第5章 离线应用和Web Workers

5.1 HTML5 离线应用概述


观看视频

Web 应用程序的资源都是存储在 Web 服务器上的，如果客户端无法连接网络，或者 Web 服务器不能提供服务，又或者网速较慢，那么传统的 Web 应用程序就无法运行了。目前，基于 Web 的应用程序越来越普遍，也越来越复杂，因此访问 Web 应用程序的速度就显得尤为重要。提高 Web 应用程序的访问速度有许多方法，网页缓存是其中之一，但是网页缓存仍然要依靠互联网，并且因便携性和网络性能等原因，网页缓存对移动 Web 应用程序的实际效果影响并不大。

HTML5 离线缓存又名为 ApplicationCache，是从浏览器的缓存中分出来一块缓存区，用来存储一定的资源，属于 HTML5 的新特性。HTML5 使用离线缓存机制，将构成 Web 应用程序的资源文件，例如 HTML 文件、CSS 文件、JavaScript 脚本等存储到本地缓存中，这样可以使 Web 应用程序在离线状态时也能正常工作。

Web 应用程序的离线缓存和浏览器的网页缓存有明显的区别，主要表现为以下几点。

1. 缓存目标

离线缓存的目标是整个 Web 应用程序，而浏览器的网页缓存针对的是单个网页。任何网页都具有网页缓存，而离线缓存只会保存指定的资源。

2. 安全性

离线缓存的安全性优于浏览器网页缓存。浏览器网页缓存是强制进行的。而对于离线缓存，可以指定被缓存的资源，还可以利用编程手段来控制缓存的更新，可以利用缓存对象的各种属性、状态和事件来开发 Web 应用程序。

3. 对互联网的依赖

有些网页缓存会主动保存缓存文件以加快网站的加载速度，但是必须有有效的网络连接。如果启动了网页缓存，但是没有有效的网络连接，则客户端会收到无法连接到服务器的错误提示信息。而对于离线缓存来说，可以在有网络连接时从服务器获取指定的资源；当无网络连接时，利用离线缓存的资源仍然可以访问 Web 应用程序。

5.2 ApplicationCache 对象


观看视频

JavaScript 为 HTML5 的离线存储这一新特性提供了专门的接口，即 AppliationCache

API。当把缓存文件保存到本地时，如果缓存的文件有更新，需要更新本地的缓存文件。利用 ApplicationCache API 可以动态地控制缓存。新的 window.applicationCache 对象可以触发一系列与缓存状态有关的事件。applicationCache 对象和缓存宿主的关系是一一对应的，window 对象的 applicationCache 属性会返回关联 window 对象的活动文档的 ApplicationCache 对象。

HTML5 的新特性并没有得到所有浏览器的支持，离线缓存也一样。IE 9 及 IE 9 以下的浏览器目前不支持离线缓存。可以使用以下代码检测客户端的浏览器是否支持离线缓存。

```
<script type="text/javascript">
    if(window.applicationCache){
        alert("支持离线缓存");
    }
    else{
        alert("不支持离线缓存");
    }
</script>
```

5.2.1 属性

applicationCache 对象的 status 属性可以返回当前 applicationCache 的状态，它的可取值及其解释如下。

- 0：表示 uncached，即未缓存，applicationCache 对象的缓存宿主与应用缓存无关联。
- 1：表示 idle，即空闲，应用缓存已经是最新的，并且没有标记为 obsolete。
- 2：表示 checking，即检查中，applicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 checking。
- 3：表示 downloading，即下载中，applicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 downloading。
- 4：表示 updateready，即更新就绪，applicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 idle，并且没有标记为 obsolete，但是缓存不是最新的。
- 5：表示 obsolete，即过期，applicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 obsolete。

如果 Web 应用程序没有使用离线缓存，即没有指定缓存清单，则这些页面的状态就是 uncached（未缓存）。idle（空闲）是缓存清单中资源的典型状态，处于空闲状态说明应用程序的所有资源已经缓存，当前不需要更新。如果缓存曾经是有效的，但是当前状态下缓存清单已经丢失，则缓存进入 obsolete（过期）状态。

