Работа с файлами в JavaScript, Часть 1: Основы

Категория: JavaScript Опубликовано: 26 июня 2012 Просмотров: 64074

ajax

html5

file api

drag and drop

file

filelist

formdata

При всем бурном развитии web, и стандартов html в частности, работа с файлами, практически никогда не менялась. К счастью, с приходом HTML5 и связанных с ним API, сейчас у нас гораздо больше возможностей для работы с файлами, чем когда-либо в предыдущих версиях браузеров(iOS до сих пор нет поддержки File API).

**Тип Файл - File**

Тип File определен в спецификации File API[1] и является абстрактным представлением файла. Каждый экземпляр File имеет следующие свойства:

name – имя файла

size – размер файла в байтах

type – MIME тип файла

Объект типа File дает важную информацию о файле, не предоставляя прямой доступ к содержимому файла. Он является лишь ссылкой на файл, и получение данных из этого файла является отдельным процессом в целом.

**Получение ссылок на файлы**

Разумеется, доступ к пользовательским файлам строго запрещен в Интернете, потому как очевидны проблемы с безопасностью личных данных. Вы не хотели бы, чтобы Вы загружали веб-страницу, а затем она сканировала Ваш жесткий диск и выясняла, что там есть полезного. Нужно разрешение от пользователя, чтобы получить доступ к файлам с его компьютера. Тем не менее для веб-страниц чтения файлов разрешено каждый раз, когда пользователь решат что-то загрузить.

Когда вы используете элемент <input type="file">, Вы даете веб странице (и серверу) разрешение на доступ к файлу. Так, что первое, как вы можете получить объект File, это поле <input type="file">.

HTML5 определяет файловые ссылки для всех <input type="file"> управления. Эта коллекция FileList, которая представляет собой структуру в виде массива под названием FileList содержащую объекты типа File для каждого выбранного файла в поле <input type="file"> (помните, HTML5 позволяет выбрать несколько файлов в этом элементе управления). Так что в любой момент времени, Вы можете получить доступ к файлам пользователя, которые он выбрал, с помощью кода вроде этого:

<input type="file" id="your-files" multiple>

и

<script>

var control = document.getElementById("your-files");

control.addEventListener("change", function(event) {

// Когда происходит изменение элементов управления, значит появились новые файлы

var i = 0,

files = control.files,

len = files.length;

for (; i < len; i++) {

console.log("Filename: " + files[i].name);

console.log("Type: " + files[i].type);

console.log("Size: " + files[i].size + " bytes");

}

}, false);

</script>

Этот сравнительно простой код ожидает событие изменения в контроле(<input type="file"> ). Когда событие происходит, это означает, что выбор файла изменился, и код перебирает все объекты типа File и выводит информацию из них. Имейте в виду, что свойство файлов всегда доступны из JavaScript, так что вам не придется ждать следующего изменения, чтобы попытаться сделать что-то другое с ними.

**Drag and drop файлов**

Доступ к файлам из формы по средствам контролов по-прежнему требует действий пользователей: нахождение и выбора интересующего файла. К счастью, HTML5 Drag and Drop предоставляет еще один способ для пользователей, чтобы предоставить доступ к своим файлам: путем простого перетаскивания файлов с рабочего стола в веб-браузер. Все, что вам нужно сделать, чтобы это реализовать отслеживать два события.

Для того, чтобы читать файлы, которые упали на элемент страницы, вы должны отслеживать события DragOver и Drop, и отменять действия по умолчанию, в обоих. Это говорит браузеру, что вы знаете, что делаете :) и отменяет стандартные действия в таких случаях. Например, когда Вы перетаскиваете на страницу файл изображения, стандартным действием в таком случае будет открытие этого файла в этой вкладке. Это действие нужно отменить.

<div id="your-files"></div>

<script>

var target = document.getElementById("your-files");

target.addEventListener("dragover", function(event) {

event.preventDefault(); // отменяем действие по умолчанию

}, false);

target.addEventListener("drop", function(event) {

// отменяем действие по умолчанию

event.preventDefault();

var i = 0,

files = event.dataTransfer.files,

len = files.length;

for (; i < len; i++) {

console.log("Filename: " + files[i].name);

console.log("Type: " + files[i].type);

console.log("Size: " + files[i].size + " bytes");

}

}, false);

</script>

event.dataTransfer.files другой FileList объект, через который вы можете получить доступ, к информации о файлах. Код почти такой же, как и контролами формы и объекты типа File могут быть доступны таким же образом.

**AJAX pагрузка файлов**

Если у вас есть ссылка на файл, то вы сможете сделать очень удобную вещь: загрузить файл с помощью Ajax. Все это возможно благодаря объекту FormData, которая определен в XMLHttpRequest . Этот объект представляет собой HTML-форму и позволяет добавлять пары ключ-значение, которые будут переданы на сервер с помощью метода append():

var form = new FormData();

form.append("name", "Николай");

Самое замечательное в объекте FormData, что вы можете добавить файл непосредственно к нему, фактически имитируя загрузку файла через HTML-форму. Все, что вам нужно сделать, это добавить в файл ссылки с определенным именем, и браузер сделает все остальное. Для примера:

// Создаем форму с несколькими значениями

var form = new FormData();

form.append("name", "Николай");

form.append("photo", control.files[0]);

// отправляем через xhr

var xhr = new XMLHttpRequest();

xhr.onload = function() {

console.log("Отправка завершена");

};

xhr.open("post", "/entrypoint", true);

xhr.send(form);

Как только объект FormData передается в send (), надлежащие к его содержимому HTTP заголовки устанавливаются автоматически. Вам не нужно беспокоиться об установке правильной кодировки формы при использовании файлов, сервер будет работать с полученными файлами, так как если бы была отправлена​​ обычная HTML форма , читая данные о присланном файле из $\_FILES['photo'] и текстовыt данных из $\_POST['name']. Это дает вам универсальность, чтобы написать код обработки на стороне сервера, который может легко работать как с традиционными HTML-формами так и с формами присланными через Ajax.

И все это работает на последней версии большинства браузеров, включая Internet Explorer 10. К сожалению Internet Explorer 9 этого пока не поддерживает.

**Что дальше**

Теперь вы знаете два способа доступа к информации о файле в браузере: через контрол формы и через нативный 'drag and drop'. Вероятно, появятся и другие способы доступа к файлам в будущем, но сейчас Вам нужно знать только эти два. Конечно, читать информацию о файлах, это только часть проблемы.Следующим шагом будет чтение данных из этих файлов, об этом и напишу во второй части статьи.

**Пример записи и чтения файла (или нескольких):**

HTML:

<input id="input" type="file">

<hr>

<textarea id="textarea"></textarea>

<hr>

<a id="download" href="">Скачать файл</a>

JavaScript:

var inputElement = document.getElementById('input'),

downloadElement = document.getElementById('download'),

textareaElement = document.getElementById('textarea');

downloadElement.style.display = 'none';

inputElement.onchange = function() {

var fileList = this.files, // список указанных файлов

textFile = fileList[0]; // для нашего примера берём один файл

// Проверяем тип файла (текстовой файл)

if (textFile.type == 'text/plain') {

// Создаём новый FileReader, который и будет читать наш файл

var reader = new FileReader();

// Событие успешного чтения

reader.onloadend = function(event) {

var text = event.target.result,

blob = new Blob([text], { type: 'text/plain' });

// Ваши любые манипуляции с данными

textareaElement.value = text;

downloadElement.style.display = 'block';

downloadElement.href = URL.createObjectURL(blob);

downloadElement.download = textFile.name;

};

// Событие ошибки

reader.onerror = function() {

alert('Ошибка чтения файла!');

};

// Читаем наш файл как текст

reader.readAsText(textFile);

} else {

alert('Это не текстовой файл!');

}

};

textareaElement.onkeyup = function() {

// Обновляем ссылку скачивания файла

// (ТОЛЬКО ДЛЯ ПРИМЕРА, СКРИПТ ВЫПОЛНЯЕТСЯ ПРИ КАЖДОМ НАЖАТИИ КЛАВИШИ)

var text = this.value,

blob = new Blob([text], { type: 'text/plain' });

downloadElement.href = URL.createObjectURL(blob);

};

Существует множество задач, связанных с созданием, парсингом, сохранением и удалением файлов. Большинство из них выполняется на стороне сервера. Прежде всего потому, что Java, C# или любой другой язык, который вы используете на сервере, может легко справиться со всеми этими задачами.

