JavaScript

The Definitive Guide

Fourth Edition

David Flanagan

O'REILLY®

JavaScript

Подробное руководство

Четвертое издание

Дэвид Флэнаган

Санкт-Петербург — Москва
2004

Дэвид Флэнаган

JavaScript. Подробное руководство, 4-е издание

Перевод Л. Фрейдина

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*А. Королев*
Редактор	*В. Овчинников*
Корректура	*С. Беляева, С. Журавина*
Верстка	*В. Овчинников*

Флэнаган Д.
JavaScript. Подробное руководство. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 960 с., ил.
ISBN 5-93286-055-3

Четвертое издание бестселлера по JavaScript содержит полное описание базового языка JavaScript, а также традиционной и стандартизованной объектных моделей документа, реализованных в веб-броузерах. Примеры, включенные в книгу, можно использовать для решения распространенных задач, таких как проверка данных формы, работа с cookies и создание переносимой анимации DHTML. Части с III по V представляют собой подробные справочники по базовому API JavaScript, традиционному клиентскому API и стандартизованному API W3C DOM, в которых описываются все объекты, методы, свойства, конструкторы, константы, функции и обработчики событий этих API. Издание дополнено с учетом возможностей JavaScript 1.5 (ECMAScript v3) и содержит описание стандарта W3C DOM (Level 1 и Level 2), при этом для обратной совместимости сохранен материал по традиционному DOM Level 0.

Эта книга необходима каждому, кто пишет на JavaScript, независимо от его опыта. Она будет особенно полезна тем, кто работает с последними, соответствующими стандартам веб-броузерами, такими как Internet Explorer 6, Netscape 6 и Mozilla. Вебмастеры узнают, как применять JavaScript для построения динамических веб-страниц. Опытные разработчики смогут быстро приступить к написанию сложных программ.

ISBN 5-93286-055-3
ISBN 0-596-00048-0 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 18.02.2004. Формат 70×100 1/16 . Печать офсетная.
Объем 60 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Эта книга посвящается всем,
кто учит жить мирно и противостоит насилию.

Оглавление

Предисловие

Со времени публикации третьего издания этой книги в мире веб-программирования на JavaScript™ произошло много изменений, в том числе:

- Публикация второго и третьего изданий стандарта ECMA-262, обновивших ядро языка JavaScript. Были выпущены соответствующие версии интерпретаторов Netscape JavaScript и Microsoft JScript.
- Исходные тексты интерпретаторов JavaScript от Netscape (один написан на C, другой на Java™) выпущены в виде открытого исходного кода и доступны для всех, кто хочет встроить язык сценариев в свое приложение.
- World Wide Web Consortium (W3C) опубликовал две версии (два уровня) стандарта Document Object Model (DOM). Новейшие броузеры поддерживают этот стандарт (в разной степени) и позволяют клиентскому коду JavaScript взаимодействовать с содержимым документа и создавать сложные эффекты Dynamic HTML (DHTML). Широкую поддержку получили и другие стандарты W3C, такие как HTML 4, CSS1 и CSS2.
- Организация Mozilla, взяв за основу предоставленный Netscape исходный код, создала хороший броузер пятого поколения. Во время написания этой книги броузер Mozilla еще не дошел до уровня версии 1.0, но он уже достаточно зрелый для того, чтобы компания Netscape использовала базу кода Mozilla в качестве базы для своих броузеров версий 6.0 и 6.1.
- Microsoft Internet Explorer стал доминировать на настольных системах. Однако броузер Netscape/Mozilla по-прежнему важен для веб-разработчиков, в особенности по причине отличной поддержки веб-стандартов. Кроме того, как не менее важные надо рассматривать и второстепенные броузеры, такие как Opera (*http://www.opera.com*) и Konquerer (*http://www.konquerer.org*).
- Веб-броузеры (и интерпретаторы JavaScript) больше не ограничены настольными системами и работают даже на PDA и сотовых телефонах.

В итоге ядро языка JavaScript стало более зрелым. Язык стал стандартизованным и используется в более разнообразных средах, чем раньше. Коллапс доли рынка Netscape сделал возможным расширение ниши настольных веб-броузеров, а те из них, в которых реализована поддержка JavaScript, также стали доступными на платформах, отличных от настольных. Произошел явный, если не полный, переход к веб-стандартам. Частичная или полная ре-

ализация стандарта DOM в новейших броузерах дает веб-разработчикам давно ожидаемый независимый интерфейс API.

Что нового в 4-м издании

Это издание «JavaScript: The Definitive Guide» было тщательно переработано в свете только что описанных изменений. Самые важные из них включают полное описание JavaScript 1.5 и третьего издания стандарта ECMA-262, на котором он основан, а также полное описание стандарта DOM Level 2.

Фокус внимания смещен от конкретных реализаций JavaScript и броузеров (JavaScript 1.2, Netscape 4, IE 5 и т. д.) к документированию стандартов, на которых базируются (или должны базироваться) эти реализации. По причине множественности реализаций теперь практически невозможно описать в одной книге все возможности, фирменные расширения, индивидуальные особенности и ошибки всех реализаций, а любому разработчику – понять их. Книга ориентирована на спецификации, а не на реализации, что делает работу с ней проще, и если вы примете тот же подход, то ваш код JavaScript станет более переносимым и легким в сопровождении. Особое внимание в новом материале уделено стандартам по ядру JavaScript и DOM.

Другое существенное изменение в этом издании касается разбиения справочного раздела на три отдельные части. Материал по базовому JavaScript был отделен от материала по клиентскому JavaScript (часть IV) и помещен в самостоятельный раздел (часть III). Это сделано для удобства JavaScript-программистов, работающих с языком в среде, отличной от веб-броузера, и не интересующихся клиентским JavaScript.

Новый материал с описанием W3C DOM помещен в самостоятельный раздел (часть V), отдельно от уже известного материала по клиентскому JavaScript. Стандарт DOM определяет API, значительно отличающийся от «устаревшего» API традиционного клиентского JavaScript. В зависимости от целевого броузера разработчики выбирают тот или иной API и обычно не переключаются с одного на другой. Раздельное описание этих двух API также позволяет сохранить организацию справочного материала по клиентскому JavaScript (часть IV), удобную для читателей, знакомых с третьим изданием.

Чтобы освободить место для нового материала и не увеличивать чрезмерно объем книги, я убрал описания тривиальных свойств объектов, повторяющие материал разделов, посвященных объектам. Всем свойствам, которые требуют основательного рассмотрения, а также всем методам отведены специальные разделы. Кроме того, волшебники из O'Reilly заново оформили книгу; она стала меньше, а читать ее по-прежнему легко и удобно.

Типографские соглашения

В этой книге приняты следующие соглашения:

Шрифт OfficinaSansC

Применяется для выделения клавиш клавиатуры или элементов пользовательского интерфейса, таких как кнопка Back или меню Options.

Курсив

Обозначает первое вхождение термина. *Курсив* также применяется для адресов электронной почты, веб-сайтов, FTP-сайтов, имен файлов и каталогов, а также групп новостей. Кроме того, в этой книге *курсивом* выделяются имена Java-классов, чтобы их нельзя было спутать с именами JavaScript.

Моноширинный шрифт

Применяется для форматирования кода JavaScript, листингов HTML и вообще всего, что непосредственно набирается на клавиатуре при программировании.

Моноширинный курсив

Обозначает аргументы функций и элементы, которые в программе необходимо заменить на реальные значения.

