Internet технологии

ЛЕКЦИЯ №4
ХРАНЕНИЕ ДАННЫХ НА СТОРОНЕ КЛИЕНТА
HTML5 APPLICATION CACHE и WEBSTORAGE

Хранение данных на стороне клиента

- **Application Cache** позволяет держать копию HTML, CSS и других элементов нашего веб-приложения в автономном режиме, которые будут использоваться, когда сеть будет недоступна.
- **Web Storage** – программные методы и протоколы веб-приложения, используемые для хранения данных в веб-браузере.Постоянное хранилище данных, наподобие более гибкой реализации cookies.
- **WebSQL** реализует SQL-базу данных внутри вашего браузера, которая может хранить копии данных веб-приложения для автономной работы. Технология устарела.

Содержание

- Application Cache.
- Манифест кэша. Подключение и структура файла манифеста.
- Application Cache API.
- Способы обновления кэша.
- Преимущества и недостатки Application Cache.

- Web Storage
- Типы Web Storage
- Сохранение, извлечение, удаление данных из хранилищ
- Синхронизация данных

Application Cache

Использование кэширования дает приложению такие преимущества:

- доступ в режиме оффлайн – пользователи могут просматривать весь сайт в режиме офлайн;
- скорость – ресурсы хранятся локально, поэтому быстрее загружаются;
- уменьшение нагрузки на сервер – браузер загружает только требуемые ресурсы.

Использование Application Cache: при первом посещении пользователем сайта, страницы **запоминаются в кэше** (хранилище Application Cache) браузера. Затем при последующих посещениях, а также при потере соединения с Интернетом, используются **заранее сохраненные** данные из хранилища. **Контроль** над кэшированием сайта производится с помощью специального файла «.manifest».

Application Cache

Отличия от стандартного кэша браузера:

- кэш привязан к домену, а не к странице, где объявлен файл .manifest.
- Данные, помещённые в стандартный кэш могут быть **автоматически**, без команды со стороны пользователя или сервера, **удаленны** при его заполнении или истечения стока действия, указанного в заголовках файлов. Данные, помещённые в хранилище Application Cache могут быть удалены **только по команде** пользователя или сервера.
- В стандартный кэш браузера попадают только файлы, загруженные в процессе просмотра страницы. В Application Cache можно поместить **любые файлы**, загружаемые с сервера согласно **инструкции manifest**.

Application Cache. Подключение манифеста кэша

Файл **манифеста** кэша представляет собой простой текстовый файл со списком ресурсов, которые браузер должен кэшировать для доступа в автономном режиме. Любая страница с атрибутом manifest, которую посетит пользователь, будет неявным образом добавлена в кэш приложения.

```
<!DOCTYPE html>
<html lang="ru" manifest="cache.manifest">
```

Атрибут manifest может указывать на абсолютный URL или относительный путь, но абсолютный URL должен находиться в том же домене, что и веб-приложение.

Application Cache. Подключение манифеста кэша

В файле .htaccess на сервере необходимо дописать строчку, которая определяет новый тип файлов:

```
AddType text/cache-manifest .manifest
```

Специально для Mozilla FireFox нужно дописать конструкцию в .htaccess:

```
<IfModule mod_expires.c> ExpiresActive On ExpiresByType text/cache-manifest "access plus 0 seconds" </IfModule>
```

.htaccess (от. англ. hypertext access) — файл дополнительной конфигурации веб-сервера. Позволяет задавать большое количество дополнительных параметров и разрешений для работы веб-сервера в отдельных каталогах (папках), таких как управляемый доступ к каталогам, переназначение типов файлов и т.д.

Application Cache. Структура файла манифеста

Строка CACHE MANIFEST является обязательной и должна быть первой.

Комментарии в файле обозначаются с помощью символа #.

HTML-файл, который ссылается на ваш файл манифеста, кэшируется **автоматически**.

Указывать его в манифесте необязательно, но рекомендуется.