Однако, в этом случае существует большая проблема: у всех ваших пользователей имеется в распоряжении только один сервер, но каждый из них хочет выполнить какие-то действия. Это может вызвать увеличение расхода ресурсов сервера или даже вывести ваш сервер из строя из-за огромного количества запросов.

Вторая причина, по которой вы все еще выполняете вышеуказанные действия на сервере, заключается в том, что вы просто еще не прочитали данную статью и не знаете всех возможностей современных браузеров. Не волнуйтесь! Мы откроем вам все их секреты и тайны.

Данную тему можно разделить на три части: прошлое, настоящее и будущее.

**Прошлое. ActiveX**

В прошлом существовало только два браузера: Netscape и IE. Однажды Microsoft решил порадовать пользователей IE и добавил невероятную для того времени функциональность, которая позволяла коду, написанному на языке JavaScript, обмениваться информацией с операционной системой Windows. Данный способ получил название ActiveX.

Описанная технология может применяться даже сегодня в последней версии браузера IE (по умолчанию, ActiveX отключен). Не будем долго останавливаться на этой теме, поскольку ActiveX актуален только для IE, и пользователю необходимо проделать много необычных действий со своим браузером для того, чтобы запустить сценарий, который содержит ActiveXObjects.

В разделе, посвященном настоящему времени, мы поговорим об API для работы с файлами и Drag-and-Drop, а также о некоторых интересных примерах.

Самая же увлекательная часть содержится в разделе о “будущем”, где будут раскрыты все секреты файловой системы в браузере. Давайте приступим к изучению возможностей браузеров.

**Настоящее. Управление добавленными файлами**

Существует два способа передать файл в современный браузер (IE10+):

1) Input тег с типом “file”;

2) Перетянуть файлы на какой-либо элемент DOM-модели.

**Старые добрые времена с элементом Input**

Первый вариант может использоваться даже в старых браузерах. Единственное отличие в том, что мы не можем получить содержимое файлов в IE9 и более ранних версий. Это означает, что для выполнения некоторых действий нам все еще необходимо использовать сервер. Чтобы получить файлы, выбранные пользователем, нам нужно добавить обработчик события “change” и проверять свойство “files” данного элемента при вызове функции-обработчика.

var template = "<div><span>Name: </span><span>{{Name}}</span></div><div><span>Size: </span><span>{{Size}}</span></div><div><span>Data: </span><span>{{Data}}</span></div>",

data = document.getElementById("fileData");

document.getElementById("fileElement").addEventListener("change", function(e){

var file = this.files ? this.files[0] : {

name: this.value

},

fileReader = window.FileReader ? new FileReader() : null;

if (file){

if (fileReader){

fileReader.addEventListener("loadend", function(e){

data.innerHTML = template.replace("{{Name}}", file.name).replace("{{Size}}", file.size).replace("{{Data}}", e.target.result.substring(0, 10));

}, false);

fileReader.readAsText(file);

} else {

data.innerHTML = template.replace("{{Name}}", file.name).replace("{{Size}}", "Don't know").replace("{{Data}}", "This browser isn't smart enough!");

}

}

}, false);

Как видите, здесь мы использовали метод “readAsText” объекта “FileReader”, чтобы получить содержимое файла. У него также есть методы, которые читают файл(файлы) и возвращают результаты в различных форматах.

Вы можете использовать метод „readAsArrayBuffer“, который читает файл, и возвращает файлы в виде массива данных в двоичном формате; „readAsDataURL“, который возвращает URL в виде “data:image/png,base64:”. Вы можете поместить этот URL в атрибуте src тега IMG, например (смотрите демо #2).

Будьте внимательны, используя метод “readAsBinaryString”, поскольку он не поддерживается браузером IE версий 10-11.

Возможно, вы также хотите контролировать процесс чтения файла, отобразить индикаторы прогресса или ожидания, или прервать его, если файлы слишком объемные. Специально для вас у FileReader есть такие события как onprogress, onabort, onloadstart, onloadend, onerror, а также метод abort, который останавливает процесс чтения и выдает ошибку, которая может быть обработана.

В данном примере мы немного исправили функциональность для IE9, у которого нет возможности чтения файлов. Из информации о выбранном файле (множественный выбор не поддерживается) мы можем получить только его имя, которое хранится в свойстве “value” элемента input.

Выберите картинку:

fileReader.addEventListener("loadend", function(e){

img.src = e.target.result;

}, false);

fileReader.readAsDataURL(file);

**Перетащи меня в браузер!**

Это был старый способ добавления файлов в браузер. Он мог использоваться даже в IE6 (addEventListener поддерживается в IE9+), с некоторыми изменениями. В современных браузерах (IE10+) существует другой способ добавления файлов. Все, что нужно сделать, это повесить обработчик события «drop” на любой элемент веб-страницы. Это может быть как специальный элемент с текстом внутри, так и просто body. После этого пользователь может перетащить какой-либо файл в элемент, на котором висит обработчик события “drop”. Выбранный файл можно найти в свойстве “dataTransfer” объекта события. Давайте посмотрим на демо:

**Перетащите свой файл сюда**

var template = "<div><span>Name: </span><span>{{Name}}</span></div><div><span>Size: </span><span>{{Size}}</span></div><div><span>Data: </span><span>{{Data}}</span></div>",

data = document.getElementById("fileData"),

preventDefault = function(e){

e.preventDefault();

},

highlight = {

add: function(e){

preventDefault(e);

e.target.classList.add("hoverClass");

},

remove: function(e){

preventDefault(e);

e.target.classList.remove("hoverClass");

}

},

uploader = document.getElementById("fileElement");

if (window.FileReader){

uploader.addEventListener("dragover", highlight.add, false);

uploader.addEventListener("mouseout", highlight.remove, false);

uploader.addEventListener("drop", function(e){

var file = e.dataTransfer.files[0],

fileReader = new FileReader();

if (file) {

if (fileReader){

fileReader.addEventListener("loadend", function(e){

data.innerHTML = template.replace("{{Name}}", file.name).replace("{{Size}}", file.size).replace("{{Data}}", e.target.result.substring(0, 10));

}, false);

fileReader.readAsText(file);

} else {

alert("Sorry, but this browser isn't smart enough( Choose another one");

}

}

preventDefault(e);

}, false);

} else {

alert("Sorry, but this browser isn't smart enough( Choose another one");

}

Несколько важных замечаний по этому демо:

1) Прежде всего, вам не стоит проверять, поддерживает ли ваш браузер событие “ondrop”, поскольку используется IE9. Он поддерживает данное событие, но не имеет свойства dataTransfer.

2) Если вы хотите поймать событие “ondrop”, вам необходимо добавить обработчик события “ondragover” и вызвать функцию preventDefault. По умолчанию, браузер попытается загрузить этот файл или отобразить его на странице (если это рисунок или PDF файл), и вам нужно его остановить.

Поддержка старых версий браузеров: в таких браузерах, как IE9, вы можете поместить элемент input из предыдущих примеров на текущем элементе, растянуть его, установить для него прозрачность равную 100%, и изменить текст на следующий: “Кликните и выберите файл”. Когда пользователь кликнет на этот элемент, он увидит обычный диалог для выбора файлов. Подобное решение подходит также для планшетов и телефонов, у которых нет возможности разделить экраны и перетягивать элементы между ними.

**Перетащи меня из браузера!**

Предыдущий пример мог вызвать следующий вопрос о функциональности drag-and-drop: “Можем ли мы перетащить файлы из браузера на компьютер?” Ответ будет “Да”, но работает это только в браузере Chrome. Тем не менее, это отличная функция для вашей веб страницы, и мы должны ее испробовать. В следующем примере вам нужно перетянуть картинку в формате PDF на свой компьютер, и она будет сохранена как PDF файл.

Перетащите эту картинку из браузера на ваш рабочий стол

document.getElementById("fileData").addEventListener("dragstart", function(e){

e.dataTransfer.setData("DownloadURL", "application/pdf:doc.pdf:file:///C:/personal/publications/demo/doc.pdf");

}, false);

Код в данном примере достаточно прост. Все, что вам нужно сделать — это добавить обработчик события “dragstart” и задать путь к данным, которые необходимо переместить. Это может быть ссылка на какой-либо документ на сервере или двоичные данные. Если вы хотите получить файл с сервера, нужно передать специальную строку, состоящую из типа содержимого, имени файла, который будет сохранен, и полного пути к нему. Например: “application/pdf:newDoc.pdf:https://mywebsite.com”.