Ошибки

Для улучшения будущих изданий и тиражей этой книги издательство O'Reilly & Associates просит сообщать о любых ошибках, неточностях, ложных или сбивающих с толку утверждениях и обычных опечатках. O'Reilly поддерживает веб-сайт, где можно найти список всех известных ошибок в этой книге:

http://www.oreilly.com/catalog/jscript4/errata/

На этой странице имеется ссылка на форму, заполнив которую можно отправить сообщение о найденной ошибке. Кроме того, можно сообщить об ошибках или задать вопросы по этой книге, послав письмо по электронной почте на адрес:

bookquestions@oreilly.com

Примеры в Сети

Примеры, приведенные в этой книге, можно загрузить с веб-сайта издательства, пройдя по ссылке *Examples* на странице:

http://www.oreilly.com/catalog/jscript4/

Комментарии и вопросы

Просьба присылать комментарии и вопросы по этой книге издателю:

O'Reilly & Associates, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

(800) 998-9938 (in the United States or Canada)

(707) 829-0515 (international/local)

(707) 829-0104 (fax)

Технические вопросы и комментарии по этой и другим книгам издательства O'Reilly можно также посылать по адресу:

bookquestions@oreilly.com

Дополнительная информация о книгах, конференциях, Resource Center и O'Reilly Network приведена на сайте издательства O'Reilly:

http://www.oreilly.com

Благодарности

Брендан Эйх (Brendan Eich) из Mozilla – один из создателей и главный новатор JavaScript. Я и другие программисты на JavaScript в огромном долгу перед ним за разработку JavaScript и за то, что в его сумасшедшем графике нашлось время для ответов на наши вопросы, причем он даже требовал еще вопросов. Кроме того, что Брендан терпеливо отвечал на мои многочисленные вопросы, он также прочитал первое и третье издания этой книги и дал очень полезные комментарии к ним.

Это руководство получило благословение первоклассных технических рецензентов, комментарии которых весьма способствовали улучшению этой книги. Валдемар Хорват (Waldermar Horwat) из Netscape рецензировал новый материал по JavaScript. Материал по W3C DOM прорецензирован Филиппом Ле Хегаре (Philippe Le Hegaret) из W3C, Питером-Паулем Кохом (Peter-Paul Koch), руководителем отдела программирования клиентских приложений в голландской компании Netlinq Framfab (*http://www.netlinqframfab.nl*), специализирующейся на интернет-консалтинге, разработке и внедрении цифровых технологий в Интернете и маркетинге, Диланом Шиманом (Dylan Schiemann) из SitePen (*http://www.sitepen.com*), а также независимым веб-разработчиком Джеффом Ятсом (Jeff Yates). Питер-Пауль и Джефф поддерживают веб-сайты, посвященные веб-дизайну и DOM – *http://www.xs4all.nl/~ppk/js/* и *http://www.pbwizard.com* соответственно. Джозеф Кесселман (Joseph Kesselman) из IBM Research ничего не рецензировал, но очень помог мне, отвечая на вопросы по W3C DOM.

Третье издание книги рецензировалось Бренданом Эйхом, Валдемаром Хорватом и Вайдуром Аппарао (Vidur Apparao) из Netscape, Германом Вентером

(Herman Venter) из Microsoft, двумя независимыми разработчиками JavaScript – Джеем Ходжесом (Jay Hodges) и Анжело Сиригосом (Angelo Sirigos). Дэн Шейфер (Dan Shafer) из CNET Builder.com выполнил некоторую предварительную работу по третьему изданию. Его материал не нашел применения в этом издании, но принадлежащие ему идеи и общие принципы принесли большую пользу. Норрис Бойд (Norris Boyd) и Скотт Фурман (Scott Furman) из Netscape, а также Скотт Айзекс (Scott Issacs) из Microsoft нашли время, чтобы поговорить со мной о будущем стандарте Document Object Model. И наконец, доктор Танкред Хиршманн (Dr. Tankred Hirschmann) показал глубокое понимание хитросплетений JavaScript 1.2.

Второе издание много выиграло от помощи и комментариев Ника Томпсона (Nick Thompson) и Ричарда Якера (Richard Yaker) из Netscape, доктора Шона Каценбергера (Dr. Shon Katzenberger), Ларри Салливана (Larry Sullivan) и Дэйва С. Митчелла (Dave C. Mitchell) из Microsoft, Линн Роллинс (Lynn Rollins) из R&B Communications. Первое издание рецензировалось Нилом Беркманом (Neil Berkman) из Bay Networks, Эндрю Шульманом (Andrew Schulman) и Терри Алленом (Terry Allen) из O'Reilly & Associates.

Эта книга также стала лучше благодаря многочисленным редакторам, которые над ней работали. Паула Фергюсон (Paula Ferguson) – редактор этого и третьего издания. Она тщательно просмотрела книгу, сделав ее более понятной и легкой для чтения. Френк Уиллисон (Frank Willison) редактировал второе издание, а Эндрю Шульман – первое.

И наконец, мои благодарности Кристи – как всегда и за очень многое.

Дэвид Флэнаган
сентябрь 2001

22

Совместное применение Java и JavaScript

В главе 14 мы уже говорили, что Netscape 3 и более поздние версии, а также Internet Explorer 3 и более поздние версии разрешают программам JavaScript читать и записывать открытые поля и вызывать открытые методы апплетов Java, встроенных в HTML-документы. Netscape обеспечивает взаимодействие JavaScript с апплетами Java посредством технологии, известной как LiveConnect. Internet Explorer, со своей стороны, рассматривает любой объект Java (включая апплеты) как элемент управления ActiveX и использует технологию сценариев ActiveX, позволяющую программам на JavaScript взаимодействовать с Java. Так как технология Netscape специально предназначена для связи JavaScript и Java, она имеет возможности, которые не предоставляет технология ActiveX в IE. Однако на практике эти две технологии вполне сравнимы. Материал этой главы базируется на Netscape LiveConnect, но ключевые возможности, описанные в ней, работают и в IE.[1]

В начале этой главы обсуждается применение JavaScript в сценариях апплетов Java, вызов кода JavaScript апплетами Java и то, как (только в Netscape) JavaScript позволяет работать непосредственно с системными классами Java. Затем описаны важные детали работы LiveConnect. Здесь предполагается, что вы имеете по крайней мере базовое представление о программировании на Java (см. книги «Java in a Nutshell»[2] Дэвида Флэнагана (David Flanagan) и «Learning Java» (Изучаем Java) Патрика Нимейера (Patrick Niemeyer) и Джонатана Кнудсена (Jonathan Knudsen), O'Reilly).

---

[1] Обратите внимание, что в Netscape 6 поддержка LiveConnect была слабой, но в Netscape 6.1 и более поздних версиях она реализована полностью.

[2] Дэвид Флэнаган «Java. Справочник». – Пер. с англ. – СПб: Символ-Плюс, 2004.

22.1. Применение Java-апплетов в сценариях

В главе 14 говорилось, что апплеты Java, встроенные в веб-страницу, становятся элементами массива `Document.applets[]`. А если в апплете указан атрибут `name` или `id`, то к апплету можно обращаться напрямую как к свойству объекта Document. Например, к апплету, созданному с помощью тега `<applet>` с атрибутом `name`, равным «chart», можно обратиться как к `document.chart`.

Открытые поля и методы любого апплета доступны JavaScript, как если бы они были свойствами и методами объекта JavaScript. Так, если апплет с именем chart определяет поле с именем `lineColor`, имеющее тип `String`, программа JavaScript может получать и устанавливать значение этого поля так:

```
var chartcolor = document.chart.lineColor;  // Чтение поля апплета
document.chart.lineColor = "#ff00ff";       // Установка поля апплета
```

JavaScript может даже читать и устанавливать значения полей, являющихся массивами. Предположим, что апплет chart определяет два метода, объявленные следующим образом (код Java):

```
public int numPoints;
public double[] points;
```

Программа JavaScript может использовать эти поля так:

```
for(var i = 0; i < document.chart.numPoints; i++)
    document.chart.points[i] = i*i;
```

Этот пример иллюстрирует сложный момент, относящийся к связи между Java и JavaScript: преобразование типов. Java – это строго типизированный язык с большим количеством отдельных элементарных типов. JavaScript слабо типизирован и имеет только один числовой тип. В предыдущем примере целое (integer) Java преобразуется в число JavaScript, а различные числа JavaScript – в значения типа `double` в Java. Чтобы эти значения были нужным образом преобразованы, выполняется очень много закулисной работы. Далее в этой главе преобразование типов данных рассматривается подробно.

