Sr.

A whitepaper on an advanced microservices development environment
for the cloud

Contents

Chapter 1

Basic Concepts

In this chapter we will cover the components that make up the JETServer environment.

1.1 Sessions

Sessions provide a congruent and flowing state management for a connected browser. The states of various elements are stored and kept as a part of the session information so a browser refresh will not disrupt the work flow intended by the application logic.

The Session Id is maintained on the browser as a cookie. The cookie is issued to the browser on the very first response from the first request to the server.

All API interactions are identified as being part of a session. A browser can only have one session per domain. Sessions can support multiple window instances and is basically the authorization token. The server can issue a new Session Id in a response to the browser at any time which will reset the cookie being created to maintin the Session Id.

The session mechanism will attempt to maintain a *state* of the interaction of the request objects present in the view port of the browser. Session variables also maintain a server based data environment for the session. Session variables are available to the views and microservice methods using the $[:*variable-name*] syntax.

As requests are placed the request URI is saved to the http session. If needed, perhaps because of a refresh, the page state can be recreated from the saved URI list. Other data and state information can be stored to the session as well.

1.2 Views

Views are document sections that can be placed into the page. Views can be static or can contain dynamic data elements. Views can also contain other views in a nested fashion.

Views are implemented as ¡DIV¿ elements within the HTML document structure.

Views can be used to define overall page layouts by putting other Requests into a view.

Other more advanced graphical elements can be created by combining views into a 'view group'.

Views have layout style specifications for the elements contained within them. Absolute layout allows a WYSIWYG layout with precise pixel location. Linear layouts line things up either vertically or horizontally. Proportional layouts use percentages so that they scale and constraint layouts are similar but offer greater flexibility in controlling the expansion/compression of the elements in the display area.

Views that have URIs attached to them are considered document level views and will be triggered when the browser makes a get request to the URI.

TABLE elements can also be specified in a view.

Javascript code can be attached to components within the view or to the view itself. Custom components can be built that plug into the layout tool environment and can be designed into the application environment with prebuilt behaviors. Complex display control can be constructed using these tools.

1.2.1 View Data Sources

Data is available to be displayed and represented that is only available when the views are rendered. Sources of this data are as follows:

1. Set variable performed in the view code or from a previously run process in the same transaction to name a piece of data for later use.

2. CGI data received in the request as form-data is available to the request handler. Use the [:variable-name] syntax to ensure that CGI data is the source of the data.

3. Session variables provide access to persistent storage that survives for the duration of the session.

4. A data result table that was retrieved from an mySQL buffer.

Standard variable syntax of just using variable-name will perform a search for a variable regardless of its type but prioritized by the following order:

1.3 View Components

Views may utilize HTML constructs to assemble part of its implementation. The use of ¡IMG¿, ¡FORM¿, ¡INPUT¿, ¡TEXTAREA¿ and a significant list of other items can be contained within a view.

The view can contain javascript that is used to interact with these components.

1.4 Forms

Using the INPUT tag within a FORM tag allows the submission of the related form data to the server to be processed as a POST request. A BUTTON can perform this activity and request a microservices method and pass the form data along as a parameter to the request.

BUTTONs are attached to server side microservice methods.

1.5 Image Library

Significant content can be provided in the form of images and are used as part of a design. The image library is not intended to be a photo cataloguing function but is designed to provide image support to the view elements such as logos. Catalogues can be accomplished through other means and usually tie into an external database system of some kind.

1.6 Work Flow Networks

JETServer is a complete work flow engine capable of delivering extremely sophisticated work flow requirements on various business data sets. Performing various activities on business entities creates needs for scheduling follow-ups and other automatic processed based upon very long time delays or other triggers that cause the need for more immediate action.

Work flow processes are initiated on a particular business entity and stick to the entity throughout the definition of the work flow process. At any one time there could be many work flow tags running on any one piece of data at a time.

Work flow processes can be initiated on a button press using the initiateWorkFlow API call. They are attached to a business entity and they may trigger other work flow processe on other entities related to various activities. Work flow process are very powerful trackers where things otherwise may slip through the cracks. Watch dog work flows can be ceated to ensure all activities are being handled and performed to avoid stale data situations.

Work Flow Networks are developed using a graphical flow chart style workbench that allows the business process analyst to snap together activities that are to be performed by automated process and/or user interaction. Work Queue objects are available to design work lists for users to interact with items requiring intervention. Users can review various work queues and perform the requested activities to complete the work flow requirements.

1.7 Requests

Requests are view 'placeholders' that can be used to make a request to the server and have the response be displayed in the viewport of the request. Requests are the fundamental building block to creating extended content pages.

When specifying a request in a view the URI must be specified. As the view is rendered to the response buffer on the server it will submit the request to the specified URI and when the response is received the placeholder will be replaced with the response data.

The requests made to the server are microservices method calls. They are not the URIs mentioned in other parts of this document and are intended to address the microservice method's address space. Since JETServer uses a technique of providing the URI to the microservice method it can provide a unique link designed for that session only with a timestamp embedded resulting in a secure interaction environment that eliminates stale access to API calls.