CACHE MANIFEST

CACHE:
script/library.js
css/stylesheet.css
images/figure1.png

FALLBACK:
photos/ figure2.png

NETWORK:
figure3.png

Application Cache. Структура файла манифеста

В файле manifest.cache три заголовка:

- **CACHE** – размещаются пути или URI к кэшируемым ресурсам. Указывается **конкретный ресурс** (файл), т.е. нельзя писать в этом разделе строку вида /images/*
- **NETWORK** – размещаются пути к файлам, которые обязательно должны **загружаться из интернета**. В этом разделе можно использовать паттерны, т.е. можно написать такую конструкцию:

```
CACHE MANIFEST
NETWORK:
*
```

- **FALLBACK** – резервные страницы на случай, если ресурс недоступен. Первый URL указывает ресурс, а второй – его резервную страницу. Оба адреса должны быть относительными

Application Cache. Разделы NETWORK и FALLBACK

- Все файлы, не обозначенные в разделах NETWORK и FALLBACK, и не имеющие сохранённых копий в хранилище Application Cache, загружаться **не будут**, даже если есть копии ресурсов в стандартном кэше браузера. Поэтому, если Вы не используете общие паттерны (* или /), то не забывайте указывать все файлы необходимые приложению, иначе вы не сможете работать с ними. Использовать общий паттерн достаточно в одном из разделов.
- Правила разделов NETWORK и FALLBACK **не перекрывают** правила раздела CACHE, то есть при работе без сети сначала грузятся кэшируемые данные, а только потом определяется политика в отношении ресурсов, указанных в этих двух разделах.
- Правила этих разделов не действуют на страницу, содержащую определение файла .manifest – она всегда **автоматически** кэшируется.

Application Cache API

Методы и свойства объекта applicationCache:
Обращение к объекту выглядит следующим образом:
window.applicationCache – обращение к объекту.
window.applicationCache.status – числовое значение соответствующее статусу состояния кэша. Возможны следующие статусы:

UNCACHED – кэш ещё не инициализирован (числовое значение 0);
IDLE – никаких действий с кэшем не производится (числовое значение 1);
CHECKING – производиться проверка файла .manifest (2);
DOWNLOADING – производится загрузка ресурсов в кэш (3);
UPDATEREADY – загрузка необходимых ресурсов выполнена и требуется их инициализация при помощи метода swapCache()(4);
OBSOLETE – текущий кэш является устаревшим (5).

Application Cache API

Методы и свойства объекта applicationCache:

window.applicationCache.update() – инициирует процесс проверки файла .manifest и последующие скачивание необходимых ресурсов.

window.applicationCache.swapCache() – переключает браузер на использование новых кэшированных файлов вместо старых. Перерисовки страницы не происходит, только при последующем обращении к кэшированным файлам они берутся уже из обновлённого кэша.

Простой альтернативой метода является перезагрузка страницы, например, при помощи location.reload().

Application Cache API

События объекта applicationCache:

checking – происходит при отправке запроса получения файла .manifest;

downloading – происходит при загрузке ресурсов в кэш;

cached – происходит при формировании первого кэша в хранилище;

progress – происходит при загрузке каждого ресурса по отдельности;

noupdate – происходит при подтверждение, что файл .manifest не обновился;

obsolete – происходит при подтверждение, что кэш в хранилище устарел и будет удалён;

error – произошла ошибка при обращении к файлам ресурсов или файлу .manifest;

updateready – происходит при окончании загрузки обновлённого кэша.

Application Cache API. Обновление кэша

Данные хранятся в кэше, пока не произойдет одно из перечисленных ниже событий:

- **Очистка хранилища данных** для соответствующего сайта в браузере.
- **Изменения в файле манифеста**. Обратите внимание: обновление файла, указанного в манифесте, не означает, что браузер повторно кэширует этот ресурс. Для этого должен измениться сам файл манифеста.
- **Программное обновление** кэша приложения. Чтобы обновить кэш программным образом, нужно прежде всего вызвать функцию applicationCache.update(). Она попытается обновить кэш пользователя (для этого необходимо, чтобы файл манифеста изменился).