Как уже было сказано, перетащить файл можно только из браузера Chrome. Однако, проблемой это не является, поскольку это не основная функциональность вашего веб-приложения, а просто отличное дополнение для пользователей Chrome, которая может быть благополучно заменена обычной ссылкой для загрузки нужного файла.

“Все течет, все изменяется … и … дважды в одну и ту же реку войти невозможно”. (Гераклит)

Теперь мы можем получить файлы и их содержимое в формате JavaScript. Но что мы можем с ними делать? Как мы можем их изменить? Вот в чем вопрос.

Прежде всего, мы можем работать с текстом. Когда мы прочитаем файл как текстовую строку, мы получим все его текстовое содержимое и сможем его парсить. Например, пользователь может перетянуть новый CSS файл, и вы сможете сразу применить его на своей странице. Или, если это файл со статьей, которую нужно опубликовать, вы можете его прочитать, применить какие-либо стили к заголовкам и поместить его в текстовый редактор на вашей странице.

Примером также может быть парсинг файла в формате excel. Вам не нужно посылать его на сервер, конвертировать в JSON и отправлять обратно. Это можно сделать на клиентской стороне, не тратя времени на запрос и ожидание ответа сервера. Уже есть библиотеки для парсинга xls и xlsx файлов, которые вы можете найти здесь: XLS и XLSX.

Еще один способ использования файлов и их содержимого в JavaScript — это работа с двоичными данными. Это могут быть рисунки, видео или какие-либо другие файлы. Если мы хотим обрабатывать двоичные данные, нам нужно использовать новые свойства JavaScript, такие как объект Blob и типизированные массивы. Мы не будем углубляться в эту тему, потому что она слишком большая и заслуживает отдельной статьи.

Объект Blob — это объект, который выглядит, как обычный файл на JavaScript, и содержит необработанные данные. Конструктор такого объекта принимает два параметра — данные и объект с его опциями. В состав данных могут входить ArrayBuffer, типизированный массив, объект Blob или строки. Это все те данные, которые будут храниться в новом объекте Blob. Опции включают в себя тип данных.1 new Blob(["<div>Hello world</div>"], {type: "text/html"})

У каждого объекта Blob есть тип и свойства, которые задают его размеры, а также метод “slice”. Все данные, которые содержатся в данном объекте Blob, можно разделить с помощью метода “slice”. Этот метод возвращает новый объект Blob, который содержит в себе отделенную часть данных.

Типизированные массивы и буферы используются для хранения двоичных данных фиксированной длины и определенного типа (Int8Array или Int32Array).

Все это позволяет нам делать занимательные вещи на клиентской стороне, например, сжимать файлы. Эта библиотека позволяет применить алгоритм Deflate ко всем выбранным файлам, и собрать их в один zip файл. Кроме того, она использует Web Workers, чтобы описанный процесс протекал в фоновом режиме.

Еще один пример обработки файла, содержащего двоичные данные, это PDF.js. Это проект, который находится в открытом доступе и используется для отображения на веб-странице документов в формате PDF. Все, что вам нужно сделать, это передать ему массив двоичных данных, и он отрисует содержимое PDF-документа на странице.

Будущее. Создайте свою файловую систему с блэкджеком и… всеми причитающимися атрибутами

Теперь мы знаем, как получать файлы и изменять их. Настало время раскрыть секрет хранения файлов. Да-да, речь идет именно о создании и хранении папок и файлов внутри браузера.

Во время написания этой статьи данная функциональность поддерживалась только браузерами Chrome и Opera. Опять-таки, это не такая уж важная проблема, поскольку данная возможность может использоваться как дополнительный инструмент для кэширования файлов в вашем браузере. Давайте более детально рассмотрим ее API.

Прежде всего, нам нужно запросить объект файловой системы с помощью метода webkitRequestFileSystem. Он принимает 4 параметра: тип, размер, функция, которая будет вызвана в случае успешной обработки запроса, и функция, которая будет вызвана при возникновении ошибки.

В браузере имеются два типа файловой системы: временная и постоянная. Первая может быть создана без запроса разрешения пользователя. Вы просто ее создаете и затем используете. Однако, если ваш браузер столкнется с какой-либо проблемой, как, например, недостаток памяти, он удалит все хранящиеся в нем данные. Поэтому данный тип хранения информации может быть использован только для кэширования данных, которые можно восстановить.

Еще один тип файловой системы — постоянный. Браузер никогда не удалит файлы, хранящиеся в ней. Тем не менее, если вы запросите данный тип файловой системы без разрешения пользователя, она не будет работать.

Для того, чтобы запросить разрешение пользователя на использование постоянного хранилища, нам нужно вызвать метод navigator.webkitPersistentStorage.requestQuota.

Константы обоих типов файловых систем хранятся в объекте window: window.TEMPORARY или window.PERSISTENT. Второй параметр — это нужный нам размер хранилища в байтах.

Теперь мы можем создать наше первое хранилище вместимостью 1Гб:

Временная файловая система:

window.webkitRequestFileSystem(window.TEMPORARY, 1000\*1024\*1024, function(fs){

alert("Storage is ready");

}, errorHandler);

**Постоянная файловая система:**

navigator.webkitPersistentStorage.requestQuota(1000\*1024\*1024, function(bytes) {

window.webkitRequestFileSystem(window.PERSISTENT, bytes, function(fs){

alert("storage is ready");

}, errorHandler);

}, errorHandler);

Как видите, мы используем много обработчиков ошибок, которые должны отлавливать ошибки, связанные с созданием хранилищ. Объект FileError содержит полный список типов ошибок:

Итак, теперь у нас есть доступ к хранилищу, и мы можем создать нашу первую папку и файл, и сохранить их. После того, как запрос на использование постоянного хранилища (так же, как и в случае с временным) одобрен, мы получаем объект файловой системы в качестве первого аргумента функции, возвращаемой при успешном выполнении запроса.

**Данный объект имеет интерфейс следующего вида:**

Как видите, для создания, перемещения, удаления и получения файлов и директорий используются специальные методы. Полученный нами объект “fs” ссылается на корневой каталог файловой системы браузера.

Данный каталог и все другие каталоги, которые мы будем создавать, представлены интерфейсом DirectoryEntry. Все файлы данной файловой системы представлены интерфейсом FileEntry. Мы будем использовать методы getFile и getDirectory для того, чтобы создать файлы и папки и получить их.

Первый параметр данных методов — это файл или путь к нему. Этот путь может быть как абсолютным, так и относительным. Второй параметр — это объект с действиями, которые выполняются в случае, если в указанной директории файл отсутствует. Данный параметр может иметь свойства “create” и “exclusive”.

Если свойству “create” задано значение “true”, будет создан новый файл или директория, а старый файл будет переписан. Если же мы для данного свойства зададим значение “false”, методы вернут объект FileEntry или выдадут ошибку, если файла не существует. Если задано свойство “exclusive” со значением “true”, файлы не будут переписаны.

Для того, чтобы различить файлы и папки, у каждого из двух интерфейсов есть флаги: isDirectory и isFile. Кроме того, существует специальный класс FileWriter, который служит для добавления содержимого в файл. Его необходимо запрашивать у браузера для каждого файла, который нужно изменить. Давайте добавим несколько файлов:

// создадим папку “files” в корневом каталоге

fs.root.getDirectory("files", {create: true}, function(directory){

//в папке "files" создадим файл “newFile.txt”

directory.getFile('newFile.txt', {create: true}, function(file) {

//добавим текст в файл “newFile.txt”

file.createWriter(function(writer) {

writer.onwriteend = function(e) {

alert("done");

};

// создадим новый объект Blob и запишем его в log.txt.

var blob = new Blob(['hello world!!!!'], {type: 'text/plain'});

writer.write(blob);

});

}, errorHandler);

});

Все прошло успешно и, вроде бы, без ошибок. Давайте проверим наши файл и папку. Необходимо запустить некоторые экспериментальные функции в DevTool браузера Chrome, чтобы увидеть содержимое файловой системы браузера. Поэтому, вам нужно перейти на страницу “chrome://flags/#enable-devtools-experiments”. После перезагрузки Chrome, в настройках DevTool вы увидите вкладку “Эксперименты”. Здесь вам нужно включить опцию “File System inspection” и перезапустить DevTool.