5.2.2 事件

对于不同的状态，ApplicationCache API 提供了特定的事件和回调特性，例如当缓存更新完成进入空闲状态时，会触发 cached 事件。applicationCache 缓存对象的事件如下。

- checking：当检查更新时，或者第一次下载 manifest 清单时，checking 事件第一个被

触发。

- noupdate：当检查到 manifest 清单文件不需要更新时，触发该事件。
- downloading：第一次下载或者更新 manifest 清单时，触发该事件。
- progress：该事件与 downloading 事件类似，但 downloading 事件只触发一次，progress 事件在清单文件下载过程中可以周期性被触发。
- cached：当 manifest 清单文件下载完毕及成功缓存后，触发该事件。
- updateready：此事件的含义表示缓存清单文件已经下载完毕，可通过重新加载页面读取缓存文件或者通过 swapCache()方法切换到新的缓存文件。
- obsolete：当访问 manifest 缓存文件返回 HTTP404 错误(请求的资源找不到或者已经被删除)或者 410 错误(请求的资源已经被删除)时，触发该事件。

5.3　离线缓存的实现


观看视频

使用离线缓存需要经过以下几个步骤。

1. 配置服务器 manifest 文件的 MIME 类型

离线存储通过 manifest 文件来管理，需要 Web 服务器的支持，不同的 Web 服务器的配置方式不同。以 Tomcat 服务器为例，需要修改 web.xml 文件，web.xml 文件一般位于 Tomcat 安装目录下的 conf 目录下，需要在 web.xml 文件中添加以下代码：

```
<mime-mapping>
    <extension>manifest</extension>
    <mime-type>text/cache-manifest</mime-type>
</mime-mapping>
```

注意：<extension>标签和</extension>标签中的内容必须和 manifest 清单文件的后缀名完全一致。

2. 编写 manifest 文件

离线缓存的 Web 应用程序中包括一个 manifest 清单文件，此文件实际上是一个文本文件，列出了需要离线缓存的所有资源。manifest 文件的第一行必须以 CACHE MANIFEST 开头，注释以＃开头。例如下列代码为一个完整的 manifest 文件的内容。

```
CACHE MANIFEST
#version 1.0
index.html
css/my.css
image/f1.jpg
image/f2.jpg
js/min.js
NETWORK:
image/button.jpg
CACHE:
image/girl.jpg
FALLBACK:
/app/ajax/ default.html
```

manifest 文件可以分为 3 部分，分别是 CACHE MANIFEST、NETWORK 和

FALLBACK。在以上代码中，第一行是必需的。如果 manifest 文件以及 manifest 文件所列出的资源无法加载，则整个缓存的更新过程无法进行，浏览器会使用最后一次成功的缓存。在 CACHE MANIFEST 下列出的资源将在首次访问后缓存到本地。在 NETWORK 下列出的文件，每次访问都需要与服务器连接，从服务器获取资源，并且不会被缓存。NETWORK 是每次都需要请求服务器加载的文件，可以使用星号来指示其他所有资源/文件都需要互联网连接。在 FALLBACK 下列出的文件指定无法建立网络连接时的回退页面，例如上述代码中如果无法访问/app/ajax/下的资源，则使用 default.html 替代其内容。CACHE、NETWORK、FALLBACK 在 manifest 中的顺序是任意的，每一部分都可以出现一次或多次。CACHE 是必需的，NETWORK 和 FALLBACK 是可选的。

3. 在页面的 html 标签中定义 manifest 属性引用 manifest 文件

manifest 属性规定了文档的缓存 manifest 的地址，Web 应用程序中每个要缓存的资源页面都要包含 manifest 属性。可以使用绝对的 URL，例如 http://www.example.com/demo.manifest；也可以使用相对的 URL，例如 demo.manifest。


观看视频

5.4 离线缓存的更新

如果已经完成了 Web 服务器的配置、manifest 文件的编写、html 标签 manifest 属性的设置，则 manifest 文件中指定的资源可以实现离线缓存。但是如果 Web 应用程序的内容发生了更改，并且已上传到服务器，则客户端访问服务器时看不到最新的结果，这是因为 HTML5 的离线缓存还没有更新。更新 HTML5 的离线缓存主要有 3 种方法。

