StateManagement

Work in progress!

			Client Techniques			Server Techniques			Client + Server Techniques
	Cookies	URLs		HTML form hidden fields	Browser Objects	Session	Database	Hidden fields + session	Hidden fields + database
		Query parameters	Clean URLs						
				Advantages					
Can store complex data types					+	+			
Works well in load balancing environments	+	+	+	+	+				
Works well in Ajax applications	+				+	+			
				Disadvantages					
User can tamper with state information		+	+	+					
Requires cookies	+					+			
Requires coding	+				+	+			
Requires redirection			+						
Requires Javascript			+		+				
Can store only small amount of data (~2,000 bytes)		+	+						
Increased network traffic and slightly slower response times	+	+	+	+					
Requires server resources to save state information						+			
Application logic tightly coupled to the user interface implementation				+					
Does not work well with redirection				+	+				
State can be lost on page reload					+				
State can be lost on direct navigation		+		+	+				
State can be lost when client is idle						+			
				Neither Advantages Or Disadvantages					
State is not tied to view	+				+	+			

Client-Side Techniques

Cookies

Pro:

- Not visible to a user
- Can be submitted to originating server only - better security
- Does not depend on request method (GET or POST) or how request is issued (HTML form, link, direct URL in the address bar)
- State is not linked to page address, can not get stale when a user navigates back on browser session history.
- Can have lifetime different from a user session; can be preserved after browser is closed (persistent cookies) or can be removed during user session.

Neither:

- State and page address are not related.

Cons:

- Does not work if cookies are turned off in a browser.
- May be percieved as potential privacy concern.

Best fit: non-critical information about a particular client, session, or application. Usually employed for user profiling, like zip code or preferred language. Also used for "remember me" login.

How Struts can help:

- Struts does not have functions specific to storing state in cookies.
- Struts helps to maintain session if cookies are not enabled. Struts automatically calls `response.encodeURL()` when processing <html:form> and <html:link> tags. This tells server to add session ID to links and HTML forms.

Rewritten URLs

Query parameters

URL *query parameters* allow to express a website structure using base path to a resource, in case of Struts that would be an action, and series of query parameters

For example, an online tire store may expose its structure like this:

```
    www.tirestore.com/vehicle.do?year=2002&make=TOYOTA&model=mr2spyder
```

A series of HTML forms may help a user to get to desired location, making the address more detailed from step to step:

```
Welcome page allows to select search criteria:
    www.tirestore.com/welcome.do

We choose to search product by vehicle year and make:
    www.tirestore.com/vehicle.do?year=2002&make=TOYOTA

Then application shows a listbox with applicable model names and allows to select one:
    www.tirestore.com/vehicle.do?year=2002&make=TOYOTA&model=mr2spyder
```

Pro:

- Easy to create: by submitting a form using GET method, by clicking on a link or by manually editing the resource address.
- Reloading a page that was previously obtained with GET method does not cause POSTDATA dialog.

Cons:

- Can be easily damaged by a user.
- Exposes data that is sent to the server.
- Has limited size, usually about 2000 characters.
- State is lost if a user navigates to an arbitrary page.
- State is linked to page address, can get stale when a user navigates back on browser session history.

Neither:

- State is tied to page address.

Best fit: applications where the query string only affects the view, like online catalogs or magazines. A page can be reloaded without affecting the server and without POSTDATA messages. Navigating back and forward is not an issue.

Clean URLs

Clean URLs or *cool URLs* is a way to express website structure using location path instead of query parameters. Clean URLs work best for static websites. Representing state with clean URLs is less common. It is not convenient either, especially when location is being built as a result of an HTML form submission.