Application Cache API. Обновление кэша

```
var appCache = window.applicationCache;
appCache.update(); // Attempt to update the user's cache.
...
if (appCache.status == window.applicationCache.UPDATEREADY) {
  appCache.swapCache();  // The fetch was successful, swap in the new cache.
}
```

После того как атрибут applicationCache.status перейдет в состояние UPDATEREADY, функция applicationCache.swapCache() заменит старый кэш на новый.

Проблемы с Mozilla Firefox.

Application Cache. Преимущества

- **Скорость** – файлы, закэшированные локально, загрузятся намного быстрее.
- **Снижение нагрузки на сервер**, так как вместо многочисленных запросов к ресурсам (имеются в виду кэшируемые на клиенте данные) для проверки их изменения, мы имеем всего лишь один запрос к файлу .manifest.
- Возможность кэшировать файлы по заранее определённым **правилам**.
- **Постоянное хранение** файлов, с жестким контролем изменения.

Application Cache. Недостатки

- при первой загрузке автоматически начинается **фоновая загрузка** кэшируемых документов.
- существует **проблема синхронизации** данных на сервере и клиенте. Изменение данных на сервере не приводит к изменению данных, хранящихся в кэше клиента. Необходимо инициировать процедуру обновления. До этого момента браузер будет загружать старые версии ресурсов.
- после обновления кэша сохранённые данные будут использоваться после перезагрузки страницы. Это приводит к тому, что даже с изменённым файлом .manifest первоначальная загрузка происходит с использованием **старого кэша**.
- размер кэшированных данных **ограничивается** 50 Мб. Для старых версий Mozilla Firefox и Google Chrome 5 Мб, у Opera размер возможного кэша можно увеличить в настройках.

Application Cache. Недостатки

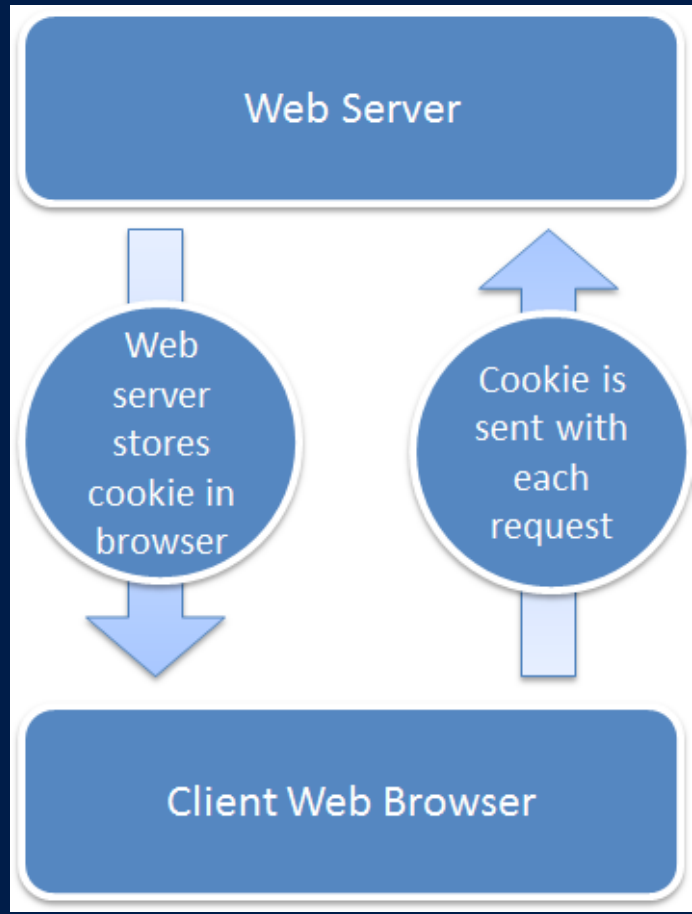
- можно привести приложение в **нерабочее состояние**, если не внести в разделы NETWORK и FALLBACK правила для файлов, необходимых для работы этого приложения.
- при использовании нескольких файлов .manifest можно создать ситуацию **цикличного кэширования** и привести приложение в нерабочее состояние.
- правила раздела FALLBACK для работы в режиме offline не перекрывают правила раздела CACHE. На практике это приводит к **усложнению структуры приложения** для работы в режиме offline.
- страница, содержащая определение файла .manifest, кэшируется автоматически. Что не позволяет легко её **заменить** в offline режиме.
- Application Cache **игнорирует настройки** сервера для обычного кэширования. Например, игнорируется Cache-Control: no-store.
- в старых версиях браузеров Application **Cache не работает**.