Кроме получения и установки значений полей апплета Java, JavaScript может также вызывать методы апплета. Предположим, что апплет chart определяет метод с именем `redraw()`. Этот метод не требует аргументов и просто уведомляет апплет о том, что массив `points[]` был изменен, и апплет должен выполнить перерисовку. JavaScript может вызывать этот метод так же, как если бы это был метод JavaScript:

```
document.chart.redraw();
```

JavaScript может вызывать методы, принимающие аргументы и возвращающие значения. LiveConnect или технология сценариев ActiveX преобразуют значения аргументов JavaScript в допустимые значения Java, а возвращаемые значения – в допустимые значения JavaScript. Предположим, что апплет chart определяет следующие методы Java:

```
public void setDomain(double xmin, double xmax);
public void setChartTitle(String title);
public String getXAxisLabel();
```

Из JavaScript эти методы вызываются так:

```
document.chart.setDomain(0, 20);
document.chart.setChartTitle("y = x*x");
var label = document.chart.getXAxisLabel();
```

И наконец, методы Java могут возвращать объекты Java, а JavaScript может получать и устанавливать значения открытых полей и вызывать открытые методы этих объектов. Предположим, что апплет Java определяет метод с именем `getXAxis()`, возвращающий объект Java, который является экземпляром класса Axis, и метод `setYAxis()`, принимающий тот же тип в качестве аргумента. Теперь предположим, что в классе Axis есть метод `setTitle()`. Мы можем использовать эти методы в таком коде JavaScript:

```
var xaxis = document.chart.getXAxis();  // Получаем объект Axis
var newyaxis = xaxis.clone();           // Делаем его копию
newyaxis.setTitle("Y");                 // Вызываем метод объекта...
document.chart.setYAxis(newyaxis);      // и передаем его другому методу
```

В случае применения JavaScript для вызова методов объекта Java есть одна сложность. Java допускает существование нескольких методов с одинаковым именем, если они имеют различные типы аргументов. Например, объект Java может объявить следующие два метода:

```
public String convert(int i);     // Преобразует целое в строку
public String convert(double d);  // Преобразует число с плавающей точкой
```

JavaScript имеет только один числовой тип и не различает целые значения и значения с плавающей точкой, поэтому когда вы с помощью JavaScript передаете число методу с именем «convert», интерпретатор не может понять, какой метод вы собираетесь вызвать. На практике эта проблема возникает редко, и ее обычно можно обойти, просто переименовав методы. Последние версии LiveConnect (в Netscape 6.1 и более поздних версиях) также позволяют устранить такие неоднозначности, включив типы аргумента в имя метода. Так, если два метода, приведенные выше, определены в `document.applets[0]`, их можно отличить следующим образом:

```
var iconvert = document.applets[0]["convert(int)"];
                                        // Получаем целочисленный метод
iconvert(3);  // Вызываем метод
```

22.2. Применение JavaScript из Java

Узнав, как управлять кодом Java из JavaScript, мы можем перейти к противоположной задаче: управлению кодом JavaScript из Java. Такое управление в основном осуществляется через класс Java *netscape.javascript.JSObject*,

представляющий объект JavaScript внутри программы на Java. Возможности связи JavaScript с Java, описанные в предыдущем разделе, обычно прекрасно работают и в Netscape и в Internet Explorer. В отличие от этого, приемы связи Java с JavaScript, описанные здесь, поддерживаются не так хорошо, и ошибки встречаются как в Netscape, так и в IE.

22.2.1. Класс JSObject

Любое взаимодействие Java с JavaScript выполняется через экземпляр класса *netscape.javascript.JSObject*. Экземпляр этого класса представляет собой обертку вокруг одиночного объекта JavaScript. Класс определяет методы, позволяющие читать и писать значения свойств и элементы массива объекта JavaScript, а также вызывать методы объекта. Вот описание этого класса:

```
public final class JSObject extends Object {
    // Статический метод для получения объекта JSObject для окна броузера,
    // в котором содержится объект.
    public static JSObject getWindow(java.applet.Applet applet);
    public Object getMember(String name);              // Чтение свойства объекта
    public Object getSlot(int index);                  // Чтение элемента массива
    public void setMember(String name, Object value);  // Установка свойства объекта
    public void setSlot(int index, Object value);      // Установка элемента массива
    public void removeMember(String name);              // Удаление свойства
    public Object call(String methodName, Object args[]); // Вызов метода
    public Object eval(String s);                       // Вычисление строки
    public String toString();                          // Преобразование в строку
    protected void finalize();
}
```

Все объекты JavaScript образуют иерархию, вершиной которой является текущее окно броузера, поэтому объекты JSObject должны также образовывать иерархию. Для того чтобы взаимодействовать с любыми объектами JavaScript, апплет Java должен сначала получить объект JSObject, представляющий окно броузера (или фрейм), в котором находится апплет. В классе JSObject не определен метод-конструктор, поэтому мы не можем просто создать нужный нам JSObject. Вместо этого необходимо вызвать статический метод `getWindow()`. Метод, которому передается ссылка на апплет, возвращает JSObject, представляющий окно броузера, которое содержит апплет. Следовательно, любой апплет, взаимодействующий с JavaScript, включает строку, которая выглядит примерно так:

```
JSObject jsroot = JSObject.getWindow(this);  // "this" - это сам апплет
```

Получив JSObject, ссылающийся на корневое окно в иерархии объектов JavaScript, можно посредством методов экземпляра JSObject читать значения свойств объекта JavaScript, который он представляет. Большинство этих свойств имеют значения, которые сами являются объектами JavaScript, поэтому можно продолжить процесс и также прочитать их свойства. Метод `getMember()` объекта JSObject возвращает значение свойства с данным име-

нем, а метод getSlot() – значение элемента массива с данным номером указанного объекта JavaScript. Вот как эти методы применяются:

```
import netscape.javascript.JSObject;  // Это должно быть в начале файла
    ...
JSObject jsroot = JSObject.getWindow(this);                    // self
JSObject document = (JSObject) jsroot.getMember("document");   //  .document
JSObject applets = (JSObject) document.getMember("applets");   //    .applets
Applet applet0 = (Applet) applets.getSlot(0);                  //      [0]
```

Здесь следует обратить внимание на два момента. Во-первых, методы getMember() и getSlot() оба возвращают значение типа «Object», которое, как правило, должно быть преобразовано в некоторое более конкретное значение, такое как JSObject. Во-вторых, значение, прочитанное из слота 0 массива applets, может быть преобразовано в Applet, а не в JSObject. Дело в том, что элементы массива applets[] в JavaScript – это объекты JavaScript, представляющие объекты Applet из Java. Когда Java читает объект JavaObject из JavaScript, обертка этого объекта «снимается» и возвращается объект Java, содержащийся в ней (в данном случае Applet). Преобразование данных, происходящее через интерфейс JSObject, описывается далее в этой главе.

Класс JSObject также поддерживает методы для установки значений свойств и элементов массивов объектов JavaScript. Методы setMember() и setSlot() соответствуют методам getMember() и getSlot(). Они устанавливают значение свойства с данным именем или элемента массива с указанным номером. Заметьте, однако, что устанавливаемое значение должно быть объектом Java. Значение элементарного типа устанавливается с помощью соответствующего класса-обертки Java: например, используется объект Integer вместо значения типа int. И наконец, метод removeMember() позволяет удалить значение указанного свойства из объекта JavaScript.