После этого вы увидите опцию “FileSystem” на вкладке “Resources” раздела DevTool. Здесь отображаются все созданные файлы и хранилища.

Данные файлы мы также можем получить по специальному внутреннему URL браузера, который может быть установлен в качестве атрибута “src” тега IMG. Он начинается с слова “filesystem:” и выглядит вот так: “filesystem:{URL of site}/persistent/files/newFile.txt”.

Чтобы получить URL, мы использовали метод “toURL”.

Теперь у нас есть вся необходимая информация для того, чтобы создать веб-приложение для кэширования аудио и видео файлов, которое позволяет слушать и смотреть их без подключения к интернету. Прежде всего, загрузим двоичный файл. Затем, мы сохраним его в постоянном хранилище и запустим.

Частная коллекция качественных материалов для тех, кто делает сайты

**HTML5 File API для взаимодействия с файлами в браузере**

HTML5 File API для взаимодействия с файлами в браузере

Недавно я столкнулся с определённой задачей: необходимо, чтобы пользователь загрузил файл через браузер, сделал несколько преобразований и затем сохранил результат.

Данную задачу можно было бы реализовать старым способом:

Загрузить изображение через AJAX

Рендеринг загруженного изображения в браузере

Преобразуем изображение через JavaScript

Делаем дополнительный запрос для преобразования изображения и

Сохраняем преобразованное изображение на сервере

Слишком много серверных операций. Данный способ не очень эффективен, так что мы нашли другой способ - HTML5 File API.

**Что такое HTML5 File API?**

HTML5 File API позволяет локально преобразовывать файлы, без предварительной загрузки на сервер.

**3 основных HTML5 File объекта**

Для работы с данным API вам нужно ознакомиться с тремя объектами:

File – простой объект файла с некоторыми метаданными.

FileList – список из объектов типа файл.

FileReader – объект для считывания файла и некоторыми метода обработки.

**Получение доступа к файлу через JavaScript**

Список файлов можно получить после их выбора через HTML элемент input. Вот небольшой пример по взаимодействию с загружаемым файлом. Мы воспользуемся console.log() для вывода промежуточной информации.

Выбор одного файла

1 // отслеживаем использование элемента input с id = “the-file-input”

2 $("#the-file-input").change(function() {

3 // выводим объект FileList

4 console.log(this.files);

5 });

1 <!-- получаем доступ через input -->

2 <input id="the-file-input" type="file">

Выбор нескольких файлов

1 <!-- для выбора нескольких файлов, добавляем атрибут multiple -->

2 <input id="the-file-input" type="file" multiple>

**Рендеринг файла в браузере**

Теперь когда мы знаем как получить доступ к объекту FileList, осталось узнать как осуществить рендеринг изображения в браузере. Для этого нам нужно скормить объект File, объекту FileReader для генерации локального url, который будет доступен через атрибут src элемента image.

01 // рендеринг изображения

02 function renderImage(file) {

03

04 // генерация нового объекта FileReader

05 var reader = new FileReader();

06

07 // подстановка изображения в атрибут src

08 reader.onload = function(event) {

09 the\_url = event.target.result

10 $('#some\_container\_div').html("<img src="" + the\_url + "" alt="">")

11 }

12

13 // при считке файла, вызывается метод, описанный выше

14 reader.readAsDataURL(file);

15 }

16

17 // обработка элемента input

18 $("#the-file-input").change(function() {

19 console.log(this.files)

20

21 // выбор первого изображения из FileList и передача в функцию

22 renderImage(this.files[0])

23 });

В данном примере мы продемонстрировали как можно отобразить изображение в браузере без загрузки на сервер! Это отличный вариант, особенно если на нашем сайте огромное количество пользователей, который совершают данную операцию.

**Заключение**

Метод, который мы показали может снизить нагрузку на сервер. Так же это новый полигон для создания приложений по обработке фото.

**Работа с файлами**

Где ваш веб-сайт должен располагаться на вашем компьютере?

Небольшое отступление о регистре и пробелах

Какую структуру должен иметь ваш веб-сайт?

Путь к файлам

Что должно быть сделано?

 Назад

 Обзор: Getting started with the web

Далее 

Веб-сайт состоит из множества файлов: текстового контента, кода, стилей, медиа-контента, и так далее. Когда вы создаете веб-сайт, вы должны собрать эти файлы в рациональную структуру на вашем локальном компьютере, убедитесь, что они могут общаться друг с другом, и весь ваш контент выглядит правильно, прежде чем вы, в конечном итоге загрузите их на сервер. Работая с файлами, обсуждайте некоторые вопросы, о которых вы должны быть в курсе, чтобы вы могли рационально настроить файловую структуру для вашего веб-сайта.

**Где ваш веб-сайт должен располагаться на вашем компьютере?**

Edit

Когда вы работаете на веб-сайте локально на вашем компьютере, вы должны держать все связанные файлы в одной папке, которая отражает файловую структуру опубликованного веб-сайта на сервере. Эта папка может располагаться где угодно, но вы должны положить её туда, где вы сможете легко её найти, может быть, на вашем рабочем столе, в домашней папке или в корне вашего жесткого диска.

Выберите место для хранения проектов веб-сайта. Здесь, создайте новую папку с именем web-projects (или аналогичной). Это то место, где будут располагаться все ваши проекты сайтов.

Внутри этой первой папки, создайте другую папку для хранения вашего первого веб-сайта. Назовите ее test-site (или как-то более творчески).

**Небольшое отступление о регистре и пробелах**

Edit

Вы заметите, что в этой статье, мы просим вас называть папки и файлы полностью в нижнем регистре без пробелов. Это потому что:

Многие компьютеры, в частности веб-серверы, чувствительны к регистру. Так, например, если вы положили изображение на свой веб-сайт в test-site/MyImage.jpg, а затем в другом файле вы пытаетесь вызвать изображение как test-site/myimage.jpg, он может не сработать.

Браузеры, веб-серверы и языки программирования не обрабатывают пробелы последовательно. Например, если вы используете пробелы в имени файла, некоторые системы могут отнестись к имени файла как к двум именам файлов. Некоторые серверы заменяют пробелы в вашем имени файла на "%20" (символьный код для пробелов в URI), нарушая все ваши ссылки. Лучше разделять слова с помощью тире и нижнего подчеркивания: my-file.html или my\_file.html.

По этим причинам, лучше всего приобрести привычку писать названия ваших папок и файлов в нижнем регистре и без пробелов, по крайней мере, пока вы не поймете, зачем это нужно. Так вы столкнетесь с меньшим количеством проблем.

**Какую структуру должен иметь ваш веб-сайт?**

Edit

Далее давайте взглянем на то, какую структуру должен иметь наш тестовый сайт. Наиболее распространенные вещи, которые есть в любом сайте, создаваемом вами: индексный файл HTML, папки, содержащие изображения, файлы стилей и файлы скриптов. Давайте создадим их сейчас:

index.html: Этот файл обычно содержит контент домашней страницы, то есть текст и изображения, которые люди видят, когда они впервые попадают на ваш сайт. Используя ваш текстовый редактор, создайте новый файл с именем index.html и сохраните его прямо внутри вашей папки test-site.

Папка images: Эта папка обычно содержит все изображения, которые вы используете на вашем сайте. Создайте папку с именем images внутри вашей папки test-site.

Папка styles: Эта папка обычно содержит CSS код, используемый для стилизации вашего контента (например, настройка текста и цвета фона). Создайте папку с именем styles внутри вашей папки test-site.

Папка scripts: Эта папка обычно содержит весь JavaScript код, используемый для добавления интерактивных функций на вашем сайте (например, кнопки которые загружают данные при клике). Создайте папку с именем scripts внутри вашей папки test-site.



Примечание: На компьютерах под управлением Windows у вас могут возникнуть проблемы с отображением имен файлов, поскольку у Windows есть надоедливая настройка с названием Скрывать расширения для известных типов файлов включенную по умолчанию. Обычно вы можете отключить ее, перейдя в проводник, выбрать вариант Свойства папки... и снять флажок Скрывать расширения для зарегистрированных типов файлов, затем щёлкнуть OK. Для получения более точной информации, охватывающей вашу версию Windows, выполните поиск в Интернете.