1. 清除离线缓存的资源

不同的浏览器清除离线缓存资源的方法不一定相同，有的浏览器只清除历史记录，无法清除离线缓存的资源。以 Chrome 浏览器为例，输入 chrome://appcache-internals/，可以查看本地的离线缓存，也可以进行删除。

2. 更新 manifest 文件

浏览器检测到 manifest 文件更新后，会主动更新本地缓存。假如没有更新 manifest 文件，即使对缓存清单中的资源进行了修改，浏览器依旧会顽强地从本地缓存中读取修改之前的文件。在 manifest 文件中，以 # 开头的是注释行，但也可以满足其他用途。例如修改注释行中的日期和版本号是一种使浏览器重新缓存文件的办法。

3. 使用 applicationCache 对象的 update()方法更新资源

如果要以编程的方式更新缓存，并且已经更新了 manifest 文件，则需要先调用 applicationCache.update()方法，此操作将尝试更新用户的缓存。然后，当 applicationCache.status 处于 UPDATEREADY 状态时，调用 applicationCache.swapCache()即可将原缓存换成新缓存。例如：

```
var appCache = window.applicationCache;
appCache.update();
…
if (appCache.status == window.applicationCache.UPDATEREADY) {
    appCache.swapCache();
}
```

以上方式只是使浏览器检查是否有新的 manifest 清单、下载指定的更新内容以及更新离线缓存。如果要向用户提供最新的资源，还需要两次重新加载资源，一次是获得新的应用缓存，另一次是刷新资源。

要避免重新加载资源的麻烦，可以使用监听器，以监听网页加载时的 updateready 事件。例如：

```
window.addEventListener("load", function(e) {
                window.applicationCache.addEventListener("updateready", function(e) {
                        if (window.applicationCache.status == window.applicationCache.
UPDATEREADY) {
                        window.applicationCache.swapCache();
                        if(confirm("服务器有更新,是否重新装载?")) {
                            window.location.reload();
                        }
                    } else {
                        console.log("manifest 没有改变");
                    }
                }, false);
            }, false);
```

5.5　离线缓存应用示例


观看视频

5.5.1　缓存首页

本示例为一个简单的首页缓存。首先新建 Web 应用程序 ch05，并创建首页 index.html，及其引用的样式表 my.css。配置 Tomcat 服务器的 manifest 清单文件的 MIME 类型。然后创建 manifest 文件，命名为 index.manifest。最后在 index.html 的 html 标签中添加 manifest 属性，并指明使用的 manifest 清单。使用 JavaScript 实现自动更新。全部完成以后的各文件代码如下。

index.html 的代码如下：

```
<!DOCTYPE html>
<html manifest="index.manifest">
    <head>
        <title>缓存首页</title>
        <link rel="stylesheet" type="text/css" href="css/my.css">
        <script type="text/javascript">
            window.addEventListener("load", function(e) {
                window.applicationCache.addEventListener("updateready", function(e) {
                        if (window.applicationCache.status == window.applicationCache.
UPDATEREADY) {
                        window.applicationCache.swapCache();
                        if(confirm("服务器有新版本的资源,是否加载?")) {
                            window.location.reload();
                        }
                    } else {
                        console.log("manifest 没有改变");
```

```
                    }
                }, false);
            }, false);
        </script>
    </head>
    <body>
        <h1>这是我的首页!</h1>
    </body>
</html>
```

my.css 的代码如下：

```
@CHARSET "UTF-8";
h1{
    color:green;
}
```

index.manifest 的代码如下：

```
CACHE MANIFEST
#version 1.0
CACHE:
    index.html
    css/my.css
NETWORK:
    *
FALLBACK:
```

图 5-1　index.html 的显示结果

在 Tomcat 服务器上部署 Web 应用程序 ch05，启动 Tomcat 服务器，请求 index.html 的显示结果如图 5-1 所示。

如果停止运行 Tomcat 服务器，继续在浏览器中请求 index.html，则显示结果仍然如图 5-1 所示。如果没有使用离线缓存 index.html，则此时会显示服务器连接错误的提示。

