Cookies

Storage with Cookies

Many applications require the use of persistent storage, or data that can be saved for later retrieval. The time frame for retrieval may be just over the course of a single session (i.e. until the browser is closed) or over multiple sessions (i.e. for a month). Because HTTP (the underlying protocol powering the www) does not inherently maintain state, another mechanism had to be used. This is the role of the cookie.

A cookie is a means by which a server and client can share data across several request/response cycles or in plain terms a way for the client and server to share data. The cookie holds simple key-value pairs that can be used to store a user's user-name and password, contents of a shopping cart, desired website theme colours, etc.

Cookies are transferred along with HTTP requests and responses and allow a HTTP server to effectively *remember* a specific client. Each cookie contains several fields that are understood by HTTP servers. The following table outlines some of these fields.

Field	Description
Domain	specifies which domain/sub-domain the cookie is valid for
Path	specifies the path for which the cookie is valid
Expires	specifies when the cookie should be discarded
HttpOnly	specifies that only HTTP may be used to request the cookie
Secure	specifies that only secure connections may be used to request the cookie

Table 1: common cookie fields

The following shows an example of a simple cookie string:

```
key=value; path=/; expires=Fri, 23 Feb 2012 09:37:51 GMT
```

Code snippet 1: sample cookie string

Note that cookies do not contain executable code and do not contain malware. This does not mean however that the security of cookies hasn't been circumvented, for example they may be used to track your movement on the Internet ... as with any technology there are pros and cons to be aware of.

For more on cookies, please see the content here.

JavaScript and Cookies

Many languages provide support for cookies, but here we are of course interested in JavaScript.

Setting Cookies

In JavaScript, the document object can be used to retrieve and set cookies through its **cookie** property. The value of this property can be set by creating a valid cookie string (see code snippet 1 above) and assigning it to the property.

```
document.cookie = "foo=bar; path=/";
```

Code snippet 2: setting a session cookie in JavaScript

```
var expiryDate = new Date(2012,11,31); // end of 2012

document.cookie = "foo=bar; path=/; expires="
    + expiryDate.toGMTString();
```

Code snippet 3: setting a persistent cookie in JavaScript

Several cookies may be set for a single site (each browser has a different limit on how many are allowed per site) using this method, however retrieval is a little trickier.

Retrieving Cookies

While cookies may be set individually, they are retrieved as one string with semi-colons separating each key/value pair.

```
key1=value1; key2=value2; key3=value3
```

Code snippet 4: sample of a retrieved cookie string

This means that the cookie string must be parsed in order to retrieve the desired data. One way to do this would be to split the cookie string on the ';' characters and then, for each element in the resulting array, split on the '=' character.

Removing Cookies

Cookies can be removed by simply setting them to an invalid value and setting the expires attribute to a time in the past.

```
var expiryDate = new Date(1970,1,1); // land before time

document.cookie = "foo=-1; path=/; expires="
    + expiryDate.toGMTString();
```

Code snippet 5: invalidating a cookie in JavaScript

Local and Session Storage

DOM Data Storage

Working with cookies can be difficult and tedious but is a requirement if you need to communicate session state between the client and server. If however you don't need to inform the server with every request, you can make use of the new DOM storage options: `sessionStorage` and `localStorage`. These objects are effectively key/value hash structures that allow a developer to easily work with data on the client. The following table lists the methods available for working with the data:

Method	Description
key(index)	Returns the 'key' at the specified index
setItem(key, value)	Sets/updates the value for 'key'
getItem(key)	Returns the value for 'key'
removeItem(key)	Removes the item 'key'
clear()	Clears the session storage

Table 1: DOM storage methods

It is important to note that DOM storage only supports the storage of string data, and cannot store more complex structures like arrays and objects.

Session Storage

Data storage that only needs to be maintained for the current life of the session (i.e. while the browser is open) can be achieved by using the `sessionStorage` object available in most modern browsers. This object provides an interface to store/retrieve, and clear temporary data on the client that is associated with the current site.

```
// set a new session item

sessionStorage.setItem("username", "jonnyD");
```

Code snippet 1: setting a sessionStorage item

```
// retrieve a session item (display 'jonnyD')

console.log(sessionStorage.getItem("username"));
```

Code snippet 2: setting a sessionStorage item

```
// remove a session item

sessionStorage.removeItem("username");
```

Code snippet 3: setting a sessionStorage item

Local Storage

Data storage that needs to be maintained indefinitely (i.e. even after the browser is closed) can be achieved by using the `localStorage` object available in most modern browsers. This object provides an interface to store/retrieve, and clear persistent data on the client that is associated with the current site. The interface for the localStorage object is identical to that of the `sessionStorage`.