**Путь к файлам**

Edit

Для того, чтобы файлы общались друг с другом, вы должны указать файлам путь между ними - обычно один файл знает, где находится другой. Чтобы продемонстрировать это, мы вставим немного HTML в наш файл index.html и научим его отображать изображение, которое вы выбрали в статье "Каким должен быть ваш веб-сайт?"

Скопируйте изображение, которое вы выбрали ранее, в папку images.

Откройте ваш файл index.html и вставьте следующий код в файл именно в таком виде. Прямо сейчас не беспокойтесь о том, что все это значит - позже в этом руководстве мы рассмотрим структуру более подробно.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>My test page</title>

</head>

<body>

<img src="" alt="My test image">

</body>

</html>

Строка <img src="" alt="My test image"> - это HTML код, который вставляет изображение на страницу. Мы должны сказать HTML, где находится изображение. Изображение находится внутри папки images, которая находится в той же директории что и index.html. Пройдя вниз по файловой структуре от index.html до нашего изображения, получим путь к файлу, который нам нужен. Он выглядит как images/your-image-filename. Например наше изображение, названное firefox-icon.png, имеет такой путь к файлу: images/firefox-icon.png.

Вставьте путь к файлу в ваш HTML код между двойными кавычками src="".

Сохраните ваш HTML файл, а затем загрузите его в вашем браузере (двойной щелчок по файлу). Вы должны увидеть вашу новую веб-страницу, отображающую ваше изображение!

**Несколько правил о путях к файлам:**

Для ссылки на целевой файл в той же директории, что и вызывающий HTML файл, просто используйте имя файла, например, my-image.jpg.

Для ссылки на файл в поддиректории, напишите имя директории в начале пути, плюс косую черту (forward slash, слеш), например: subdirectory/my-image.jpg.

Для ссылки на целевой файл в директории выше вызывающего HTML файла, напишите две точки. Например, если index.html находится внутри подпапки test-site, а my-image.png - внутри test-site, вы можете обратиться к my-image.png из index.html, используя ../my-image.png.

Вы можете комбинировать их так, как вам нравится, например ../subdirectory/another-subdirectory/my-image.png.

На данный момент это все, что вам нужно знать

Примечание: Файловая система Windows стремится использовать обратный слеш (backslash), а не косую черту, например C:\windows. Это не имеет значения, даже если вы разрабатываете веб-сайт на Windows, вы все равно должны использовать обычные слеши в вашем коде.

**Что должно быть сделано?**

Edit

Поиск (теги): НОВОСТИ (+) КОНТЕНТ WIKI MAN'ы ФОРУМ

Каталог документации / Раздел "Web мастеру, CGI, Perl, PHP, Apache" / Оглавление документа

Оглавление | Назад | Вперёд | Индекс

Глава 7

Другая Функциональность JavaScript

В этой главе рассматривается дополнительная функциональность серверного JavaScript, которую Вы можете использовать для отправки e-mail из Вашего приложения, для доступа к файловой системе сервера, подключения внешних библиотек или непосредственного манипулирования клиентскими запросами и клиентскими ответами.

В главе имеются следующие разделы:

Почтовая Служба

Служба Файловой Системы

Работа с Внешними Библиотеками

Манипуляции с Запросами и Ответами

Почтовая Служба/Mail Service

Вашему приложению может понадобиться отправить email-сообщение. Для этого Вы используете экземпляры класса SendMail. Единственным методом SendMail является send, для отправки сообщения, а errorCode и errorMessage служат для интерпретации ошибок.

Например, следующий скрипт отсылает почту в vpg со специфицированной темой/subject и телом сообщения:

<server>

SMName = new SendMail();

SMName.To = "vpg@royalairways.com";

SMName.From = "thisapp@netscape.com";

SMName.Subject = "Here's the information you wanted";

SMName.Body = "sharm, maldives, phuket, coral sea, taveuni, maui,

cocos island, marathon cay, san salvador";

SMName.send();

</server>

В таблице даны свойства класса SendMail. Свойства To и From необходимы; все остальные свойства - по выбору/optional.

Таблица 7.1 Свойства класса SendMail

To

Список разделённых запятыми первичных/primary получателей сообщения.

From

Имя пользователя/user name отправляющего сообщение.

Cc

Список разделённых запятыми дополнительных получателей сообщения.

Bcc

Список разделённых запятыми получателей сообщения, чьи имена не должны быть видны в сообщении.

Smtpserver

Имя почтового сервера (SMTP). По умолчанию это свойство имеет значение, установленное на сервере администрирования.

Subject

Тема сообщения.

Body

Текст сообщения.

Вы можете добавлять к этим свойствам любые другие. Все свойства класса SendMail включаются в шапку/header сообщения при фактической отправке. Например, следующий код отсылает сообщение получателю bill от vpg, устанавливая в поле vpg organization значение Royal Airways. Отвечает на сообщение от vpgboss.

mailObj["Reply-to"] = "vpgboss";

mailObj.Organization = "Royal Airways";

mailObj.From = "vpg";

mailObj.To = "bill";

mailObj.send();

Дополнительно о предопределённых полях шапки см. RFC 822, стандарт формата текстовых сообщений Internet.

Класс SendMail позволяет отправлять простое текстовое почтовое сообщение или сложное MIME-сообщение. Вы можете также добавить в сообщению приложение/attachment. Для отправки MIME-сообщения добавьте свойство Content-type к объекту SendMail и укажите в нём MIME-тип сообщения.

Например, следующий участок кода отсылает изображение в формате GIF:

<server>

SMName = new SendMail();

SMName.To = "vpg@royalairways.com";

SMName.From = "thisapp@netscape.com";

SMName.Subject = "Here's the image file you wanted";

SMName["Content-type"] = "image/gif";

SMName["Content-Transfer-Encoding"] = "base64";

// В следующем операторе image2.gif обязан быть кодирован с базой 64/base 64.

// Если вы используете uuencode для кодирования GIF-файла, удалите header

// (например, "begin 644 image2.gif")

и замыкающий ("end").

fileObj = new File("/usr/somebody/image2.gif");

openFlag = fileObj.open("r");

if ( openFlag ) {

len = fileObj.getLength();

SMName.Body = fileObj.read(len);

SMName.send();

}

</server>

Некоторые MIME-типы требуют больше информации. Например, если content type это multipart/mixed, Вы обязаны также специфицировать сепаратор границ для одного или более различных наборов данных тела. Например, следующий код отсылает многочастное сообщение, содержащее две части, каждая из которых является обычным текстом:

<server>

SMName = new SendMail();

SMName.To = "vpg@royalairways.com";

SMName.From = "thisapp@netscape.com";

SMName.Subject = "Here's the information you wanted";

SMName["Content-type"]

= "multipart/mixed; boundary=\"simple boundary\"";

fileObj = new File("/usr/vpg/multi.txt");

openFlag = fileObj.open("r");

if ( openFlag ) {

len = fileObj.getLength();

SMName.Body = fileObj.read(len);

SMName.send();

}

</server>

Вот файл multi.txt, содержащий многочастное сообщение:

Это место для преамбулы.

Она игнорируется.

Это удобное место для комментария,

предназначенного для читателей, не знакомых с MIME.

--простая граница

Это первая часть тела сообщения.

Это НЕ конец с символом обрыва строки.

--простая граница

Content-Type: text/plain; charset=us-ascii

Это вторая часть тела сообщения.

Это КОНЕЦ с символом обрыва строки.

--простая граница--

Это эпилог. Он также игнорируется.

Вы можете вкладывать друг в друга многочастные сообщения. То есть, если у Вас есть сообщение, content type которого многочастный, Вы можете включить другой многочастное сообщение в его тело. В таких случаях будьте внимательны и убедитесь, штаа каждый вложенный многочастный объект использует разные ограничители.

Детали о MIME-типах см. в RFC 1341 1 , MIME-стандарте. Об отправке почтовых сообщений в JavaScript см. также описание этого класса в книге Серверный JavaScript. Справочник .

Служба Файловой Системы

JavaScript предоставляет класс File, который даёт приложению возможность записывать в файловой системе сервера. Это используется для генерации постоянных HTML-файлов и хранения информации без использования сервера БД. Одним из важнейших преимуществ хранения информации в файле вместо JavaScript-объектов является то, что информация сохраняется даже при отказе сервера.