如果要更新 index.html 或者 my.css，则需要同时更新 manifest 清单。例如将 my.css 中 h1 的 color 更改为 red，同时将 manifest 清单中的版本号更改为 #version 1.1，重新请求 index.html 时会弹出对话框提示服务器有新的资源，是否重新装载，如果选择是，则会以新的样式表渲染 index.html。

5.5.2　缓存图像

与缓存首页类似，可以将 Web 应用程序中的图像等资源缓存，对应的文件代码分别如下。

image.html 的代码如下：

```
<!DOCTYPE html>
<html manifest="image.manifest">
```

```
    <head>
        <meta charset="UTF-8">
        <title>缓存图像</title>
        <script type="text/javascript">
            window.addEventListener("load", function(e) {
                window.applicationCache.addEventListener("updateready", function(e) {
                        if(window.applicationCache.status == = window.applicationCache.
UPDATEREADY) {
                        window.applicationCache.swapCache();
                        if(confirm("服务器有新版本的资源,是否加载?")) {
                            window.location.reload();
                        }
                    } else {
                        console.log("manifest 没有改变");
                    }
                }, false);
            }, false);
        </script>
    </head>
    <body>
        <img src="image/back1.jpg" width="120" />
        <img src="image/back2.jpg" width="120" />
    </body>
</html>
```

image.manifest 的代码如下：

```
CACHE MANIFEST
#version 1.0
CACHE:
    image.html
    image/back1.jpg
NETWORK:
    *
FALLBACK:
```

在网页 image.html 中引用的两幅图像 back1.jpg 和 back2.jpg，对 back1.jpg 进行了离线缓存，但是 back2.jpg 没有离线缓存。当 Web 服务器可以访问时，image.html 的显示结果如图 5-2 所示，此时两幅图像都可以正常显示。当 Web 服务器停止运行时，image.html 的显示结果如图 5-3 所示，此时只有第一幅图像 back1.jpg 可以正常显示，第二幅图像因为没有离线缓存，所以在离线状态不能正常访问。

图 5-2　image.html 在线状态时的显示结果

图 5-3　image.html 离线状态时的显示结果

5.6 Web Workers

在之前的 Web 应用程序中，由于所有的处理都是单线程执行的，如果脚本的运行时间较长，则界面会一直处于停止响应的状态，因此用户的体验效果不理想。Web Workers 为网页的脚本提供了一种在后台进程中运行的方法。Web Workers 是运行在后台的 JavaScript，不会影响前台页面的性能。Web Workers 运行期间，页面的单击、选取内容等不受影响。


观看视频

5.6.1 Web Workers 概述

Web Workers 允许开发人员编写能够长时间运行而不被用户中断的后台程序，用于执行事务或者逻辑，并同时保证页面对用户的及时响应。Web Workers 为 Web 前端网页上的脚本提供了一种能在后台进程中运行的方法。一旦它被创建，Web Workers 就可以通过 postMessage()方法向任务池发送任务请求，执行完之后再通过 postMessage()方法返回消息给创建者指定的事件处理程序。Web Workers 进程能够在不影响用户界面的情况下处理任务，并且，它还可以使用 XMLHttpRequest 来处理 I/O。但是，后台进程不能对 DOM 进行操作。如果希望后台程序处理的结果能够改变 DOM，只能通过返回消息给创建者的回调函数进行处理。

利用 Web Workers 可以做以下事情：

- 可以加载一个 JavaScript 文件进行大量的复杂计算而不挂起主进程，并通过 postMessage、onMessage 进行通信。
- 可以在 Worker 中通过 importScripts(url)加载其他的脚本文件。
- 可以使用 setTimeout()、clearTimeout()、setInterval()和 clearInterval()方法。
- 可以使用 XMLHttpRequest 进行异步请求。
- 可以访问 navigator 的部分属性。
- 可以使用 JavaScript 的核心对象。

但是 Web Workers 也存在一些局限性，主要表现如下：

- 不能跨域加载 JavaScript。
- Worker 内的代码不能访问 DOM。
- 各个浏览器对 Web Workers 的实现不完全一致。
- 某些浏览器不支持 Web Workers，例如 IE 11 之前的浏览器。

