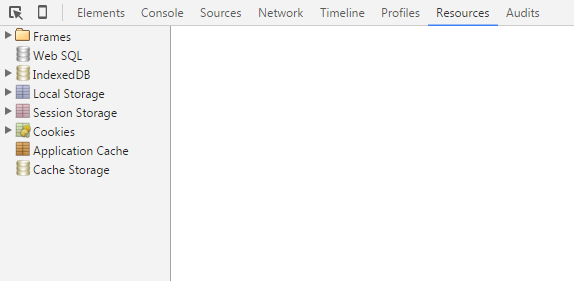
浏览器缓存（Browser Caching）是浏览器端保存数据用于快速读取或避免重复资源请求的优化机制，有效的缓存使用可以避免重复的网络请求和浏览器快速地读取本地数据，整体上加速网页展示给用户。浏览器端缓存的机制种类较多，总体归纳为九种，这里详细分析下这九种缓存机制的原理和使用场景。打开浏览器的调试模式->resources左侧就有浏览器的8种缓存机制。

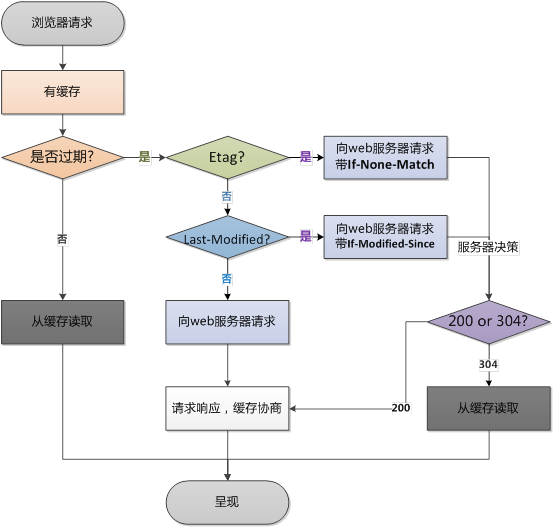
  浏览器缓存（Browser Caching）是浏览器端保存数据用于快速读取或避免重复资源请求的优化机制，有效的缓存使用可以避免重复的网络请求和浏览器快速地读取本地数据，整体上加速网页展示给用户。浏览器端缓存的机制种类较多，总体归纳为九种，这里详细分析下这九种缓存机制的原理和使用场景。打开浏览器的调试模式->resources左侧就有浏览器的8种缓存机制。

[](http://7tszky.com1.z0.glb.clouddn.com/Fh5DJkgam82rD_89kXdaxA3cWBC7)

**一、http缓存**

  http缓存是基于HTTP协议的浏览器文件级缓存机制。即针对文件的重复请求情况下，浏览器可以根据协议头判断从服务器端请求文件还是从本地读取文件，chrome控制台下的Frames即展示的是浏览器的http文件级缓存。以下是浏览器缓存的整个机制流程。主要是针对重复的http请求，在有缓存的情况下判断过程主要分3步：

* 判断expires，如果未过期，直接读取http缓存文件，不发http请求，否则进入下一步
* 判断是否含有etag，有则带上if-none-match发送请求，未修改返回304，修改返回200，否则进入下一步
* 判断是否含有last-modified，有则带上if-modified-since发送请求，无效返回200，有效返回304，否则直接向服务器请求

[](http://7tszky.com1.z0.glb.clouddn.com/Fg9HGSm1K7AlSY_67ZwhTbKWG7QA)

如果通过etag和last-modified判断，即使返回304有至少有一次http请求，只不过返回的是304的返回内容，而不是文件内容。所以合理设计实现expires参数可以减少较多的浏览器请求。

**二、websql**

   websql这种方式只有较新的chrome浏览器支持，并以一个独立规范形式出现，主要有以下特点

* Web Sql 数据库API 实际上不是HTML5规范的组成部分；
* 在HTML5之前就已经存在了，是单独的规范；
* 它是将数据以数据库的形式存储在客户端，根据需求去读取；
* 跟Storage的区别是： Storage和Cookie都是以键值对的形式存在的；
* Web Sql 更方便于检索，允许sql语句查询；
* 让浏览器实现小型数据库存储功能；
* 这个数据库是集成在浏览器里面的，目前主流浏览器基本都已支持；

  websql API主要包含三个核心方法：

* openDatabase : 这个方法使用现有数据库或创建新数据库创建数据库对象。
* transaction : 这个方法允许我们根据情况控制事务提交或回滚。
* executeSql : 这个方法用于执行真实的SQL查询。