Проблемы Безопасности

Соблюдайте осторожность при использовании класса File. Приложение JavaScript может читать или записывать файлы везде, где позволяет операционная система, в том числе, возможно, и в системных файлах. Вы должны быть уверены, что Ваше приложение не позволяет читать файлы паролей и другую закрытую информацию или записывать в файлы. Делайте так, чтобы имена файлов, которые Вы передаёте в методы, не могли быть изменены хакерами.

Например, не используйте свойства объектов client или request в качестве имён файлов, поскольку эти значения могут стать доступными хакеру через куки или URL. В таких случаях хакер сможет модифицировать куки или URL, чтобы получить доступ к закрытым файлам.

Исходя из таких же соображений, Navigator не предоставляет автоматического доступа к файловой системе клиентской машины. Если необходимо, пользователь может сохранять информацию непосредственно в клиентской файловой системе, делая соответствующий выбор в меню Navigator'а.

Создание File-Объекта

Чтобы создать экземпляр класса File, используйте стандартный синтаксис JavaScript для создания объекта:

fileObjectName = new File("path");

Здесь fileObjectName это имя, по которому Вы обращаетесь к файлу, а path это полный путь к файлу. Этот path должен быть в формате серверной файловой системы, а не URL.

Вы можете отобразить имя файла, используя функцию write с File -объектом в качестве аргумента. Например, следующий оператор выводит имя файла:

x = new File("\path\file.txt");

write(x);

Открытие и Закрытие Файла

После создания File -объекта Вы можете использовать метод open для открытия файла и чтения и записи. Метод open имеет следующий синтаксис:

result = fileObjectName.open("mode");

Это метод возвращает true, если операция прошла успешно, и false в ином случае. Если файл уже открыт, операция терпит неудачу, и оригинальный файл остаётся открытым.

Параметр mode это строка, специфицирующая режим открытия файла. В таблице описаны эти режимы.

Таблица 7.2 Режимы доступа к файлуРежим Описание

r

Открывает файл, если он существует, как текстовый файл для чтения и возвращает true. Если файл не существует, возвращает false.

w

Открывает файл как текстовый файл для записи. Создаёт новый (первоначально пустой) текстовый файл, независимо от того, существует файл или нет.

a

Открывает файл как текстовый файл для дополнения (записи в конец файла). Если файл ещё не существует, создаёт его.

r+

Открывает файл как текстовый файл для чтения и записи. Чтение и запись начинаются в начале файла. Если файл существует, возвращает true. Если не существует, возвращает false.

w+

Открывает файл как текстовый файл для чтения и записи. Создаёт новый (первоначально пустой) текстовый файл, независимо от того, существует файл или нет.

a+

Открывает файл как текстовый файл для чтения и записи. Чтение и запись начинаются в конце файла. Если файл не существует, создаёт его.

b

Если присоединён к одному из вышеуказанных режимов, открывает файл как бинарный/двоичный файл, а не как текстовый. Применяется только для операционных систем Windows.

Когда приложение заканчивает использование файла, оно может закрыть его, вызвав метод close. Если файл не открыт, close терпит неудачу. Это метод возвращает true при успехе и false - в противном случае.

Блокировка Файлов

Часто доступ ко многим приложениям могут выполнять одновременно многие пользователи. Вообще разные пользователи не должны пытаться одновременно вносить изменения в файлы, поскольку это может привести к непредсказуемым ошибкам.

Чтобы предотвратить модификацию файла одновременно несколькими пользователями, используйте один из механизмов блокирования, предоставляемых службой Session Management Service, как описано в разделе "Безопасное Совместное Использование Объектов с Помощью Блокировки". Если один пользователь блокировал файл, другие пользователи приложения должны ждать, пока файл не будет разблокирован. В общем это означает, что lock (замок/блокировка) должна предшествовать всем файловым операциям; после выполнения операций должно выполняться unlock (разблокирование).

Если только одно приложение может модифицировать данный файл, Вы можете получать блокировку в объекте project. Если более чем одно приложение может иметь доступ к одному и тому же файлу, получайте блокировку в объекте server.

Например, у Вас создан файл myFile. Затем Вы может использовать его так:

if ( project.lock() ) {

myFile.open("r");

// ... файл используется ...

myFile.close();

project.unlock();

}

Таким образом, только один пользователь приложения может изменять файл в данный момент времени. Для более тонкого управления блокировкой Вы можете создать Ваш собственный экземпляр класса Lock для управления доступом к данному файлу. Это описано в разделе "Использование Lock-Экземпляров".

Работа с Файлами

Класс File имеет несколько методов, которые можно использовать после открытия файла:

Позиционирование: setPosition, getPosition, eof. Это методы для установки и получения текущей позиции указателя в файле и для определения, не находится ли указатель в конце файла.

Чтение из файла: read, readln, readByte.

Запись в файл: write, writeln, writeByte, flush.

Конвертация двоичного и текстового форматов: byteToString, stringToByte. Конвертируют одно число в символ и наоборот.

Информационные методы: getLength, exists, error, clearError. Для получения информации о файле и для получения и очистки error-статуса.

Эти методы описаны в последующих разделах.

Позиционирование Внутри Файла

Физический файл, ассоциированный с File -объектом, имеет указатель текущей позиции в файле. Когда Вы открываете файл, указатель находится в начале либо в конце файла, в зависимости от режима, использованного при открытии файла. В пустом файле начало и конец файла это одна точка.

Метод setPosition позиционирует указатель в файле, возвращая true при успехе и false - в ином случае.

fileObj.setPosition(position);

fileObj.setPosition(position, reference);

Здесь fileObj это File -объект, position это целое число, указывающее позицию указателя, а reference указывает относительную точку для position таким образом:

0: относительно начала файла

1: относительно текущей позиции

2: относительно конца файла

Иное (или unspecified): относительно начала файла

Метод getPosition возвращает текущую позицию в файле, где первый байт файла это всегда байт 0. Этот метод возвращает -1, если имеется ошибка.

fileObj.getPosition();

Метод eof возвращает true, если указатель находится в конце файла, и false - в ином случае. Этот метод возвращает true после первой операции чтения, которая пытается прочесть после конца файла.

fileObj.eof();

Чтение из Файла

Используйте методы read, readln и readByte для чтения из файла.

Метод read читает специфицированное количество байтов из файла и возвращает строку.

fileObj.read(count);

Здесь fileObj это File -объект, а count это целое число, специфицирующее количество байтов для чтения. Если count специфицирует больше байтов, чем осталось в файле, метод читает до конца файла.

Метод readln читает следующую строку файла и возвращает её как строку.

fileObj.readln();

Здесь fileObj это File -объект. Символы-разделители строк (\r\n в Windows или просто \n в Unix или Macintosh) не включаются в строку. Символ \r пропускается; \n определяет действительный конец строки. Этот компромисс даёт осмысленное поведение на всех платформах.

Метод readByte читает следующий байт из файла и возвращает числовое значение следующего байта или -1.

fileObj.readByte();

Запись в Файл

Для записи в файл имеются методы write, writeln, writeByte и flush.

Метод write записывает строку в файл. В случае успеха операции возвращает true и false - в ином случае.

fileObj.write(string);

Здесь fileObj это File -объект, в string это строка JavaScript.

Метод writeln записывает сроку в файл и вводит последующие \n (\r\n в текстовом режиме Windows). Возвращает true при успешном выполнении записи и false - в ином случае.

fileObj.writeln(string);

Метод writeByte записывает байт в файл. Возвращает true в случае успеха и false - в противном случае.

fileObj.writeByte(number);

Здесь fileObj это File -объект, а number это число.

Когда Вы используете один из этих методов, содержимое файла внутренне буферизуется. Метод flush записывает буфер в файл на диске. Этот метод возвращает true в случае успеха и false - в противном случае.

fileObj.flush();

Конвертация Данных

Есть два основных формата файлов : ASCII-текст и бинарный. Методы byteToString и stringToByte класса File конвертируют данные этих форматов.

Метод byteToString конвертирует число в односимвольную строку. Это static-метод. Вы можете использовать класс File сам по себе, а не его экземпляр, для вызова этого метода.

File.byteToString(number);

Если аргумент - не число, метод возвращает пустую строку.

Метод stringToByte конвертирует первый символ своего аргумента, строку, в число. Это также static-метод.

File.stringToByte(string);

Метод возвращает числовое значение первого символа или 0.

Получение Информации о Файле

Вы можете использовать несколько методов класса File для получения информации о файлах и работы с error-статусом.