Ограничения HTTP COOKIE

На протяжении долгого времени cookies были единственным кроссбраузерным способом сохранить данные, которые будут доступны после перезагрузки страницы.

Однако у cookie есть важные особенности, например:

- cookie имеет **ограничение по размеру**, 4-10 килобайт данных;
- cookie участвуют в формировании каждого http-запроса к серверу, т.е. при каждом запросе все cookie автоматически отправляются вместе с запросом, что **увеличивает трафик**;
- cookie сопоставлены с web-сайтом и, если пользователь работает с сайтом через две вкладки, он оперирует **одними и теми же данными cookie**. Этот момент может нарушить правильную работу сайта и ограничивает применение cookie.



WebStorage. Преимущества

Механизм DOM Storage предлагает следующие возможности:

- **большой объем хранилища**: до 10 мегабайт в Internet Explorer для хранения данных для каждого сайта (ограничивается настройками браузера и вашим HDD, например, 5 Мбайт на домен в Mozilla Firefox, Google Chrome, и Opera);
- **высокая производительность** – доступ **только на стороне клиента**, данные DOM Storage не отправляются вместе с запросами;
- два механизма: **localStorage** (**не ограничено временем жизни**) и **sessionStorage** позволяют гибко управлять данными, контекст sessionStorage и его данные существуют только для одной вкладки и если пользователь закроет ее или откроет еще одну то, данные из вкладки доступны не будут.
- **простота в использовании для веб-разработчиков**: интерфейс представляет из себя ассоциативный массив модели данных, где ключи и значения являются строками.

WebStorage. Поддержка браузерами

Chrome	Firefox	Safari	Opera	Internet Explorer
4+	4+	4+	11+	8+

iOS	Android	Opera Mini	Opera Mobile
5+	3+	NA	11+

WebStorage. Проверка поддержки браузером

```
function isLocalStorageAvailable() {
  try {
    return window['localStorage'] !== null;
  } catch (e) {
    return false;
  }
}

if (isLocalStorageAvailable()) {
  // есть поддержка localStorage
} else {
  // нет поддержки localStorage
}
```

WebStorage. Типы Web-хранилищ

Существуют два основных типа веб-хранилища: локальное хранилище (**localStorage**) и сессионное хранилище (**sessionStorage**), ведущие себя аналогично постоянным и сессионным кукам соответственно.

Данные веб-хранилища хранятся только в активном браузере.

- localStorage – локальное хранилище, хранит данные без «срока годности». Эти данные будут доступны даже если вкладки браузера открыты или закрыты. Данные могут быть доступны между запросами страниц, при помощи нескольких вкладок, а также между сеансами браузера.

- sessionStorage – сессионное хранилище, хранит данные за один сеанс. Материалы данных будут очищены, как только пользователь закроет браузер.

WebStorage. Методы

Метод	Описание
setItem(key, value)	Добавляет пару ключ/значение к объекту Web Storage для дальнейшего использования. Значение может быть представлено любым типом данных: строка, число, массив и т.д.
getItem(key)	Возвращает значение по ключу, который был использован для первоначального сохранения.
clear()	Удаляет все пары ключ/значение для объекта Web Storage.
removeItem(key)	Удаляет отдельную пару ключ/значение из объекта Web Storage по ключу.
key(n)	Возвращает значение для ключа[n].