Кроме чтения и установки значений свойств и элементов массивов из объектов JavaScript класс JSObject позволяет вызывать методы объектов JavaScript. Метод call() класса JSObject вызывает указанный метод заданного объекта JavaScript и передает массив объектов Java в качестве аргументов этого метода. Как мы видели при установке значений свойств JavaScript, невозможно передать методу JavaScript в качестве аргументов значения элементарных типов Java; тут нужны соответствующие объектные типы Java. Например, посредством метода call() можно открыть новое окно броузера:

```
public JSObject newwin(String url, String window_name)
{
    Object[] args = { url, window_name };
    JSObject win = JSObject.getWindow(this);
    return (JSObject) win.call("open", args);
}
```

В классе JSObject имеется еще один важный метод: eval(). Этот метод Java работает так же, как функция JavaScript с тем же именем – он исполняет строку, содержащую код JavaScript. Работать с eval() часто намного проще,

чем с другими методами класса JSObject. Код передается в виде строки, поэтому можно использовать строковые представления нужных типов данных и не надо преобразовывать элементарные типы Java в соответствующие им объектные типы. Сравните, например, следующие две строки кода, устанавливающие свойства главного окна броузера:

```
jsroot.setMember("i", new Integer(0));
jsroot.eval("self.i = 0");
```

Вторая строка более понятна. Как еще один пример рассмотрим применение `eval()` для записи в определенный фрейм, отображаемый в окне броузера:

```
JSObject jsroot = JSObject.getWindow(this);
jsroot.eval("parent.frames[1].document.write('Привет от Java!')");
```

Сделать то же самое без метода `eval()` намного сложнее:

```
JSObject jsroot = JSObject.getWindow(this);
JSObject parent = (JSObject) jsroot.getMember("parent");
JSObject frames = (JSObject) parent.getMember("frames");
JSObject frame1 = (JSObject) frames.getSlot(1);
JSObject document = (JSObject) frame1.getMember("document");
Object[] args = { "Hello from Java!" };
document.call("write", args);
```

22.2.2. Применение JSObject в апплетах

Пример 22.1 иллюстрирует применение метода `init()` апплета, взаимодействующего с JavaScript при помощи LiveConnect.

Пример 22.1. Использование JavaScript из метода апплета

```
import netscape.javascript.*

public void init()
{
    // Получение JSObject, представляющего окно броузера, открываемое апплетом
    JSObject win = JSObject.getWindow(this);

    // Запускаем код  JavaScript с помощью eval().
    // Будьте осторожны с вложенными кавычками!
    win.eval("alert('Апплет CPUHog теперь работает на вашем компьютере. " +
       "Вы можете заметить, что ваша программа несколько замедлилась.');");
}
```

Для того чтобы можно было запускать любой апплет, его необходимо скомпилировать, а затем встроить в HTML-файл. Когда апплет взаимодействует с JavaScript, для выполнения обоих шагов требуются особые инструкции.

22.2.2.1. Компиляция апплетов, использующих класс JSObject

Любой апплет, взаимодействующий с JavaScript, использует класс *netscape.javascript.JSObject*. Следовательно, для того чтобы скомпилировать такой апплет, компилятор Java должен знать, где найти определение этого класса. Поскольку класс определен и поставляется Netscape, а не Sun, компилятор *javac* от Sun о нем не знает. Здесь рассказано, как заставить компилятор найти этот нужный класс. Тем, кто не работает с Sun JDK, надо обратиться за подробностями к документации от поставщика компилятора или среды разработки Java, так как им придется выполнять немного другие действия.

Чтобы сообщить компилятору JDK, где искать классы, надо установить переменную окружения `CLASSPATH`. Эта переменная окружения задает список каталогов и JAR-файлов (или ZIP-файлов), в которых компилятор должен искать определения классов (в дополнение к стандартному каталогу системных классов). Сложность состоит в том, чтобы выяснить, в каком файле в системе находится определение класса *netscape.javascript.JSObject*. Для Netscape 6.1 это файл *plugins/java2/javaplugin.jar* внутри каталога установки Netscape, а для Netscape 4 – файл *java/classes/java40.jar* в каталоге установки Netscape. Так, в Netscape 4 на платформе Windows файл *java40.jar*, скорее всего, находится в *C:\Program Files\Netscape\Communicator\Program\Java\Classes\java40.jar*.

Для Internet Explorer искомое определение класса обычно находится в одном из ZIP-файлов в каталоге *c:\Windows\Java\Packages*. Неприятность в том, что здесь находится группа файлов, имена которых совершенно невнятны и изменяются от версии к версии. Как правило, нужен самый большой из них. Убедиться, что он содержит файл *netscape/javascript/JSObject.class*, нетрудно, – например при помощи утилиты unzip.

Найдя нужный JAR- или ZIP-файл, можно сообщить компилятору об этом, установив переменную среды `CLASSPATH`. Для систем Unix установите ее так:

```
setenv CLASSPATH .:/usr/local/netscape/plugins/java2/javaplugin.jar
```

А для Windows так:

```
set CLASSPATH=.;C:\Windows\Java\Packages\5fpnnz7t.zip
```

Установив `CLASSPATH`, можно компилировать апплет с помощью *javac*.

22.2.2.2. Атрибут mayscript

Есть еще одно требование для запуска апплета, взаимодействующего с JavaScript. В качестве меры безопасности апплету не разрешается использовать JavaScript, если только автор веб-страницы (который может не быть автором апплета) явно не даст это разрешение (для чего необходимо включить новый атрибут `mayscript` в тег `<applet>` в HTML-файле).

В примере 22.1 показан фрагмент апплета, использующего JavaScript для вывода диалогового окна с предупреждением. Успешно скомпилировав этот апплет, можно включить его в HTML-файл следующим образом:

```
<applet code="CPUHog.class" width="300" height="300" mayscript></applet>
```

Без атрибута `mayscript` апплету нельзя будет использовать класс JSObject.

22.3. Непосредственное использование классов Java

Как говорилось в двух предыдущих разделах, и Netscape и Internet Explorer позволяют коду JavaScript взаимодействовать с апплетом Java, а апплету Java – взаимодействовать с JavaScript. Технология Netscape LiveConnect также позволяет программам на JavaScript создавать собственные объекты Java и использовать их, даже в отсутствие каких-либо апплетов. Internet Explorer не имеет какой-либо аналогичной возможности.

В Netscape объект `Packages` предоставляет доступ ко всем пакетам Java, о которых знает Netscape. Выражение `Packages.java.lang` ссылается на пакет *java.lang*, а выражение `Packages.java.lang.System` – на класс *java.lang.System*. Для удобства вместо `Packages.java` достаточно написать `java`. В Netscape код JavaScript может вызывать статический метод класса *java.lang.System* так:

```
// Вызов статического метода Java System.getProperty()
var javaVersion = java.lang.System.getProperty("java.version");
```

Такое применение не ограничено системными классами, поскольку LiveConnect позволяет создавать новые экземпляры классов Java посредством оператора JavaScript `new` (точно так же мы бы делали это в Java). В примере 22.2 показан код JavaScript, использующий стандартные классы Java (на самом деле код JavaScript выглядит практически так же, как код Java) для вывода окна и отображения в нем текста. Результат его работы показан на рис. 22.1.

***Рис. 22.1.** Окно Java, созданное из JavaScript*

Пример 22.2. Использование встроенных Java-классов

```
var f = new java.awt.Frame("Hello World");
var ta = new java.awt.TextArea("hello, world", 5, 20);
f.add("Center", ta);
f.pack();
f.show();
```

Код этого примера создает простой пользовательский интерфейс Java. Однако в нем отсутствует какая-либо обработка событий или взаимодействие с пользователем. Подобная программа ограничена только выдачей результата, так как не содержит никакого способа уведомления кода JavaScript о взаимодействии пользователя с окном Java. Возможно, хотя и затруднительно, определить пользовательский интерфейс Java, реагирующий на события, средствами JavaScript. В Java 1.1 и более поздних версиях уведомление о событии выполняется путем вызова метода объекта EventListener. Апплеты Java могут исполнять произвольные строки кода JavaScript, поэтому можно определить класс Java, реализующий соответствующий интерфейс EventListener и вызывающий указанную строку кода JavaScript при уведомлении о возникновении события. Написав апплет с методом, позволяющим создавать такие объекты EventListener, можно средствами JavaScript формировать графические интерфейсы Java, включающие обработчики событий, определенные в JavaScript.

Заметим, что LiveConnect не дает полного доступа к Java-системе; другими словами, многое нельзя сделать с помощью LiveConnect. Например, LiveConnect не позволяет определять новые классы или подклассы Java из JavaScript, а также не дает возможности создавать массивы Java.[1] Кроме этих причин доступ к стандартным классам Java ограничен соображениями безопасности. Недоверенная программа на JavaScript не может работать, например, с классом *java.io.File*, так как это даст возможность читать, писать и удалять файлы системы. Недоверенный код JavaScript может использовать Java только теми способами, которыми это делают недоверенные апплеты.