Requests are implemented as ¡DIV¿ elements with an id tag that provides the name of the request.

A request response will contain HTML document data that will be replaced as the innerHTML of the request 'placeholder'. Variables specified in the view will be rendered and the HTML is attached to the document structure where the request exists.

The document content of the response may contain a script tag which contains javascript that executes immediately after the HTML is attached to the document.

1.8 Business Entities

Business entities provide the framework in which to hang all thiese views and logic. Business entities allow for the description of account, customers, products, inventories and any other associated data elements that may be used in business logic consideration.

Instances of business entities are considered entity objects or entity instances. Interaction with business entities is performed using an object style interface and JETServer uses mySQL underneath to perform the persistent storage operations.

Views are associated with a business entity combined with a microservice method to create functionality.

1.9 Microservice Methods

JETServer uses a microservices style architecture and provides access to backend services through an API set that is designed to interact with the various view components of the application.

Microservice methods are oriented around the business entity structure and are used to provide information query handling as well as the add, update and delete functionality of new business entity instances.

Microservice methods can also be used to provide lists of objects that can be further used in modifying or removing instances.

Applications can be designed that employee various business logic entities. Views that define forms for adding new customers or information on books or some other data entity can be specified and attached to the business entity. Common lists of objects that have filters applied can be retrieved and views selected for presentation of the data.

1.10 URIs

A URI provides an external access string to be mapped to a site view. When a browser submits a GET request to a URI that is mapped to the server then it will return the resulting site marker forcing the session to check its current state and send the appropriate information.

The URI may be specified as *complete* or *partial*. Partial URI specification will match the pattern check but will also pass additional parameters not specified into the view so that it may parse the content and provide the appropriate display result.

1.11 Packages

Packages provide extensibility to the JETServer environment by allowing custom tools, applications, themes, templates, entities and components to be added to the server and utilized in the presentation. Installed packages are provided a root namespace and all related content must be contained and extended by this root path namespace.

Packages can contain many things including full application request flows that can be integrated into other work flows or used as is.

Local code is setup automatically in a default package named 'local'. The local package is always searched first when resolving names to finding objects.

The system itself maintains a package space which is where the tools and control logic are kept. This space is static within the JETServer executable and is not modifiable.

You can create an empty package and give it a namespace and write custom items into that package space. This package space can then be saved and distributed on the JETStore if it provides useful utility to others.

It is intended to provide a JETStore for the user to browse for registered packages of all kinds from themes to full application work flows. It is hopeful that someday there will be a rich environment for providing tools and applications for the JETServer system.

Chapter 2

ServerCore Network Platform

JETServer is utilizing the ServerCore library to implement the TCP networking requirements. Using the Linux operating system and the epoll system architecture JETServer delivers high performance networking with extreme control.

Written in C++ JETServer is designed for performance at all levels of the technical implementation. Using the core namespace along with the http namespace to create the basic core functionality JETServer provides a framework in which to create complete work flow patterns required to build complex applications.

Chapter 3

User Interfaces and Tools

3.1 View Layout Editor

The View Layout Editor provides a WYSIWYG interface for combining HTML elements to formulate the presentation. Style sheet activities and parameters are automatically maintained as elements are manipulated in a graphical environment. Previews provided by various automatic layout objects including absolute mode which allows WYSIWYG placement of elements.

Development of a theme editor provides ability to allow user to set various themes for the graphical elements of the design. Ability to view the layouts using various theme data configurations provides a useful thematic testing environment.

The designed view layouts can be saved to create a view library and are usually associated to an entity and placed into the workflow storyboard.

In order to create a subfunctionlity within a view you can add the request object into the view layouts. As these layouts are rendered to the browser they willmake their subsequent requests to the server keeping the event chain alive for the initiating or root request object.

3.2 Business Entity Editor

The Business Entity Editor provides a work management area to create and maintain business entity related data and the relationship between them. A repository provides for linking together data description elements with views and work flow elements to create a feature rich base to construct highly capable work flows for business and function.

3.3 Storyboard Editor

The Storyboard Editor provides the designer the ability to layout the pages and their components and design the flow relationships between the elements. As designs require extending the control elements of various views the storyboard editor keeps track of the request branching and allows the designer to maintain control over the hierarchical flow elements of the UI design.

The design approach provides the ability to layout *request* objects and associate various URI elements to these internal request objects to generate a complete presentation of the elements.

Various views can be layed out in the storyboard. Storyboards are based around the request object as each request object will perform a request activity which may result in a chain of further request events.

Chapter 4

JET Tag Reference

JET tags can be used within views as well as microservice methods to build business logic.

4.1 CALL

4.2 COMMENT

4.3 DATABASE

4.4 EXCLUDE

4.5 EXTRACT

4.6 FOR

4.7 IF.. ELSE

4.8 IFNOROW... ELSE

4.9 IFROW... ELSE

4.10 SYSTEM

4.11 TAG

4.12 WHILE

4.13 WHILEROW