Серверный JavaScript 1.4.

Руководство по Использованию.

Перевод выполнил Александр Пирамидин.

Прошу все замечания и предложения направлять по e-mail: a\_pyramidin@yahoo.com

В этой книге рассматривается использование ядра и серверного JavaScript версии 1.4. JavaScript это созданный фирмой Netscape межплатформенный объектно-ориентированный язык скриптов (сценариев) для клиентских и серверных приложений.

Оглавление

Введение

Что Нового в Этой Версии?

Поддержка JavaScript 1.4

Изменения в Менеджере Приложений JavaScript

Что Вы Уже Должны Знать

Версии JavaScript

Где Найти Информацию о JavaScript

Обновление Предыдущей Версии

Обратная Совместимость с Предыдущими Версиями

Соглашения по Документам

Глава 1 JavaScript. Обзор.

Что Такое JavaScript?

Ядро, Клиентский и Серверный JavaScript

Ядро JavaScript

Клиентский JavaScript

Серверный JavaScript

JavaScript и Java

Отладка в JavaScript

Visual JavaScript

JavaScript и Спецификация ECMA

Соотношение Между Версиями JavaScript и ECMA

Документация JavaScript и Спецификация ECMA

JavaScript и Технология ECMA

ЧАСТЬ I. Разработка Серверных Приложений.

Глава 2 Введение

Архитектура Приложений JavaScript

Системные Требования

Информация о Конфигурации

Подключение Серверного JavaScript

Защита Менеджера Приложений

Настройка LiveConnect

Локализация Компилятора

Глава 3 Технология Разработки Приложений JavaScript

Основные Этапы Создания Приложения

Менеджер Приложений JavaScript

Создание Исходных Файлов Приложения

Компиляция Приложения

Инсталяция Нового Приложения

URL Приложения

Управление Доступом к Приложению

Модификация Приложения

Удаление Приложения

Старт, Остановка и Рестарт Приложения

Запуск Приложения

Отладка Приложения

Использование Менеджера Приложений для Отладки

Использование URL Отладки

Использование Функции debug

Публикация Приложения

Менеджер Приложений. Детали.

Конфигурация, Установки по Умолчанию.

За Кулисами

ЧАСТЬ II. Возможности Серверного JavaScript.

Глава 4 Быстрый Старт. Образец Приложения.

О Приложениях-Образцах Серверного JavaScript

Hello World

Что Делает Hello

Исходный Скрипт

Модифицирование Hello World

Hangman

Исходные Файлы

Отладка Hangman

Глава 5 Основы Серверного JavaScript

Что Делать и Где

Обзор Процессов Времени Выполнения

Серверный Язык. Обзор.

Прототипы

Использование

Окружение

Классы и Объекты

Внедрение JavaScript в HTML

Тэг SERVER

Обратные Кавычки

Когда Использовать Эту Технику?

Процессинг Времени Выполнения на Сервере

Конструирование HTML-Страницы

Генерация HTML

Очистка Буфера Вывода

Переход к Новому Клиентскому Запросу

Доступ к Переменным CGI

Сообщение Между Сервером и Клиентом

Отправка Сообщений с Клиента на Сервер

Отправка Значений с Сервера Клиенту

Использование "Куков"

Сбор Мусора

Обработка Ошибок в Серверном JavaScript

Глава 6 Обслуживание Сессий

Предопределённые Объекты. Обзор.

