Local data

intro to web storage

Most web applications rely on some method of data storage, which usually involves a server-side solution such as a SQL Server database.

However, in many scenarios, that might be excessive, and the ability to store simple, non-sensitive data in your browser would easily meet your needs.

understanding cookies

```
// setting the cookie value
function setCookie(cookieName, cookieValue, expirationDays) {
    var expirationDate = new Date();
    expirationDate.setDate(expirationDate.getDate() + expirationDays);
    cookieValue = cookieValue + "; expires=" + expirationDate.toUTCString();
    document.cookie = cookieName + "=" + cookieValue;
}

// retrieving the cookie value
function getCookie(cookieName)
{
    var cookies = document.cookie.split(";");

    for (var i = 0; i < cookies.length; i++) {
      var cookie = cookies[i];
      var index = cookie.indexOf("=");
      var key = cookie.substr(0, index);
      var val = cookie.substr(index + 1);

      if (key == cookieName)
        return val;
    }
}

// usage
setCookie('firstName', 'Glenn', 1);
var firstName = getCookie('firstName');
```

jQuery plug-in

```
$.cookie('firstName', 'Glenn');
var firstName = $.cookie('firstName');
```

Working with cookie limitations

Cookies will continue to be an effective tool for the foreseeable future, but they have some drawbacks.

- **Capacity limitations** Cookies are limited to about 4 KB of data, which is not large, although you can create more than 30 cookies per site. (The actual maximum limit depends on which browsers you are targeting; the average is between 30 and 50.)
- **Overhead** Every cookie is sent with each HTTP request/response made, regardless of whether the values are needed. This is often true even for requests for static content (such images, css files, and js files), which can create heavier-than-necessary HTTP messages.

Understanding HTML5 storage

Existing solutions leave a lot to be desired; HTML5 breaks new ground with several innovative tools. Each is unique and carries its own set of pros and cons, which this chapter discusses individually.

- **Web storage** Easily the simplest new form of storage, web storage provides a way to store key/value pairs of data in a manner that rivals cookies in ease of use. In the next section, you see that it's currently the most widely supported option.
- **Web SQL database** For more complex applications, this might be a good alternative to web storage. It provides the power of a full relational database, including support for SQL commands, transactions, and performance tuning. Unfortunately, its support is extremely limited, and it might be left behind in favor of other options.
- **IndexedDB** This tool appears to be a strong candidate for the solution to complex storage requirements in the future. As a non-relational (NoSQL) database, it provides simplicity that's similar to web storage while still accommodating common needs such as indexing and transactions.
- **Filesystem API** This tool is useful for storing larger data types such as text files, images, and movies. However, it suffers from a lack of adoption by many of today's modern browsers. As of this writing, it's primarily supported in Chrome only.

Exploring localStorage

Using the *localStorage* object reference

The following is a list of methods and attributes available on the Storage object as it pertains to *localStorage*.

- **setItem(key, value)** Method that stores a value by using the associated key. The following is an example of how you can store the value of a text box in *localStorage*. The syntax for setting a value is the same for a new key as for overwriting an existing value.

```
localStorage.setItem('firstName', $('#firstName').val());
And since it's treated like many other JavaScript dictionaries, you could also set
values using other common syntaxes.
localStorage['firstName'] = $('#firstName').val();
or
localStorage.firstName = $('#firstName').val();
```

Exploring localStorage

- **getItem(key)** Method of retrieving a value by using the associated key. The following example retrieves the value for the 'firstName' key. If an entry with the specified key does not exist, null will be returned.

```
var firstName = localStorage.getItem('firstName');
And like setItem, you also have the ability to use other syntaxes to retrieve
values from the dictionary.
var firstName = localStorage['firstName'];
or
var firstName = localStorage.firstName;
```

Exploring localStorage

- **removeItem(key)** Method to remove a value from *localStorage* by using the associated key. The following example removes the entry with the given key. However, it does nothing if the key is not present in the collection.

```
localStorage.removeItem('firstName');
```

Exploring localStorage

- **clear()** Method to remove all items from storage. If no entries are present, it does nothing. The following is an example of clearing the *localStorage* object.

```
localStorage.clear();
```

Exploring localStorage

- **length** Property that gets the number of entries currently being stored. The following example demonstrates the use of the length property.

```
var itemCount = localStorage.length;
```

Exploring localStorage

- **key(index)** Method that finds a key at a given index. The World Wide Web Consortium (W3C) indicates that if an attempt is made to access a key by using an index that is out of the range of the collection, null should be returned. However, some browsers will throw an exception if an out-of-range index is used, so it's recommended to check the length before indexing keys.

```
var key = localStorage.key(1);
```

Does browser support?

```
function isWebStorageSupported() {
    return 'localStorage' in window;
}
```

```
if (isWebStorageSupported()) {
    localStorage.setItem('firstName', $('#firstName').val());
}
```

The popular JavaScript library, Modernizr, comes with a method that could do this check for you.

```
if (Modernizr.localstorage) {
    localStorage.setItem('firstName', $('#firstName').val());
}
```

reaching the limit

```
try{
    localStorage.setItem('firstName', $('#firstName').val());
}
catch(e) {
    // degrade gracefully
}
```

Thanks