22.4. Типы данных LiveConnect

Для того чтобы понять, что LiveConnect делает для соединения JavaScript с Java, необходимо разобраться в типах данных JavaScript, используемых в LiveConnect. Эти типы данных описаны в следующих разделах. Хотя в Internet Explorer реализована другая технология, понимание механизма работы LiveConnect поможет вам понять и работу IE. Некоторые из типов данных LiveConnect, описанные здесь, имеют аналоги в IE.

22.4.1. Класс JavaPackage

Пакет в Java представляет собой коллекцию связанных классов Java. Класс JavaPackage – это тип данных JavaScript, представляющий пакет Java. Свойства JavaPackage – это классы, которые содержатся в пакете (классы представляются классом JavaClass, который мы скоро рассмотрим), а также любые другие пакеты, которые содержатся в этом пакете. У класса JavaPackage есть ограничение: вы не можете использовать цикл `for/in` в Ja-

[1] Программы JavaScript могут создавать массивы опосредованно, с помощью метода `java.lang.reflect.Array.newInstance()` из Java 1.1.

vaScript для получения полного списка всех пакетов и классов, содержащихся в JavaPackage. Это ограничение является результатом внутреннего ограничения виртуальной машины Java.

Все объекты JavaPackage содержатся в родительском JavaPackage; свойство объекта Window с именем `Package` является объектом JavaPackage самого верхнего уровня, который служит вершиной этой иерархии пакетов. Он имеет такие свойства, как `java`, `sun` и `netscape`, которые являются объектами JavaScript, представляющими различные иерархии доступных броузеру классов Java. Например, объект JavaPackage `Package.java` содержит объект JavaPackage `Packages.java.awt`. Для удобства во всех объектах Window также имеются свойства `java`, `sun` и `netscape`, являющиеся сокращениями для `Packages.java`, `Packages.sun` и `Packages.netscape`. Следовательно, вместо `Packages.java.awt` вы можете набирать просто `java.awt`.

Продолжая пример, `java.awt` – это объект JavaPackage, содержащий объекты JavaClass, такие как `java.awt.Button`, который представляет класс `java.awt.Button`. Но в нем содержится другой объект JavaPackage, `java.awt.image`, представляющий пакет *java.awt.image* из Java.

Как можно видеть, схема назначения имен для свойств иерархии JavaPackage отражает схему назначения имен для пакетов Java. Однако обратите внимание, что имеется одно большое различие между классом JavaPackage и реальным пакетом Java, который он представляет. Пакеты в Java являются коллекциями классов, а не коллекциями других пакетов, то есть *java.lang* является именем пакета Java, а *java* не является. Поэтому объект JavaPackage с именем `java` фактически не представляет пакет Java – это просто удобное место в иерархии пакетов для других объектов JavaPackage, представляющих реальные пакеты Java.

На большинстве систем Java-классы устанавливаются в файлах в иерархии каталогов, соответствующих именам их пакетов. Например, класс *java.lang.String* хранится в файле *java/lang/String.class*. Фактически этот файл обычно содержится в ZIP-файле, но иерархия каталогов все равно сохраняется внутри архива. Следовательно, вместо того чтобы считать объект JavaPackage представлением пакета Java, вам может быть понятнее представлять его как каталог или подкаталог в иерархии каталогов классов Java.

Класс JavaPackage имеет несколько недостатков. В LiveConnect нет способа заранее определить, ссылается ли свойство JavaPackage на класс Java или на другой пакет Java, поэтому JavaScript предполагает, что это класс, и пытается загрузить класс. В результате, когда вы используете выражение вроде `java.awt`, LiveConnect сначала ищет файл класса *java/awt.class*. Он даже может искать этот класс в сети, приводя к возникновению ошибки «404 Файл не найден» на веб-сервере. Если LiveConnect не находит класс, он предполагает, что свойство ссылается на пакет, но нет способа убедиться, что пакет действительно существует, и в нем имеются реальные классы. Из этого следует еще один недостаток: если вы неправильно напишете имя класса, LiveConnect спокойно посчитает его именем пакета и не сообщит вам, что класс, который вы хотите использовать, не существует.

22.4.2. Класс JavaClass

Класс JavaClass – это тип данных JavaScript, представляющий класс Java. Объект JavaClass не имеет собственных свойств – все его свойства представляют (и имеют те же имена), что и открытые статические поля и методы представляемого им класса Java. Эти открытые статические поля и методы иногда называются *полями класса* и *методами класса*, указывая на их связь с классом, а не с экземпляром объекта. В отличие от класса JavaPackage, JavaClass допускает использование цикла `for/in` для перебора его свойств. Заметьте, что объект JavaClass не имеет свойств, представляющих поля и методы экземпляра класса Java – индивидуальные экземпляры класса Java представлены классом JavaObject, описанным в следующем разделе.

Как мы видели ранее, объекты JavaClass содержатся в объектах JavaPackage. Например, `java.lang` – это объект JavaPackage, в котором содержится свойство `System`. Следовательно, `java.lang.System` – это объект JavaClass, представляющий класс Java *java.lang.System*. Этот объект JavaClass, в свою очередь, имеет такие свойства, как `out` и `in`, представляющие статические поля класса *java.lang.System*. Подобным образом вы можете использовать JavaScript для обращения к любому из стандартных системных классов Java. Класс *java.lang.Double*, например, имеет имя `java.lang.Double` (или `Packages.java.lang.Double`), а класс *java.awt.Button* – `java.awt.Button`.

Другой способ получения объекта JavaClass в JavaScript – использование функции `getClass()`. Имея любой объект JavaObject, вы можете получить объект JavaClass, представляющий класс этого объекта Java, передав JavaObject в качестве аргумента функции `getClass()`.[1]

Имея объект JavaClass, вы можете использовать его несколькими способами. Класс JavaClass реализует функциональность LiveConnect, позволяющую программам JavaScript читать и устанавливать значения открытых статических полей классов Java и вызывать открытые статические методы классов Java. Например, `java.lang.System` – это объект JavaClass. Мы можем прочитать значение статического поля `java.lang.System` следующим образом:

```
var java_console = java.lang.System.out;
```

Аналогично можно вызвать статический метод класса `java.lang.System`:

```
var java_version = java.lang.System.getProperty("java.version");
```

Вспомните, что Java – типизированный язык – все поля и аргументы методов имеют тип. Если попытаться установить неправильное значение поля или передать аргумент неверного типа, генерируется исключение. (В версиях JavaScript до 1.5 возникает ошибка JavaScript.)

Класс JavaClass предоставляет еще одну важную возможность – использования объектов JavaClass с оператором `new` JavaScript для создания новых эк-

[1] Не путайте функцию `getClass()` JavaScript, возвращающую объект JavaClass, с методом `getClass()` из Java, возвращающим объект `java.lang.Class`.

экземпляров классов Java – то есть для создания объектов JavaObject, что совпадает с синтаксисом создания объектов в JavaScript (и с синтаксисом Java):

```
var d = new java.lang.Double(1.23);
```

И наконец, создав подобным образом объект JavaObject, мы можем вернуться к функции `getClass()` и показать пример ее использования:

```
var d = new java.lang.Double(1.23);    // Создание JavaObject
var d_class = getClass(d);             // Получение JavaClass для JavaObject
if (d_class == java.lang.Double) ...;  // Результатом этого сравнения будет истина
```

Работая подобным образом со стандартными системными классами, мы, как правило, непосредственно используем имя системного класса, а не вызываем `getClass()`. Функция `getClass()` более полезна для получения класса пользовательского объекта, например экземпляра апплета.

Можно не обращаться к JavaClass с помощью длинных выражений вроде `java.lang.Double`, а определить переменную для упрощенного доступа к классу:

```
var Double = java.lang.Double;
```

Этим вы имитируете инструкцию `import` Java и можете увеличить эффективность вашей программы, так как LiveConnect не придется искать свойство `lang` в `java` и свойство `Double` в `java.lang`.