5.6.2 Web Workers 成员

要使用 Web Workers 必须创建 Web Workers 对象，并传入希望执行的 JavaScript 文件。

HTML5 中的 Web Workers 分为两种不同的线程类型，一种称为专用线程(Dedicated Worker)，另一种称为共享线程(Shared Worker)，Shared Worker 也是 Worker，但是多个页面可以共用一个 Shared Worker 后台线程，并且可以通过该后台线程共享数据。

创建 Worker 的代码如下：

```
var worker = new Worker(url);
```

url 用于指定后台 JavaScript 脚本文件的 URL 地址。

创建 Shared Worker 的方法与创建 Worker 的方法类似，只是构造器略有不同。

```
var worker = new SharedWorker(url, [name]);
```

该方法的第一个参数用于指定后台线程文件的 URL 地址，该脚本文件中定义了后台线程要执行的处理。第二个参数可选，用于指定 Worker 的名称。当用户创建多个 Shared Worker 对象时，脚本程序将根据创建 Shared Worker 对象时使用的 url 参数与 name 参数来确定是否创建不同的线程。

Web Workers 对象发送的消息和错误信息需要使用事件监听器监听。如果要在 Web Workers 和页面之间通信，则需要通过 postMessage()函数传递。在线程调用的 JavaScript 脚本文件中，所有可用的变量、函数与类如下。

- self：表示本线程范围内的作用域。
- postMessage(message)：向创建线程的源窗口发送信息。
- onmessage：获取接收消息的事件句柄。
- importScript(urls)：导入其他的 JavaScript 脚本文件。
- navigator 对象：与 window.navigator 对象类似，可以用来标识浏览器的字符。
- sessionStorage 和 localStorage：在线程中可以使用的 Web Storage。
- XMLHttpRequest：在线程中可以处理 AJAX 请求。
- Web Workers：在线程中可以嵌套线程。
- setTimeout()、setInterval()、clearTimeout()和 clearInterval()：在线程中可以实现定时处理。
- close：结束本线程。
- eval()、isNaN()、escape()等：可以使用的所有 JavaScript 核心函数。
- object：可以使用本地对象。
- WebSockets：可以使用 WebSockets API 向服务器发送消息和从服务器接收消息。

5.6.3　Web Workers 示例

1. 本示例使用 Web Workers 完成较烦琐的计算

1）在主程序 computeMain.html 中创建 Worker 实例

computeMain.html 的代码如下：

```
<!DOCTYPE html>
    <head>
        <meta charset="UTF-8">
        <title>计算</title>
        <script type="text/JavaScript">
            function init() {
                //创建执行运算的线程
                var worker = new Worker("js/compute.js");
                //接收从线程中传出的计算结果
                worker.onmessage = function(event) {
```

```
                    //使用 DIV 显示计算结果
                    document.getElementById("result").innerHTML += event.data + "<br />";
                };
            }
        </script>
    </head>
    <body onload="init()">
        <div id="result"></div>
    </body>
</html>
```

2）在 compute.js 中调用 postMessage()方法返回计算结果

compute.js 的代码如下：

```
var i = 0;
function count(){
    for(var j = 0,sum = 0;j < 100;j++){
        for(var i = 0;i < 1000000;i++){
            sum += i;
        }
    }
    //向主线程发送消息
    postMessage(sum);
}
postMessage("计算之前的时间:" + new Date());
count();
postMessage("计算之后的时间:" + new Date());
close();
```