For example, an online tire store may expose its structure like this:

```
www.tirestore.com/vehicle/2002/toyota/mr2spyder/
```

A series of HTML forms may help a user to get to desired location, making the address more detailed from step to step:

```
Welcome page allows to select search criteria:
    www.tirestore.com/welcome

We choose to search product by vehicle year and make:
    www.tirestore.com/vehicle/2002/toyota/

Then application shows a listbox with applicable model names and allows to select one:
    www.tirestore.com/vehicle/2002/toyota/mr2spyder/
```

Pros:

- URLs may help visualize the site structure.
- Such URLs are more likely to be saved by search engines and crawling machines.
- Manually editing URL allows to move "up" and "down" hierarchical structure of an application just like navigating directory structure on a local computer.
- Underlying technology is not exposed; easier to replace a web framework, harder for hackers to get into.

Cons:

- Can be easily damaged by a user.
- Has limited size, usually about 2000 characters.
- Creating a clean URL from a browser requires either building it on client using Javascript or building it on server and then using redirect.

Neither:

- State is tied to page address.

Best fit: online catalogs or magazines that have hierarchical structure and should be represented as a static website. May improve site searchability and page rankings.

Additional information:

- Towards Next Generation URLs – contains links to other notable resources, like Tim Berners Lee's Cool URIs don't change and Jakob Nielsen's URL as UI.

Browser objects (DOM elements, Javascript variables, Flash storage)

Pro:

- Can store complex structures and objects.
- Depending on application requirements, state can be made relative or non-relative to page address.

Neither:

- State and page address are not related.

Cons:

- Lost when a page is reloaded unless special care is taken.

Best fit: Ajax-style Single Page applications.

How Struts can help:

- Struts does not have built-in features for saving state in client DOM/Javascript objects.

Additional information:

- Saving Session Across Page Loads Without Cookies, On The Client Side – tutorial by Brad Neuberg

HTML form hidden fields

By year, vehicle, make:

```
<form action="...">
  <input type="hidden" name="yr" value="2002"/>
  <input type="hidden" name="mk" value="TOYOTA"/>
  <input type="hidden" name="model" value="mrspyder"/>
  ...
</form>
```

Pros:

- No size limitations.
- URLs are short.

Neither:

- State and page address are not directly related.

Cons:

- Every page must be an HTTP form to transfer state from page to page.
- HTTP forms must be submitted with POST method only, hidden fields are not submitted with GET.
- State is lost on redirection or on direct navigation; breaking out of a predefined page flow is not possible.
- State information grows while a user moves forward from page to page.
- Reloading a page or navigating back/forward in browser session history causes a resubmit that is preceded by a user-unfriendly POSTDATA message.
- The application is unfriendly to search engines.

When to use: all application pages are based on HTML forms and storing state information in the URL is undesireble.

What to store: non-critical state information, like state of controls and widgets (viewstate).

What to keep in mind: the moment you update the server objects or database, you switch from pure client-based technique to combined client-server tecnhique, which is a totally different beast. Many developers using hidden fields think they use client-based state tracking technique while in fact they are not.

Implications: Application must provide explicit navigational links and buttons so users would not have to use standard browser navigation buttons. Application must recognize and handle double-submit situations.

How Struts can help:

- Use %3Chtml:form%3E tag to display data entry form, and a single ActionForm to handle both render and submit phases of request/response cycle. On render phase Struts renders HTML form fields using values from an ActionForm. On submit phase Struts automatically populates ActionForm properties with values of HTML form fields. This setup ensures that content of form fields is saved and redisplayed automatically. Automatic saving/restoring of HTML form fields is similar to handling *postback controls* in ASP.NET. Struts does not have notion of *non-postback controls* like labels or data grids, and does not automatically save such information.
- Use %3Chtml:hidden%3E tag to store stringified data in an HTML form and to submit it to location specified in <html:form> tag. Struts does not provide built-in functionality to encrypt hidden fields.
- Use TokenProcessor class to distinguish resubmits.

Server-Side Techniques

Server session object

Server session object only

Pro:

- Can be tied to model state, not affected by browser state or current page.
- Allows to separate the concerns, keeping state safely and securely where model resides, while treating browser as an agent providing interface with a user.
- Developing interactive applications is greatly simplified.

Neither:

- State and page address are not related.