Метод getLength возвращает число символов в текстовом файле или количество байтов в любом другом файле. Возвращает -1, если возникла ошибка.

fileObj.getLength();

Метод exists возвращает true, если файл существует, и false - в ином случае.

fileObj.exists();

Метод error возвращает статус ошибки или -1, если файл не открыт или не может быть открыт. Статус ошибки/error status это ненулевое значение, если ошибка возникла, и 0 в ином случае (нет ошибки). Коды статуса ошибки зависят от платформы; обратитесь к документации по Вашей ОС.

fileObj.error();

Метод clearError очищает error-статус (значение error) и значение eof.

fileObj.clearError();

Пример

На сервере Netscape имеется приложение-образец Viewer. Поскольку это приложение даёт возможность просматривать файлы на сервере, оно не устанавливается автоматически.

Viewer это хороший пример использования класса File. Если Вы установили это приложение, позаботьтесь об ограничении доступа к нему, чтобы неавторизованный пользователь не мог просматривать файлы на сервере. Об ограничении доступа к приложению см. раздел "Публикация Приложения".

Следующий код из приложения Viewer создаёт экземпляр класса File, открывает его для чтения и генерирует HTML, отражающий строки файла, с разделительной линией после каждой строки.

x = new File("\tmp\names.txt");

fileIsOpen = x.open("r");

if (fileIsOpen) {

write("file name: " + x + "<BR>");

while (!x.eof()) {

line = x.readln();

if (!x.eof())

write(line+"<br>");

}

if (x.error() != 0)

write("error

reading file" + "<BR>");

x.close();

}

Работа с Внешними Библиотеками

Для взаимодействия с внешними приложениями рекомендуется использовать LiveConnect, как описано в Главе 14, "LiveConnect. Обзор.". Однако Вы можете также вызывать функции, написанные на других языках, таких как C, C++ или Pascal, и скомпилированных в библиотеки. Такие функции называются native-функции или внешние функции. Библиотеки внешних функций, называемые внешними библиотеками, являются библиотеками динамической компоновки/dll в ОС Windows и совместно используемыми объектами/shared objects - в ОС Unix.

Важно!

Будьте осторожны при использовании внешних функций в Вашем приложении. Внешние функции могут нарушить защиту/безопасность, если внешняя программа выполняет команды пользователя в командной строке (например, программа, дающая возможность войти в ОС, или команды оболочки/shell). Эта функциональность опасна, так как хакер может присоединить дополнительные команды, используя точку с запятой для присоединения нескольких операторов. Лучше исключить использование ввода командной строки, если Вы не проверяете его достаточно жёстко.

Внешние функции используются в следующих случаях:

Если у Вас уже имеются сложные функции, написанные на других языках, которые можно использовать в приложении.

Если приложению нужны функции интенсивных вычислений. Обычно функции, написанные в native-коде, работают быстрее, чем функции, написанные на языке JavaScript.

Если приложение должно выполнять задачи, которые невозможно выполнить в JavaScript.

В директории примеров jsaccall есть несколько исходных и header-файлов, иллюстрирующих вызов функций внешних библиотек из приложения JavaScript.

В Application Manager Вы ассоциируете внешнюю библиотеку с определённым приложением. Но после того как библиотека ассоциирована с одним из приложений, она становится доступной всем другим установленным приложениям.

Выполните следующие действия для использования библиотеки внешних функций в приложении JavaScript:

Напишите и скомпилируйте библиотеку внешних функций в форме, совместимой с JavaScript. (См. "Рекомендации по Написанию Внешних Функций")

.

С помощью Application Manager идентифицируйте используемую библиотеку, установив новое приложение или изменив параметры установки существующего приложения. После этого все приложения на данном сервере смогут вызывать внешние функции этой библиотеки. (См. "Идентификация Файлов Библиотек")

.

Рестартуйте сервер, чтобы загрузить библиотеку с Вашим приложением. Функции внешней библиотеки теперь доступны для всех приложений на этом сервере.

В вашем приложении используйте функции JavaScript registerCFunction - для идентифицирования вызываемых функций библиотеки - и callC - для вызова этих функций. (См. "Регистрация Внешних Функций" и "Использование Внешних Функций в JavaScript")

.

Рекомпилируйте и рестартуйте Ваше приложение, чтобы изменения вступили в силу.

Важно!

Вы обязаны рестартовать Ваш сервер, чтобы установить библиотеку для использования с приложениями. Вы обязаны рестартовать сервер каждый раз после добавления файлов новых библиотек или изменения имён файлов библиотек, используемых приложениями.

Рекомендации по Написанию Внешних Функций

Хотя Вы можете написать внешние библиотеки на любом языке, JavaScript использует соглашения языка C по вызову. Ваш код обязан подключать header-файл jsaccall.h, находящийся в директории js\samples\jsaccall\.

Эта директория также содержит исходный код примеров приложений, которые вызывают функции C, определённые в jsaccall.c. Просмотрите эти файлы, чтобы найти более конкретные рекомендации по написанию функций C для использования с JavaScript.

Функции, вызываемые из JavaScript, обязаны быть экспортируемыми и обязаны соответствовать этому определению типа:

typedef void (\*LivewireUserCFunction)

(int argc, struct

LivewireCCallData argv[],

struct LivewireCCallData\* result, pblock\* pb,

Session\* sn, Request\* rq);

Идентификация Файлов Библиотек

Прежде чем Вы сможете запустить приложение, использующее функции внешних библиотек, Вы обязаны идентифицировать файлы этих библиотек. Используя Application Manager, Вы можете идентифицировать библиотеки, когда Вы устанавливаете приложение (щёлкнув Add) или когда модифицируете параметры инсталяции приложения (щёлкнув Modify). Дополнительно об идентификации файлов библиотек с помощью Application Manager см. "Установка Нового Приложения".

Важно!

После ввода пути к файлам библиотек в Application Manager Вы обязаны рестартовать сервер, чтобы изменения вступили в силу. Затем необходимо скомпилировать и рестартовать приложение.

После идентификации внешних библиотек с помощью Application Manager все приложения, запущенные на данном сервере, могут вызывать функции этих библиотек (используя registerCFunction и callC).

Регистрация Внешних Функций

Используйте JavaScript-функцию registerCFunction для регистрации внешней функции для использования с приложением JavaScript. Эта функция имеет следующий синтаксис:

registerCFunction(JSFunctionName, libraryPath, CFunctionName);

Здесь JSFunctionName это имя функции как она будет вызываться в JavaScript функцией callC. Параметр libraryPath это полный путь к библиотеке, использующий соглашения Вашей ОС, а параметр CFunctionName это имя C-функции как она определена в библиотеке. В вызове этого метода Вы обязаны вводить имя точно в указанном регистре, указанном в Application Manager, даже в ОС NT.

ПРИМЕЧАНИЕ:

Backslash (\) это специальный символ в JavaScript, поэтому Вы обязаны использовать forward slash (/) или двойной backslash (\\) для отделения Windows-директории и имён файлов в libraryPath.

Данная функция возвращает true, если функция зарегистрирована успешно, и false - в ином случае. Функция может потерпеть неудачу, если JavaScript не сможет найти библиотеку по специфицированному пути или не найдёт специфицированную функцию в библиотеке.

Приложение обязано использовать registerCFunction для регистрации функции, прежде чем сможет использовать callC для её вызова. После того как приложение зарегистрировало функцию, оно может вызывать эту функцию любое число раз. Хорошим местом для регистрации функций является начальная страница приложения.

Использование Внешних Функций в JavaScript

После того как Ваше приложение зарегистрировало функцию, оно может использовать callC для её вызова. Эта функция имеет следующий синтаксис:

callC(JSFunctionName, arguments);

Здесь JSFunctionName это имя функции, как она была идентифицирована с помощью registerCFunction, а arguments это список разделённых запятыми аргументов внешней функции. В качестве аргументов могут использоваться любые значения JavaScript: строки, числа, булевы значения, объекты или null. Количество аргументов обязано соответствовать количеству необходимых аргументов внешней функции. Хотя Вы можете специфицировать объект JavaScript в качестве аргумента, это используется редко, поскольку объект конвертируется в строку перед передачей внешней функции.

Эта функция возвращает строковое значение, возвращённое внешней функцией. Функция callC может возвращать только строковые значения.