**openDatabase方法可以打开已经存在的数据库，不存在则创建**

1. var db = openDatabase('mydatabase', '2.0', my db', 2 \* 1024);

  openDatabasek中五个参数分别为：数据库名、版本号、描述、数据库大小、创建回调。创建回调没有也可以创建数据库。

**database.transaction() 函数用来查询，executeSql()用于执行sql语句**

  例如在mydatabase数据库中创建表t1：

1. var db = openDatabase(' mydatabase ', '1.0', 'Test DB', 2 \* 1024 \* 1024);
2. db.transaction(function (tx) {
3. tx.executeSql('CREATE TABLE IF NOT EXISTS t1 (id unique, log)');
4. });

  插入操作

1. var db = openDatabase('mydatabase', '2.0', my db', 2 \* 1024);
2. db.transaction(function (tx) {
3. tx.executeSql('CREATE TABLE IF NOT EXISTS t1 (id unique, log)');
4. tx.executeSql('INSERT INTO t1 (id, log) VALUES (1, "foobar")');
5. tx.executeSql('INSERT INTO t1 (id, log) VALUES (2, "logmsg")');
6. });

  在插入新记录时，我们还可以传递动态值，如：

1. var db = openDatabase(' mydatabase ', '2.0', 'my db', 2 \* 1024);
2. db.transaction(function (tx) {
3. tx.executeSql('CREATE TABLE IF NOT EXISTS t1 (id unique, log)');
4. tx.executeSql('INSERT INTO t1 (id,log) VALUES (?, ?'), [e\_id, e\_log]; //e\_id和e\_log是外部变量
5. });

  读操作，如果要读取已经存在的记录，我们使用一个回调捕获结果：

1. var db = openDatabase(mydatabase, '2.0', 'my db', 2\*1024);
2. db.transaction(function (tx) {
3. tx.executeSql('CREATE TABLE IF NOT EXISTS t1 (id unique, log)');
4. tx.executeSql('INSERT INTO t1 (id, log) VALUES (1, "foobar")');
5. tx.executeSql('INSERT INTO t1 (id, log) VALUES (2, "logmsg")');
6. });
7. db.transaction(function (tx) {
8. tx.executeSql('SELECT \* FROM t1, [], function (tx, results) {
9. var len = results.rows.length, i;
10. msg = "<p>Found rows: " + len + "</p>";
11. document.querySelector('#status').innerHTML += msg;
12. for (i = 0; i < len; i++){
13. alert(results.rows.item(i).log );
14. }
15. }, null);
16. });

**三、indexDB**

  IndexedDB 是一个为了能够在客户端存储可观数量的结构化数据，并且在这些数据上使用索引进行高性能检索的 API。虽然 DOM 存储 对于存储少量数据是非常有用的，但是它对大量结构化数据的存储就显得力不从心了。IndexedDB 则提供了这样的一个解决方案。  IndexedDB 分别为同步和异步访问提供了单独的 API 。同步 API 本来是要用于仅供 Web Workers 内部使用，但是还没有被任何浏览器所实现。异步 API 在 Web Workers 内部和外部都可以使用，另外浏览器可能对indexDB有50M大小的限制，一般用户保存大量用户数据并要求数据之间有搜索需要的场景。

* 异步API

  异步 API 方法调用完后会立即返回，而不会阻塞调用线程。要异步访问数据库，要调用 window 对象 indexedDB 属性的 open() 方法。该方法返回一个 IDBRequest 对象 (IDBOpenDBRequest)；异步操作通过在 IDBRequest 对象上触发事件来和调用程序进行通信。

1. - IDBFactory 提供了对数据库的访问。这是由全局对象 indexedDB 实现的接口，因而也是该 API 的入口。
2. - IDBCursor 遍历对象存储空间和索引。
3. - IDBCursorWithValue 遍历对象存储空间和索引并返回游标的当前值。
4. - IDBDatabase 表示到数据库的连接。只能通过这个连接来拿到一个数据库事务。
5. - IDBEnvironment 提供了到客户端数据库的访问。它由 window 对象实现。
6. - IDBIndex 提供了到索引元数据的访问。
7. - IDBKeyRange 定义键的范围。
8. - IDBObjectStore 表示一个对象存储空间。
9. - IDBOpenDBRequest 表示一个打开数据库的请求。
10. -IDBRequest 提供了到数据库异步请求结果和数据库的访问。这也是在你调用一个异步方法时所得到的。
11. - IDBTransaction 表示一个事务。你在数据库上创建一个事务，指定它的范围（例如你希望访问哪一个对象存储空间），并确定你希望的访问类型（只读或写入）。
12. - IDBVersionChangeEvent 表明数据库的版本号已经改变。