3）演示结果

在 Chrome 浏览器中请求 computeMain.html，其显示结果如图 5-4 所示。由于计算是在后台线程进行的，因此并没有出现停止响应的现象。

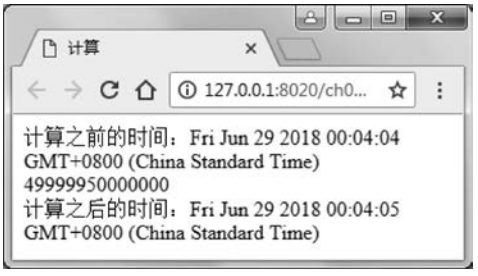

图 5-4　computeMain.html 的显示结果

2. 后台生成若干随机数，并将随机数和其中的素数发送到前台

1）主程序

primeNumberMain.html 的代码如下：

```
<!DOCTYPE html>
<html>
    <head>
```

```
        <meta charset = "UTF - 8">
        <title>从随机数中选择素数</title>
    </head>
    <body>
        <h2>生成的随机数是:</h2>
        <span id = "number"></span>
        <h2>其中的素数是:</h2>
        <span id = "result"></span>
        <script type = "text/javascript">
            //生成随机数,将结果存入字符串 str,使用分号分隔
            var array = new Array(50);
            var str = "";
            for(var i = 0; i < 50; i++) {
                array[i] = Math.floor((Math.random() * 100));
                if(i != 0)
                    str += ";";
                str = str + array[i];
            }
            document.getElementById("number").innerHTML = str;
            //将生成的随机数发送给后台线程
            var worker = new Worker("js/primeNumber.js");
            worker.postMessage(str);
            //接收后台发送的数据并显示到页面中
            worker.onmessage = function(event) {
                document.getElementById("result").innerHTML = event.data;
            }
        </script>
    </body>
</html>
```

2）脚本文件

primeNumber.js 的代码如下：

```
onmessage = function(event) {
    var result = "";
    //接收前台传送的数据
    var str = event.data;
    //将字符串分隔成整型数组
    var array = str.split(";");
    //判断素数
    for(var i = 0; i < array.length; i++) {
        var flag = true;
        for(var j = 2; j < array[i]; j++) {
            if(array[i] % j == 0) {
                flag = false;
                break;
            }
        }
        if(flag == true && array[i] >= 2)
            result = result + array[i] + ";";
    }
    //将结果发送到前台
```

```
        postMessage(result);
        //关闭线程
        close();
    }
```

3）演示结果

在浏览器中请求 primeNumberMain.html，其显示结果如图 5-5 所示，由于是生成随机数，因此每次刷新显示的结果可能不相同。

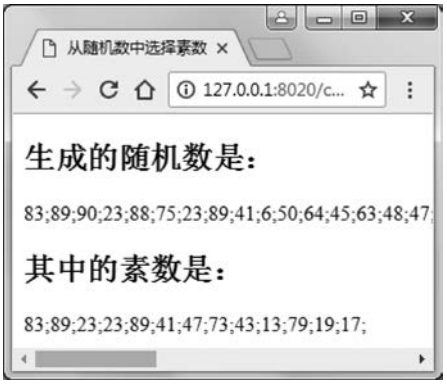

图 5-5　primeNumberMain.html 的显示结果

小结

- HTML5 离线缓存又称为 ApplicationCache，是从浏览器的缓存中分出来的一块缓存区，在此缓存区中可以存储一定的资源。
- JavaScript 为 HTML5 的离线存储这一新特性提供了专门的接口，即 ApplicationCache API。
- applicationCache 对象的 status 属性可以返回当前 applicationCache 的状态。
- 对于不同的状态，ApplicationCache API 提供了特定的事件和回调特性，例如当缓存更新完成进入空闲状态时，会触发 cached 事件。
- 使用 HTML5 的离线缓存需要 3 个步骤，分别为配置服务器 manifest 文件的 MIME 类型、编写 manifest 文件、在页面的 html 标签中定义 manifest 属性引用 manifest 文件。
- 可以通过清除离线缓存的资源、更新 manifest 文件、使用 applicationCache 对象的 update()方法更新本地缓存资源。
- Web Workers 允许开发人员编写能够长时间运行而不被中断的后台程序，用于执行事务或者逻辑，同时保证页面对用户的及时响应。
- Web Workers 可以通过 postMessage()方法向任务池发送任务请求，执行完之后再通过 postMessage()方法返回消息给创建者指定的事件处理程序。

习题

1. 创建一个 Worker 线程的方法是(　　)。
   A. new Worker(url);　　B. create Worker(url);
   C. start Worker(url);　　D. set Worker(url);
2. Worker 线程文件中使用(　　)方法向 HTML 页面回传数据。
   A. onMessage()　　B. getMessage()
   C. postMessage()　　D. 以上都不对
3. 什么是离线的 Web 应用程序？为什么要开发离线的 Web 应用程序？
4. 使用离线缓存需要经过几个步骤？
5. manifest 文件主要包括哪些内容？
6. 实现前台页面和后台线程互相传递数据有几种方法？