Приложение-образец jsaccall иллюстрирует использование внешних функций. Директория jsaccall содержит исходный код C (в jsaccall.c), определяющий C-функцию с именем mystuff\_EchoCCallArguments. Эта функция принимает любое количество аргументов и возвращает строку, содержащую HTML с перечислением аргументов. Это пример иллюстрирует вызов функций C из приложения JavaScript и возвращаемые значения.

Чтобы запустить jsaccall, Вы обязаны скомпилировать jsaccall.c имеющимся у Вас компилятором C. Командные строки для нескольких распространённых компиляторов даны в файле в виде комментариев.

Следующие операторы JavaScript (взятые из jsaccall.html) регистрируют C-функцию как echoCCallArguments в JavaScript, вызывают функцию echoCCallArguments, а затем генерируют HTML на основе значения, возвращённого данной функцией.

var isRegistered = registerCFunction("echoCCallArguments",

"c:\\mycode\\mystuff.dll", "mystuff\_EchoCCallArguments");

if (isRegistered == true) {

var returnValue = callC("echoCCallArguments",

"first arg",

42,

true,

"last arg");

write(returnValue);

}

else {

write("registerCFunction() returned

false, "

+ "check server error log for details")

}

Функция echoCCallArguments создаёт результирующую строку, содержащую HTML, который выводит тип и значение каждого переданного ей аргумента JavaScript. Если registerCFunction возвращает true, вышеприведённый код генерирует такой HTML:

argc = 4<BR>

argv[0].tag: string; value = first arg<BR>

argv[1].tag: double; value = 42<BR>

argv[2].tag: boolean; value = true<BR>

argv[3].tag: string; value = last arg<BR>

Манипуляции с Запросами и Ответами

Обычный запрос, высылаемый клиентом на сервер, не имеет content type (тип содержимого). Машина выполнения JavaScript автоматически обрабатывает такие запросы. Однако, если пользователь отправляет форму, то клиент автоматически помещает content type в шапку/header, чтобы сообщить серверу, как интерпретировать данные формы. Этот content type обычно application/x-www-form-urlencoded. Машина выполнения также автоматически обрабатывает запросы с этим content type. В таких ситуациях Вам редко нужен прямой доступ к шапкам/header запроса или ответа. Если, однако, Ваше приложение использует иной content type, оно обязано уметь работать с шапкой запроса.

В свою очередь, типичный ответ сервера клиентку имеет тип содержимого text/html. Машина выполнения автоматически добавляет этот content type в свой ответ. Если Вам в ответе нужен другой content type, Вы обязаны предоставить его сами.

Для поддержки этих потребностей машина выполнения JavaScript на сервере даёт возможность Вашему приложению иметь доступ к (1) шапке/header любого запроса и к (2) телу запроса, имеющего нестандартный content type. Вы уже контролируете тело ответа тэгом SERVER и тэгами HTML. Функциональность, описанная в этом разделе, даёт также возможность управлять шапкой/header ответа.

Вы можете использовать эту функциональность в различных целях. Например, как описано в разделе "Использование Кук", можно осуществлять связь серверных и клиентских процессов с помощью кук. Также Вы можете использовать эту функциональность для поддержки выгрузки файлов.

World Wide Web Consortium публикует онлайновую информацию о протоколе HTTP и о том, что может пересылаться по этому протоколу. См., например, HTTP Specifications and Drafts .

Шапка/Header Запроса

Для доступа к парам имя/значение шапки клиентского запроса используйте метод httpHeader объекта request. Этот метод возвращает объект, чьи свойства и значения соответствуют парам имя/значение шапки.

Например, если запрос содержит куки, header["cookie"] или header.cookie будет его значением. Свойство cookie, содержащее все пары имя/значение этой куки (со значениями, кодированными так, как описано в разделе "Использование Кук"), обязано разбираться Вашим приложением.

Следующий код выводит свойства и значения шапки:

var header = request.httpHeader();

var count = 0;

var i;

for (i in header ) {

write(count + ". " + i + " " + header[i] + "<br>\n");

count++;

}

Если Вы отправили форму методом GET, на выходе получится примерно так:

0. connection Keep-Alive

1. user-agent Mozilla/4.0b1 (WinNT; I)

2. host piccolo:2020

3. accept image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, \*/\*

Если для отправки формы использован метод POST, вывод будет таким:

0. referer http://piccolo:2020/world/hello.html

1. connection Keep-Alive

2. user-agent Mozilla/4.0b1 (WinNT; I)

3. host piccolo:2020

4. accept image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, \*/\*

5. cookie NETSCAPE\_LIVEWIRE.oldname=undefined; NETSCAPE\_LIVEWIRE.number=0

6. content-type multipart/form-data; boundary=---------------------------79741602416605

7. content-length 208

Тело Запроса

Для нормальных HTML-запросов content type (тип содержимого) запроса будет application/x-www-form-urlencoded. После получения запроса с этим content type машина выполнения JavaScript на сервере обрабатывает запрос, используя данные из тела запроса. В этой ситуации Вы не можете получить прямой доступ к необработанным данным тела запроса. (Конечно, Вы можете получить доступ к его содержимому через объекты request и client, сконструированные машиной выполнения).

Если, однако, запрос имеет любой иной content type, машина выполнения не обрабатывает автоматически тело запроса. В этом случае Ваше приложение должно определить, что делать с содержимым.

Возможно, другая страница Вашего приложения отправила запрос этой странице. Следовательно, Ваше приложение обязано предусматривать возможность получения необычного content type и должно знать, как его обработать.

Для получения доступа к телу запроса Вы используете метод getPostData объекта request. Этот метод принимает в качестве параметра количество символов тела. Если Вы специфицируете 0, тело возвращается целиком. return-значение это строка, содержащая запрошенные символы. Если доступных данных нет, метод возвращает пустую строку.

Можно использовать этот метод для получения всех символов за один раз или читать участки данных. Представляйте тело запроса как поток символов. При чтении Вы можете только продвигаться вперёд; Вы не можете читать одни и те же символы несколько раз.

Чтобы назначить всё тело запроса переменной postData, можно использовать следующий оператор:

postData = request.getPostData(0);

Если в качестве параметра специфицировано 0, метод получает весь запрос. Вы можете точно определить, сколько символов содержится в информации, используя свойство content-length шапки таким образом:

length = parseInt(header["content-length"], 10);

Для получения тела запроса небольшими блоками Вы можете специфицировать иной параметр. Например, следующий код обрабатывает тело запроса блоками по 20 символов:

var length = parseInt(header["content-length"], 10);

var i = 0;

while (i < length) {

postData = request.getPostData(20);

// ...обработка postData...

i = i + 20;

}

Конечно, такой подход имеет смысл только тогда, когда Вы знаете, что блоки состоят из 20 символов информации.

Шапка/Header Ответа

Если отправляемый клиенту ответ/response использует специальный content type (тип содержимого), Вы должны кодировать этот content type в шапке ответа. Машина выполнения JavaScript автоматически добавляет content type по умолчанию (text/html) в шапку ответа/response header. Если вам необходима специальная шапка, Вы обязаны сначала удалить из шапки старый content type по умолчанию, а затем уже добавить новый. Это делается при помощи функций addResponseHeader и deleteResponseHeader.

Например, если Ваш response использует royalairways-format как специальный content type, Вы можете специфицировать его так:

deleteResponseHeader("content-type");

addResponseHeader("content-type","royalairways-format");

Вы можете использовать функцию addResponseHeader для добавления в шапку ответа любой другой нужной информации.

Важно!

Помните, что header отсылается с первой частью ответа/response. Следовательно, Вы должны вызывать эти функции раньше в скрипте на каждой странице. В общем, Вы должны быть уверены, что header ответа установлен до любого из следующих событий:

Машина выполнения сгенерировала 64KB содержимого для HTML-страницы (в этой точке буфер вывода автоматически очищается).

Вы вызываете функцию flush для очистки буфера вывода.

Вы вызываете функцию redirect для изменения клиентских запросов.

Дополнительно см. разделы "Очистка Буфера Вывода" и "Процессинг Времени Выполнения на Сервере".

1 http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1341.txt

Оглавление | Назад | Вперёд | Индекс

Дата последнего обновления: 29 сентября 1999 г.

© Copyright ╘ 1999 Sun Microsystems, Inc. Некоторая часть Copyright ╘ 1999 Netscape Communications Corp. Все Права Зарезервированы.

Закладки на сайте

Проследить за страницей Created 1996-2017 by Maxim Chirkov

Добавить, Реклама, Вебмастеру, ГИД