* 同步API

  规范里面还定义了 [API 的同步](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API)版本。同步 API 还没有在任何浏览器中得以实现。它原本是要和 [webWork](https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers) 一起使用的。

<http://mxr.mozilla.org/mozilla-central/source/modules/libpref/src/init/all.js><http://caniuse.com/#feat=indexeddb>

**四、cookie**

  Cookie（或者Cookies），指一般网站为了辨别用户身份、进行session跟踪而储存在用户本地终端上的数据（通常经过加密）。cookie一般通过http请求中在头部一起发送到服务器端。一条cookie记录主要由键、值、域、过期时间、大小组成，一般用户保存用户的认证信息。cookie最大长度和域名个数由不同浏览器决定，具体如下：

| **浏览器** | **支持域名个数** | **最大长度** |
| --- | --- | --- |
| IE7以上 | 50个 | 4095B |
| Firefox | 50个 | 4097B |
| Opera | 30个 | 4096B |
| Safari/WebKit | 无限制 | 4097B |

  不同域名之间的cookie信息是独立的，如果需要设置共享可以在服务器端设置cookie的path和domain来实现共享。浏览器端也可以通过document.cookie来获取cookie，并通过js浏览器端也可以方便地读取/设置cookie的值。

<https://github.com/component/cookie/blob/master/index.js>

**五、localstorage**

  localStorage是html5的一种新的本地缓存方案，目前用的比较多，一般用来存储ajax返回的数据，加快下次页面打开时的渲染速度。

| **浏览器** | **最大长度** |
| --- | --- |
| IE8以上 | 5M |
| Firefox 8以上 | 5.24M |
| Opera | 2M |
| Safari/WebKit | 2.6M |

1. //localStorage核心API:
2. localStorage.setItem(key, value) //设置记录
3. localStorage.getItem(key) //获取记录
4. localStorage.removeItem(key) //删除该域名下单条记录
5. localStorage.clear() //删除该域名下所有记录

  值得注意的是，localstorage大小有限制，不适合存放过多的数据，如果数据存放超过最大限制会报错，并移除最先保存的数据。

<https://github.com/machao/localStorage>

**六、sessionstorage**

  sessionStorage和localstorage类似，但是浏览器关闭则会全部删除，api和localstorage相同，实际项目中使用较少。

**七、application cache**

  application cahce是将大部分图片资源、js、css等静态资源放在manifest文件配置中。当页面打开时通过manifest文件来读取本地文件或是请求服务器文件。  离线访问对基于网络的应用而言越来越重要。虽然所有浏览器都有缓存机制，但它们并不可靠，也不一定总能起到预期的作用。HTML5 使用ApplicationCache 接口可以解决由离线带来的部分难题。前提是你需要访问的web页面至少被在线访问过一次。  使用缓存接口可为您的应用带来以下三个优势：

* 离线浏览 – 用户可在离线时浏览您的完整网站
* 速度 – 缓存资源为本地资源，因此加载速度较快。
* 服务器负载更少 – 浏览器只会从发生了更改的服务器下载资源。

  一个简单的离线页面主要包含以下几个部分：

**index.html**

1. <html manifest="clock.manifest">
2. <head>
3. <title>AppCache Test</title>
4. <link rel="stylesheet" href="clock.css">
5. <script src="clock.js"></script>
6. </head>
7. <body>
8. <p><output id="clock"></output></p>
9. <div id="log"></div>
10. </body>
11. </html>

**clock.manifest**

1. CACHE MANIFEST
2. #VERSION 1.0
3. CACHE:
4. clock.css
5. clock.js

clock.js和clock.css为独立的另外文件。  另外需要注意的是更新缓存。在程序中，你可以通过window.applicationCache 对象来访问浏览器的app cache。你可以查看 status 属性来获取cache的当前状态：