Объект request

Свойства

Работа с Картами Изображений

Объект client

Свойства

Уникальное Обращение к Объекту client

Создание Специального Объекта client

Объект project

Свойства

Совместное Использование Объекта project

Объект server

Свойства

Совместное Использование Объекта server

Техника Работы с Объектом client

Сравнение Разных Техник Обслуживания Клиента

Клиентская Техника

Серверная Техника

Период Существования Объекта client

Присоединение Свойств client'а к URL Вручную

Безопасное Использование Объектов с Блокировкой

Использование Lock-Экземпляров

Специальные Lock для Объектов project и server

Как Избежать Мёртвых Блокировок

Глава 7 Дополнительная Функциональность JavaScript

Почтовая Служба

Служба Файловой Системы

Проблемы Безопасности

Создание File-Объекта

Открытие и Закрытие Файла

Блокировка Файлов

Работа с Файлами

Пример

Работа с Внешними Библиотеками

Указания по Написанию Внешних Функций

Идентификация Файлов Библиотек

Регистрация Внешних Функций

Использование Внешних Функций в JavaScript

Манипуляции с Запросами и Ответами

Шапка Запроса

Тело Запроса

Шапка Ответа

ЧАСТЬ III. Служба LiveWire Database Service

Глава 8 Соединение с БД

Взаимодействие с Базами Данных

Соединение с БД

Пулы Соединений с БД

Однопоточные и Многопоточные БД

Рекомендации

Обслуживание Пулов Соединений

Совместное Использование Фиксированного Набора Пулов Соединений

Совместное Использование Массива Пулов Соединений

Индивидуальные Соединения с БД

Обслуживание Соединения по Нескольким Запросам

Ожидание Соединения

Запрашивание Свободного Соединения

Глава 9 Работа с БД

Взаимодействие с Реляционной БД

Автоматическое Отображение Результатов Выполнения Запроса

Выполнение Произвольных Операторов SQL

Манипуляции с Результатами Выполнения Запросов с Помощью Курсоров

Создание Курсора

Отображение Значений Записи

Отображение Выражений и Агрегатных Функций

Навигация с Помощью Курсоров

Работа со Столбцами

Изменение Информации Базы Данных

Обслуживание Транзакций

Методы Управления Транзакциями

Работа с Двоичными Данными

Вызов Хранимых Процедур

Обмен Информацией

Этапы Использования Хранимых Процедур

Регистрация Хранимой Процедуры

Определение Прототипа для Хранимой Процедуры

Выполнение Хранимой Процедуры

Работа с Наборами Результатов

Работа с Возвращаемыми Значениями

Работа с Параметрами Вывода

Исключения Informix и Sybase

Глава 10 Конфигурирование Базы Данных

О Службе LiveWire Database Service

Проверка Конфигурации Вашей БД

Поддерживаемые Клиенты БД и ODBC-Драйверы

DB2

Informix

Удалённый Informix

Локальный Informix

ODBC

ODBC DSN (Только NT)

OpenLink ODBC-Драйвер (Только Solaris)

Visigenic ODBC-Драйвер (Только Unix)

Oracle

Удалённый Oracle

Локальный Oracle

Sybase

Удалённый Sybase

Локальный Sybase

Sybase (Только Unix)

Глава 11 Конвертация Типов Данных

О Конвертации Типов Данных

Работа с Датами и БД

Конвертация Типа Данных Базой Данных

Глава 12 Обработка Ошибок LiveWire

Проверка Ошибочных Условий

Возвращаемые Значения

Число

Объект

Булево

Строка

Пустое

Методы для Работы с Ошибками

Статус-Коды

Глава 13 Приложения-Образцы Videoapp и Oldvideo

О Приложениях Videoapp и Oldvideo

Конфигурирование Окружения

Соединение с БД и Перекомпиляция

Создание БД

Запуск Videoapp

Исходные Файлы

Архитектура Приложения

Модифицирование Videoapp

ЧАСТЬ IV Работа с LiveConnect

Глава 14 LiveConnect. Обзор.

Что Такое LiveConnect?

Работа с Оболочками

Взаимодействие JavaScript и Java

Объект Packages

Работа с Массивами Java

Обращение к Классу и Пакету

Аргументы Типа char

Пример Вызова Java из JavaScript

Взаимодействие Java и JavaScript

Использование Классов LiveConnect

Доступ к Серверному JavaScript

Конвертация Типов Данных

Конвертация из JavaScript в Java

Конвертация из Java в JavaScript

Словарь

Индекс

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

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