Cons:

- Session size should be carefully controlled, can affect perfomance.
- Degrades performance in clustered system.
- Session object can expire when browser is still open.
- Leaving a webapp and then navigating to it again by specifying location in the address bar leads to losing state if cookies have not been enabled.

Best fit: Ajax- and non-Ajax applications that treat server state as primary, and client state is secondary. If discrepancy between view and model occurs, view is synched with the model, not vice versa.

Server session object + redirection after submitting an HTML form

Pro:

- Clean URLs (with cookies enabled)
- One URL can be used for one logical page or a series of pages, this makes back/forward navigation easier for a user.
- POSTDATA dialog is now shown when a user tries to refresh a page.
- Double submit does not occur when a user tries to refresh a page, or navigates back/forward.
- Developing interactive applications is greatly simplified.

Cons:

- Redirection may increase response time.
- Can be problematic in clustered systems or with address-translating web servers since the proper redirection location cannot be obtained.

Best fit: applications that treat server state as primary state source. Non-Ajax Single Page Applications.

How Struts can help:

- Struts allows to define session scope for an ActionForm in `struts-config.xml` file.
- EventActionDispatcher helps to structure and simplify the action code, if both input and render request phases are processed by one action.
- StrutsWebIslands – how single-page web resources (web-islands) can be implemented with Struts, no Ajax.

Combination Client-Server Techniques

HTML form hidden fields + database

This is one of the more popular state management techniques. It is often used in situations where storing complete state information on server is undesired for performance reasons. Hidden fields store intermediate state during, for example, filling out a series of forms like wizard pages. One of the classic examples is web store checkout process. A series of screens ask for name, address, payment information, delivery optins, etc. The last page in the series submits the shopping cart to the server, where it is processed, the inventory is adjusted and user's credit card is charged.

Two different states have been created: client state and server state. At this point these states are synchronized, but what happens if a user clicks Back or Reload button? A Reload button will cause resubmit of the same request, which will cause server code to charge a user's credit card and to adjust warehouse inventory one more time.

Clicking Back button will cause browser to display checkout page with the cart, this page does not represent server state anymore, here you can see that server state and client state got out of sync. Synchonizing server to client is not possible, database update has been committed and cannot be rolled back. What is left is either synchronizing view to server (preferred choice), or throwing exception and bailing out to some safe location like welcome page.

Pros:

- No size limitations.
- URLs are short.

Cons:

- Every page must be an HTTP form to transfer state from page to page.
- HTTP forms must be submitted with POST method only, hidden fields are not submitted with GET.
- State is lost on redirection or on direct navigation; breaking out of a predefined page flow is not possible.
- State information grows while a user moves forward from page to page.
- Reloading a page or navigating back/forward in browser session history causes a resubmit that is preceded by a user-unfriendly POSTDATA message.
- The application is unfriendly to search engines.

When to use: all application pages are based on HTML forms and storing state information in the URL is undesireble.

What to store: non-critical state information, like state of controls and widgets (viewstate).

Implications: With this technique state information is saved on server from time to time (checkpoints). Thus, synchronization between client and server state will be required. Moving back in browser session history should either be prohibited after a "save-to-server" checkpoint, or application must be able to synchronize client and server state. Application must recognize and handle double-submit situations. Application must provide explicit navigational links and buttons so users would not have to use standard browser navigation buttons.

How Struts can help:

- Use %3Chtml:hidden%3E tag to store hidden field information and to submit it to location specified in <html:form> tag.
- Use TokenProcessor class to distinguish resubmits.

todo, saved for later

- Same URL can be used for several pages/forms or same URL can be used to redisplay data entry form. In most browsers this allows to reduce number of entries in page history buffer and to provide Single Page Application behavior.
- If submitting a form changes server state like updating a database, then resubmitting a request can potentially produce cumulative (and wrongful) changes to server state. At this point state is handled both on client and server and its synchronization becomes problematic.

  Example: a typical ASP.NET WebForms application.