1. var appCache = window.applicationCache;
2. switch (appCache.status) {
3. case appCache.UNCACHED: // UNCACHED == 0
4. return 'UNCACHED';
5. break;
6. case appCache.IDLE: // IDLE == 1
7. return 'IDLE';
8. break;
9. case appCache.CHECKING: // CHECKING == 2
10. return 'CHECKING';
11. break;
12. case appCache.DOWNLOADING: // DOWNLOADING == 3
13. return 'DOWNLOADING';
14. break;
15. case appCache.UPDATEREADY: // UPDATEREADY == 4
16. return 'UPDATEREADY';
17. break;
18. case appCache.OBSOLETE: // OBSOLETE == 5
19. return 'OBSOLETE';
20. break;
21. default:
22. return 'UKNOWN CACHE STATUS';
23. break;
24. };

  为了通过编程更新cache，首先调用 applicationCache.update()。这将会试图更新用户的 cache（要求manifest文件已经改变）。最后，当 applicationCache.status 处于 UPDATEREADY 状态时， 调用applicationCache.swapCache()，旧的cache就会被置换成新的。

1. var appCache = window.applicationCache;
2. appCache.update(); // Attempt to update the user’s cache.
3. …
4. if (appCache.status == window.applicationCache.UPDATEREADY) {
5. appCache.swapCache(); // The fetch was successful, swap in the new cache.
6. }

  这里是通过更新menifest文件来控制其它文件更新的。

**八、cacheStorage**

  CacheStorage是在ServiceWorker的规范中定义的。CacheStorage 可以保存每个serverWorker申明的cache对象，cacheStorage有open、match、has、delete、keys五个核心方法，可以对cache对象的不同匹配进行不同的响应。

**cacheStorage.has()**如果包含cache对象，则返回一个promise对象。**cacheStorage.open()**打开一个cache对象，则返回一个promise对象。**cacheStorage.delete()**删除cache对象，成功则返回一个promise对象，否则返回false。**cacheStorage.keys()**含有keys中字符串的任意一个，则返回一个promise对象。**cacheStorage.delete()**匹配key中含有该字符串的cache对象，返回一个promise对象。

1. caches.has('v1').then(function() {
2. caches.open('v1').then(function(cache) {
3. return cache.addAll(myAssets);
4. });
5. }).catch(function() {
6. someCacheSetupfunction();
7. });;
8. var response;
9. var cachedResponse = caches.match(event.request).catch(function() {
10. return fetch(event.request);
11. }).then(function(r) {
12. response = r;
13. caches.open('v1').then(function(cache) {
14. cache.put(event.request, response);
15. });
16. return response.clone();
17. }).catch(function() {
18. return caches.match('/sw-test/gallery/myLittleVader.jpg');
19. });
20. then.addEventListener('activate', function(event) {
21. var cacheWhitelist = ['v2'];
23. event.waitUntil(
24. caches.keys().then(function(keyList) {
25. return Promise.all(keyList.map(function(key) {
26. if (cacheWhitelist.indexOf(key) === -1) {
27. return caches.delete(keyList[i]);
28. }
29. });
30. })
31. );
32. });

<https://developer.mozilla.org/en-US/docs/Web/API/CacheStorage>

**九、flash缓存**

  这种方式基本不用，这一方法主要基于flash有读写浏览器端本地目录的功能，同时也可以向js提供调用的api，则页面可以通过js调用flash去读写特定的磁盘目录，达到本地数据缓存的目的。

**注释PS**

* Web Storage / Web SQL Database / Indexed Database 的数据都存储在浏览器对应的用户配置文件目录(user profile directory)下，以 Windows 7 为例，Chrome 的数据存储在”C:\Users\your-account-name\AppData\Local\Google\Chrome\User Data\Default\”下，而 Firefox 的数据存储在”C:\Users\your-account-name\AppData\Local\Mozilla\Firefox\Profiles\”目录下。
* cookie文件存储于documents and settings\userName\cookie\文件夹下。通常的命名格式为：userName@domain.txt。
* 较多的缓存机制目前主流浏览器并不兼容，不过可以使用polyfill的方法来处理