WebStorage. Сохранение строк

```
// добавление пары ключ/значение к объекту хранения
localStorage.setItem("myVar1", "abc");
// или как свойства объекта
localStorage.myVar2 = 123;
localStorage["myVar3"] = true;

// определение количества записей в хранилище:
var items = localStorage.length;
// определение названия ключа по его индексу:
var keyName = localStorage.key(index);

// ежесекундное сохранение значения текстового поля в сессионном
хранилище
setInterval(function(){ sessionStorage.setItem("autosave", field.value); }, 1000);
```

WebStorage. Получение строк

```
// получение пары ключ/значение в объекте WebStorage
var val1 = localStorage.getItem("myVar1");
alert(val1); // abc

// или как свойства объекта
var val2 = localStorage.myVar2;
 alert(val2); // 123

var val3 = localStorage["myVar3"];
alert(val3); // true
```

WebStorage. Сохранение и получение объекта

Сохранять можно **только строковые данные**. Для хранения массива или объекта вам необходимо будет использовать объект JSON, который позволит конвертировать данные в строку при помощи метода JSON.stringify. Для извлечения данных вы можете использовать JSON.parse, получив в результате массив или объект.

```
//сохранение объекта
var foo = {'1': [1, 2, 3]};
localStorage.setItem('formData', JSON.stringify(foo));
var fooFromLS = JSON.parse(localStorage.getItem('formData'));
//сохранение массива
var myArray = new Array('First Name', 'Last Name', 'Email Address');
localStorage.formData = JSON.stringify(myArray);
....
var myArray = JSON.parse(localStorage.formData);
```

WebStorage. Превышение размера хранилища

```
try {
   localStorage.setItem('foo', 'bar');
} catch (e) {
      if (e == QUOTA_EXCEEDED_ERR) {
      alert('Локальное хранилище переполнено');
   }
}
```

```
// удаление отдельной пары ключ/значение из объекта WebStorage
localStorage.removeItem("myVar1");
// или как свойства объекта
delete localStorage.myVar2;
// очистить всё хранилище
localStorage.clear();
```

WebStorage. Синхронизация данных

При выполнении операций с хранилищем, срабатывает событие **onstorage**. Функция обработки таких событий **storageHandler()** назначается в HTML как:

```
<body onstorage="storageHandler()">
```

или в Javascript:

```
window.addEventListener("storage", storageHandler, false);
```

как анонимная функция:

```
window.addEventListener('storage', function(event) {
    console.log('The value for ' + event.key + ' was changed from' +
event.oldValue + ' to ' + event.newValue);
}, false)
```

WebStorage. Синхронизация данных

В функции обработки объект события предоставляет нам такие свойства, связанные с хранилищем:

```
function storageHandler(event) {
  var key = event.key; // ключ изменяемых данных
  var oldValue = event.oldValue; // старое значение
  var newValue = event.newValue; // новое значение
  var win = event.window;
  var url = event.url;
  var storageArea = event.storageArea;
}
```

Итог пакета лекций:

- PDO и MySQLi.
- соединение с БД, выполнение основных запросов.
- выборка строк из запроса, режимы выборки.
- подготовленные запросы, использование именованных и неименованых псевдопеременных.
- обработка транзакций.
- особенности NoSQL-решений.
- Application Cache
- WebStorage

Вопросы

- Каковы особенности NoSQL-подхода?
- В чем состоит преимущество хранения данных на клиентской стороне?
- Какие инструменты хранения данных на стороне клиента Вам известны?
- Отличия Application Cache от стандартного кэша браузера.
- Что такое манифест кэша? Структура файла манифеста.
- Какие события Application Cache происходят при первоначальной загрузке документа?
- В каких случаях происходит обновление Application Cache?
- Преимущества и недостатки Application Cache.
- Ограничения механизма HTTP COOKIE и преимущества использования WebStorage.
- Опишите особенности использования LocalStorage и SessionStorage.
- Как определить превышение размера локального хранилища?