22.4.3. Класс JavaObject

Класс JavaObject – это тип данных JavaScript, представляющий объект Java. Классы JavaObject и JavaClass во многом похожи. Как JavaClass, JavaObject не имеет собственных свойств – все его свойства представляют (и имеют те же имена) открытые поля экземпляра и открытые методы экземпляра представляемого им объекта Java. Как и для JavaClass, в JavaScript можно в цикле `for/in` осуществить перебор всех свойств объекта JavaObject. Класс JavaObject реализует функциональность LiveConnect, позволяющую получать и устанавливать значения поля экземпляра и вызывать открытые методы объекта Java.

Пусть `d` – это JavaObject, представляющий экземпляр класса *java.lang.Double*, тогда метод этого объекта Java вызывается из кода JavaScript так:

```
n = d.doubleValue();
```

Аналогично, мы видели, что класс *java.lang.System* имеет статическое поле `out`, ссылающееся на объект класса Java *java.io.PrintStream*. В JavaScript на соответствующий объект JavaObject можно ссылаться следующим образом:

```
java.lang.System.out
```

и вызвать метод этого объекта:[1]

```
java.lang.System.out.println("Hello world!");
```

[1] Результат этой строки появляется не в веб-броузере, а в консоли Java. В Netscape 6, чтобы увидеть это окно, выберите Tasks → Tools → Java Console.

Объект JavaObject также позволяет читать и записывать значения открытых полей экземпляра представляемого им объекта Java. Однако ни класс *java.lang.Double*, ни класс *java.io.PrintStream*, использованные в предыдущих примерах, не имеют открытых полей экземпляра. Но предположим, что мы используем JavaScript для создания экземпляра класса *java.awt.Rectangle*:

```
r = new java.awt.Rectangle();
```

Затем мы можем прочитать и записать значения открытых свойств экземпляра с помощью следующего кода JavaScript:

```
r.x = r.y = 0;
r.width = 4;
r.height = 5;
var perimeter = 2*r.width + 2*r.height;
```

Изящество LiveConnect состоит в том, что он позволяет использовать объект Java `r` точно так же, как если бы это был объект JavaScript. Однако требуется некоторая осторожность: `r` – это JavaObject, и он не ведет себя точно так же, как обычные объекты JavaScript. Различия между ними будут описаны далее. Также помните, что в отличие от JavaScript, поля объектов Java и аргументы их методов являются типизированными. Если вы укажете значения JavaScript неверных типов, вы получите ошибки или исключения JavaScript.

В Netscape 6.1 и более поздних версиях класс JavaObject делает методы доступными по имени и по имени с указанием типа аргумента, что может быть полезно, когда имеется несколько методов, имеющих одинаковое имя, но требующих различные типы аргументов. Как мы видели ранее в этой главе, если JavaObject `o` обозначает объект, имеющий два метода с именем «convert», свойство `convert` объекта `o` может ссылаться на любой из этих методов. Однако в последних версиях LiveConnect объект `o` также определяет свойства, включающие типы аргументов, и вы можете указать, какая версия метода вам нужна, включив информацию о типах:

```
var iconvert = o["convert(int)"];  // Получаем требуемый метод
iconvert(3);                       // Вызываем его
```

Так как имя свойства включает скобки, вы не можете использовать для доступа к нему обычную нотацию с оператором «точка» (.) и должны указать его в виде строки в квадратных скобках. Тип JavaClass предоставляет такую возможность и для перегруженных статических методов.

22.4.4. Класс JavaArray

Последний тип данных LiveConnect для JavaScript – это класс JavaArray. Как вы можете предположить, экземпляры этого класса представляют массивы Java и предоставляют функциональность LiveConnect, позволяющую JavaScript читать элементы массивов Java. Как и массивы JavaScript (и массивы Java), объект JavaArray имеет свойство `length`, которое задает количество содержащихся в нем элементов. Элементы объекта JavaArray читаются с помощью стандартного оператора JavaScript для доступа к массиву – `[]`.

Кроме того, их можно перебрать с помощью цикла `for/in`. Вы можете использовать объекты JavaArray для доступа к многомерным массивам (фактически массивам массивов), как в JavaScript или в Java.

Предположим, что мы создаем экземпляр класса *java.awt.Polygon*:

```
p = new java.awt.Polygon();
```

В объекте JavaObject `p` имеются свойства `xpoints` и `ypoints`, являющиеся объектами JavaArray, которые представляют массивы Java, состоящие из целых значений. (Чтобы узнать имена и типы этих свойств, посмотрите документацию по *java.awt.Polygon* в справочном руководстве по Java.) Мы можем использовать эти объекты JavaArray для случайной инициализации объекта Polygon с помощью следующего кода:

```
for(var i = 0; i < p.xpoints.length; i++)
    p.xpoints[i] = Math.round(Math.random()*100);
for(var i = 0; i < p.ypoints.length; i++)
    p.ypoints[i] = Math.round(Math.random()*100);
```

22.4.5. Методы Java

Классы JavaClass и JavaObject позволяют нам вызывать соответственно статические методы и методы экземпляра. В Netscape 3 для внутреннего представления методов Java применялся объект JavaMethod. Однако в Netscape 4 методы Java стали просто обычными методами, как методы встроенных объектов JavaScript, таких как String и Date.

Работая с методами Java, помните, что им требуется фиксированное количество аргументов фиксированных типов. Если количество или тип переданных аргументов неправильно, вы получите ошибку JavaScript.

22.5. Преобразование данных в LiveConnect

Java – строго типизированный язык с относительно большим количеством типов данных, а JavaScript – нетипизированный язык с относительно небольшим количеством типов. Вот почему одной из главных обязанностей LiveConnect является преобразование данных. Когда JavaScript устанавливает значения поля класса Java или экземпляра или передает аргумент в метод Java, значение JavaScript должно быть преобразовано в эквивалентное значение Java, а когда JavaScript читает поле класса Java или экземпляра или получает значение, возвращаемое методом Java, это значение Java должно быть преобразовано в совместимое значение JavaScript.[1]

[1] Кроме того, преобразование данных может произойти, когда Java читает или устанавливает значение поля JavaScript или вызывает метод JavaScript. Эти преобразования, однако, выполняются по-другому и описаны в этой главе позднее – там же, где и вызов JavaScript из Java. Сейчас мы обсуждаем только преобразование данных, выполняемое, когда код JavaScript взаимодействует с Java, а не наоборот.

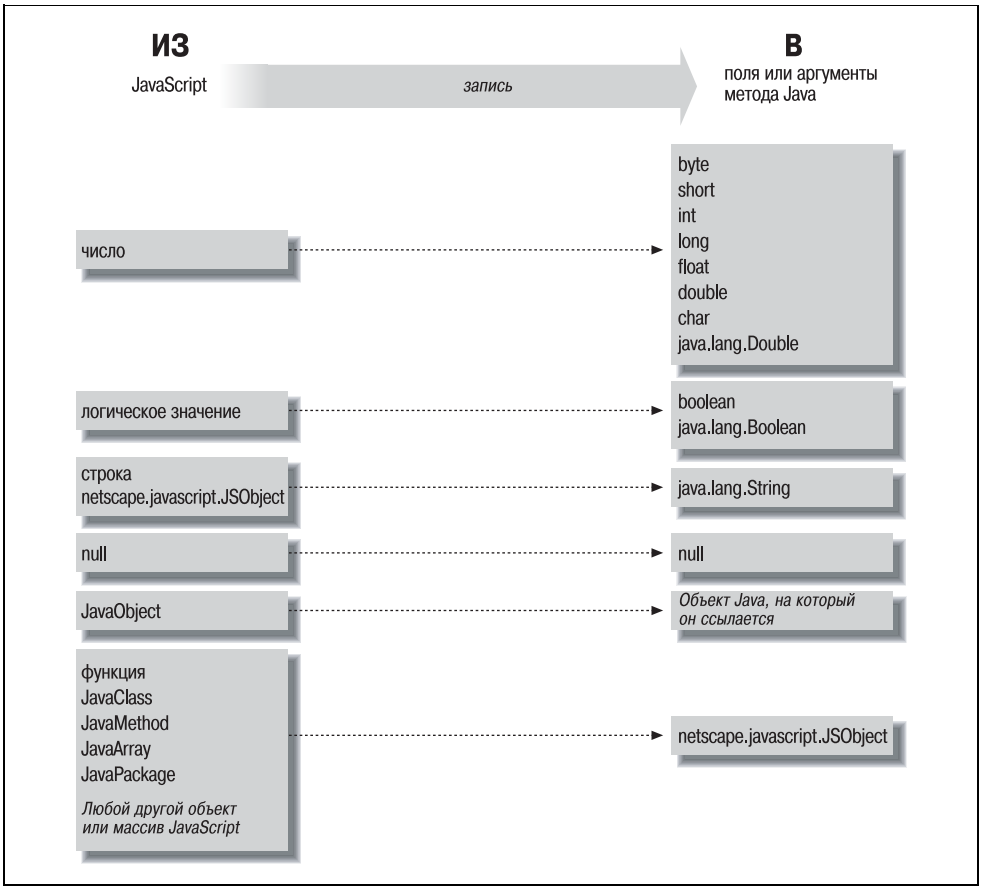

Рис. 22.2. *Преобразование данных, выполняемое JavaScript при записи значений Java*

Преобразование данных, выполняемое при записи и чтении значений Java из JavaScript, показано соответственно на рис. 22.2 и 22.3.