**1、Cookie**

cookie是什么：cookie是指存储在用户本地终端上的数据，同时它是与具体的web页面或者站点相关的。cookie数据会自动在web浏览器和web服务器之间传输，也就是说HTTP请求发送时，会把保存在该请求域名下的所有cookie值发送给web服务器，因此服务器端脚本是可以读、写存储在客户端的cookie的操作。

cookie的有效期：cookie默认情况下的有效期是很短暂的，一旦用户关闭浏览器，cookie保存的数据就会丢失。如果想要延长cookie的有效期，可以通过设置HTTP头信息中的cache-control属性的max-age值，或者修改HTTP头信息中的expires属性的值来延长有效期。

cookie的作用域：它是通过文档源和文档路径来确定的。该作用域通过cookie的path和domain属性也是可配置的。默认情况下，cookie和创建它的web页面有关，并对该页面以及和该页面同目录或者子目录的其他页面可见。有时候，你可能希望让整个网站都能够使用cookie的值，而不管是哪个页面创建它。要满足这样的需求可以设置cookie的路径(设置cookie的path属性)。cookie的作用域默认由文档源限制。但是，有的大型网站想要子域之间能够互相共享cookie。

cookie的数目和大小的限制：每个web服务器(域名)保存的cookie数不能超过50个，每个cookie保存的数据不能超过4KB，如果超过了4KB（IE6大约只能存储2K），服务器会处理不了。

cookie的优点：能用于和服务器端通信；当cookie快要过期时，可以重新设置而不是删除。

cookie的缺点：它会随着http头信息一起发送，增加了网络流量（文档传输的负载）；它只能储存少量的数据；它只能储存字符串；有潜在的安全问题。

另外，自从有了Web Storage API(Local and Session Storage)，cookie就不被推荐用于存储数据了~

**2、LocalStorage**

localStorage是什么：在HTML5中，新加入了一个localStorage特性，这个特性主要是用来作为本地存储来使用的，解决了cookie存储空间不足的问题(cookie中每条cookie的存储空间为4k)，localStorage中一般浏览器支持的是5M大小，这个在不同的浏览器中localStorage会有所不同。

localStorage的优点：

1）localStorage拓展了cookie的4K限制；

2）localStorage会可以将第一次请求的数据直接存储到本地，这个相当于一个5M大小的针对于前端页面的数据库，相比于cookie可以节约带宽，但是这个却是只有在高版本的浏览器中才支持的；

3）localStorage 方法存储的数据没有时间限制。第二天、第二周或下一年之后，数据依然可用。

localStorage的缺点：

1）浏览器的大小不统一，并且在IE8以上的IE版本才支持localStorage这个属性；

2）目前所有的浏览器中都会把localStorage的值类型限定为string类型，这个在对我们日常比较常见的JSON对象类型需要一些转换；

3）localStorage在浏览器的隐私模式下面是不可读取的；

4）localStorage本质上是对字符串的读取，如果存储内容多的话会消耗内存空间，会导致页面变卡；

5）localStorage不能被爬虫抓取到。

**3、SessionStorage**

sessionStorage与localStorage的唯一一点区别就是localStorage属于永久性存储，而sessionStorage属于当会话结束的时候，sessionStorage中的键值对就会被清空。

**4、UserData、GlobalStorage、Google Gear**

这三种的使用都有一定的局限性，例如userData是IE浏览器专属，它的容量可以达到640K，这种方案可靠，不需要安装额外插件，只不过它仅在IE下有效~

globalStorage适用于Firefox 2+的浏览器，类似于IE的userData~

google gear是谷歌开发出的一种本地存储技术，需要安装Gear组件。

5、Flash ShareObject

这种方式能能解决上面提到的cookie存储的两个弊端，而且能够跨浏览器，应该说是目前最好的本地存储方案。不过，需要在页面中插入一个Flash，当浏览器没有安装Flash控件时就不能用了。所幸的是，没有安装Flash的用户极少。

**强调一下：cookie,localStorage和sessionStorage的异同：**

共同点：都是保存在浏览器端，且同源的。

区别：

1）cookie数据始终在同源的http请求中携带（即使不需要），即cookie在浏览器和服务器间来回传递；而sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存；

2）cookie数据有路径（path）的概念，可以限制cookie只属于某个路径下；

3）存储大小限制也不同，cookie数据不能超过4k，同时因为每次http请求都会携带cookie，所以cookie只适合保存很小的数据，如会话标识；sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大；

4）数据有效期不同，sessionStorage：仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持；localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie只在设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭；

5）作用域不同，sessionStorage不在不同的浏览器窗口中共享，即使是同一个页面；localStorage 在所有同源窗口中都是共享的；cookie也是在所有同源窗口中都是共享的；

6）Web Storage 支持事件通知机制，可以将数据更新的通知发送给监听者；Web Storage 的 api 接口使用更方便。