Обратите внимание на следующие моменты в преобразовании данных, иллюстрируемые рис. 22.2:

- На рис. 22.2 не показаны все возможные преобразования между типами JavaScript и Java. Дело в том, что перед преобразованием из JavaScript в Java могут происходить внутренние преобразования JavaScript. Например, если вы передаете число JavaScript методу Java, ожидающему аргумент типа *java.lang.String*, JavaScript сначала преобразует число в строку JavaScript, которая затем может быть преобразована в строку Java.
- Число JavaScript может быть преобразовано в любой элементарный числовой тип Java. Фактическое выполняемое преобразование зависит, конечно, от типа поля Java или аргумента метода, которому передается значение.

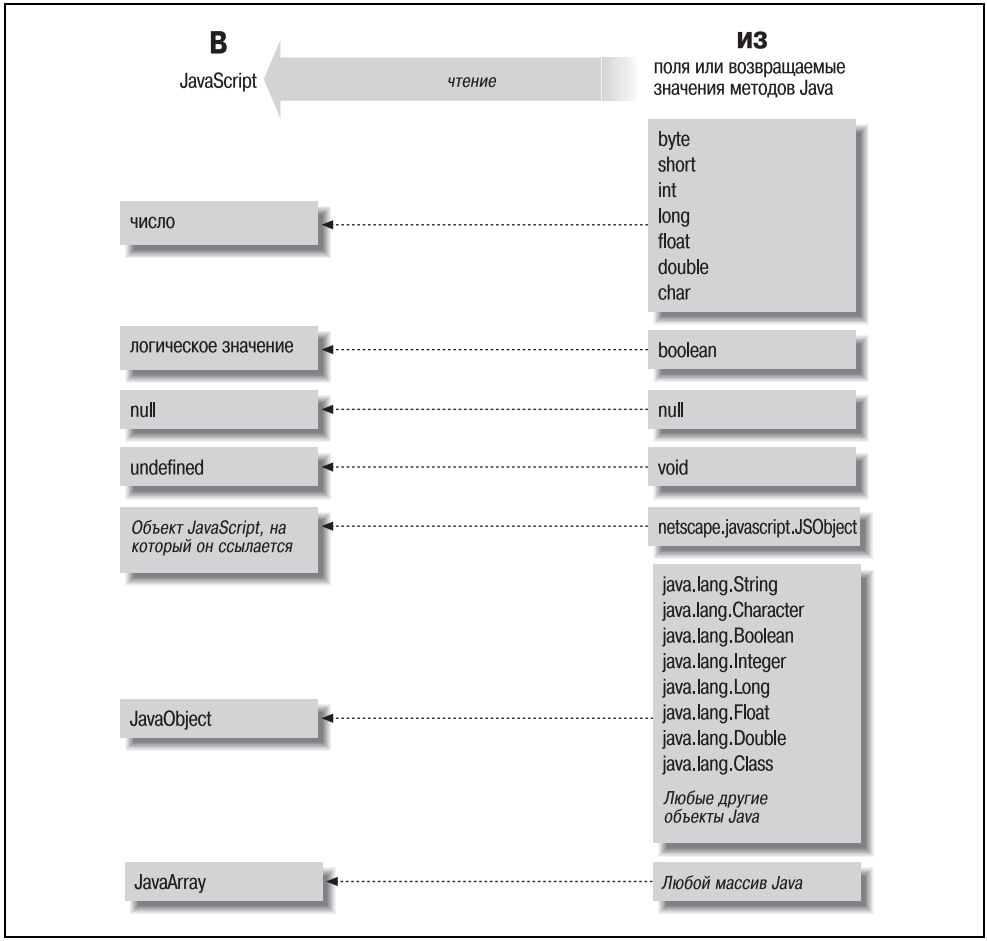

Рис. 22.3. *Преобразование данных, выполняемое JavaScript при чтении значений Java*

Обратите внимание, что при этом может теряться точность, если, например, передается большое число полю Java с типом `short` или значение с плавающей точкой целочисленному типу Java.

- Число JavaScript может быть также преобразовано в экземпляр класса Java *java.lang.Double*, но не в экземпляр другого схожего класса, такого как *java.lang.Integer* или *java.lang.Float*.
- В JavaScript нет представления для символьных данных, поэтому число JavaScript может также быть преобразовано в элементарный Java-тип `char`.
- JavaObject в JavaScript при передаче в Java освобождается от «обертки», то есть преобразуется в объект Java, который он представляет. Однако заметьте, что объекты JavaClass в JavaScript не преобразуются в экземпляры *java.lang.Class*, как можно было ожидать.

- Массивы JavaScript не преобразуются в массивы Java.

Также отметим следующие моменты в отношении преобразований, изображенных на рис. 22.3:

- Так как JavaScript не имеет типа для символьных данных, элементарный тип Java `char` преобразуется в число JavaScript, а не в строку, как можно было бы ожидать.
- Экземпляр *java.lang.Double*, *java.lang.Integer* или другого аналогичного класса Java не преобразуется в число JavaScript. Как любой объект Java, он преобразуется в JavaScript в объект JavaObject.
- Строка Java является экземпляром *java.lang.String*, поэтому, как любой другой объект Java, преобразуется в JavaObject, а не в реальную строку JavaScript.
- Любой массив Java преобразуется в JavaScript в объект JavaArray.

22.5.1. Объекты-обертки

Еще одно понятие, связанное с рис. 22.2 и 22.3, – это объекты-обертки. Если преобразования между большинством элементарных типов JavaScript и Java возможны, то преобразования между объектными типами – как правило, нет. Поэтому в LiveConnect определяется для JavaScript объект JavaObject – он представляет объект Java, который не может быть непосредственно преобразован в объект JavaScript. В каком-то смысле JavaObject – это обертка JavaScript вокруг объекта Java. Когда JavaScript читает данные Java (поле или возвращаемое методом значение), любые объекты Java упаковываются, и JavaScript видит объект JavaObject.

То же самое происходит, когда объект JavaScript записывается в поле Java или передается в метод Java. Нет способа преобразовать объект JavaScript в объект Java, поэтому объект упаковывается. Java-«обертка» для объекта JavaScript – это класс Java *netscape.javascript.JSObject*.

Становится еще интереснее, когда объекты-обертки передаются обратно. Если JavaScript записывает JavaObject в поле Java или передает его методу Java, LiveConnect сначала распаковывает объект, преобразуя JavaObject обратно в объект Java, который он представляет. Аналогично, если JavaScript читает поле Java или получает значение, возвращаемое методом Java, являющееся экземпляром *netscape.javascript.JSObject*, этот JSObject также распаковывается для извлечения и возвращения исходного объекта JavaScript.

22.5.2. Преобразование данных LiveConnect в Netscape 3

В Netscape 3 имеется ошибка в способе, которым LiveConnect преобразует значения Java в значения JavaScript: значение элементарного поля объекта Java возвращается в виде объекта JavaScript, а не в виде элементарного значения JavaScript. Например, если JavaScript читает значение поля типа `int`,

LiveConnect в Netscape 3 преобразует это значение в объект Number, а не в элементарное числовое значение. Аналогично, LiveConnect преобразует значение полей Java с типом `boolean` в объекты JavaScript Boolean, а не в элементарные логические значения. Обратите внимание, что эта ошибка возникает только при чтении данных из полей Java. Она не возникает, когда LiveConnect преобразует значение, возвращаемое методом Java.

Объекты Number и Boolean ведут себя почти (но не точно) как элементарные числовые и логические значения. Отличие в том, что объекты Number, как все объекты JavaScript, используют оператор + для конкатенации строк, а не для сложения. И выполнение кода, подобного приведенному ниже, использующего LiveConnect в Netscape 3, может дать неожиданный результат:

```
var r = new java.awt.Rectangle(0,0,5,5);
var w = r.width;    // Это объект Number, а не элементарное числовое значение.
var new_w = w + 1;  // Ой! new_w теперь равно "51", а не 6, как ожидалось.
```

Чтобы обойти эту проблему, вы можете явно вызвать метод `valueOf()` для преобразования объекта Number в соответствующее числовое значение. Например:

```
var r = new java.awt.Rectangle(0,0,5,5);
var w = r.width.valueOf();  // Теперь получилось элементарное числовое значение.
var new_w = w + 1;          // Теперь new_w равно 6, как и требовалось.
```

22.6. Преобразование JavaObject в JavaScript

Разобравшись с непростым предыдущим разделом, вы можете ожидать, что с темой преобразования данных покончено. К сожалению, нам следует еще обсудить, как JavaScript преобразует объекты JavaObject в различные элементарные типы JavaScript. На рис. 22.3 можно заметить, что многие типы данных Java, включая строки Java (экземпляры *java.lang.String*), преобразуются в JavaScript в объекты JavaObject, а не в элементарные типы данных JavaScript, такие как строки. Это значит, что при использовании LiveConnect вам часто придется работать с объектами JavaObject.

Обратимся снова к табл. 11.1, в которой показано, как преобразуются различные типы данных JavaScript при использовании в различных контекстах. Например, когда число используется в строковом контексте, оно преобразуется в строку, а когда объект используется в логическом контексте, он преобразуется в значение `false`, если он равен `null`, и в `true` в противном случае. Эти правила преобразования не применяются к объектам JavaObject, которые преобразуются по своим собственным правилам, приведенным ниже:

- Когда JavaObject используется в числовом контексте, он преобразуется в число путем вызова метода `doubleValue()` представляемого им объекта Java. Если в объекте Java этот метод не определен, возникает ошибка JavaScript.
- В логическом контексте объект JavaObject преобразуется в логическое значение при помощи метода `booleanValue()` представляемого им объекта

Java. Если в объекте Java этот метод не определен, возникает ошибка JavaScript.

- В строковом контексте объект JavaObject преобразуется в строковое значение путем вызова метода `toString()` представляемого им объекта Java. Все объекты Java определяют или наследуют этот метод, поэтому такое преобразование всегда оказывается успешным.
- В объектном контексте для объекта JavaObject никакое преобразование не требуется, так как он уже является объектом JavaScript.

Из-за этих специальных правил преобразования, а также по другим причинам объекты JavaObject ведут себя не так, как другие объекты JavaScript, и есть несколько подводных камней, о которых вы должны знать. Во-первых, не принято работать с объектами JavaObject, представляющими экземпляры *java.lang.Double* или другие числовые объекты. Во многих отношениях такой объект JavaObject ведет себя так же, как элементарное числовое значение, но будьте аккуратны при использовании оператора +. Когда вы используете объект JavaObject (или любой объект JavaScript) с оператором +, вы задаете строковый контекст, поэтому объект преобразуется в строку для выполнения конкатенации строк, а не в число для выполнения сложения.

Когда необходимо явно преобразовать объект JavaScript в элементарное значение, обычно вызывается его метод `valueOf()`. Заметьте, что для объектов JavaObject это не годится. Мы уже говорили, что в классе JavaObject не определены собственные свойства, все его свойства представляют поля и методы представляемого им объекта Java. То есть объекты JavaObject не поддерживают обычные методы JavaScript, такие как `valueOf()`. В случае нашего объекта *java.lang.Double* в обертке JavaObject, когда вам требуется привести объект к элементарному значению, вы должны вызвать метод Java `doubleValue()`.

Еще одно различие между объектами JavaObject и другими типами данных JavaScript состоит в том, что JavaObject могут выступать в логическом контексте, только если в них определен метод `booleanValue()`. Предположим, что `button` – это переменная JavaScript, которая содержит либо `null`, либо объект JavaObject, представляющий экземпляр класса *java.awt.Button*. Чтобы проверить, содержит ли переменная `null`, можно по привычке написать:

```
if (!button) { ... }
```

Пока значение `button` равно `null`, все прекрасно. Но если `button` действительно содержит JavaObject, представляющий экземпляр *java.awt.Button*, LiveConnect попытается вызвать метод `booleanValue()`. Когда выяснится, что в *java.awt.Button* он не определен, возникает ошибка JavaScript. Решение в этом случае состоит в явном указании цели вашей проверки, чтобы избежать использования JavaObject в логическом контексте:

```
if (button != null) { ... }
```

Это в любом случае полезная привычка, так как в результате читать и понимать код становится легче.

22.7. Преобразование данных из Java в JavaScript

В последних двух разделах рассмотрены правила, по которым значения преобразуются при чтении и записи значений полей Java и вызове методов Java из JavaScript. Эти правила определяют, как объекты JavaObject, JavaArray и JavaClass преобразуют данные; они применяются только к ситуации, когда JavaScript манипулирует данными Java. Когда Java манипулирует данными JavaScript, преобразование выполняется классом Java JSObject, и правила преобразования другие. Эти преобразования показаны на рис. 22.4 и рис. 22.5.

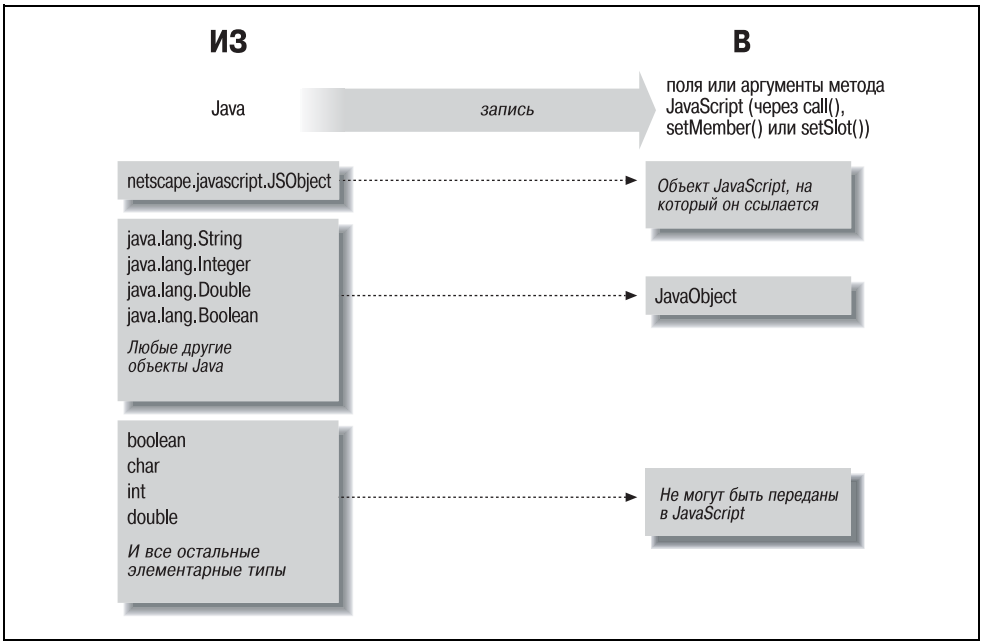

Рис. 22.4. *Преобразования данных, выполняемые при записи значений JavaScript из кода Java*

Изучая эти рисунки, следует помнить, что Java может взаимодействовать с JavaScript только через API, предоставляемый классом JSObject. Так как Java – строго типизированный язык, методы, определенные в этом классе, могут работать только с объектами Java, а не со значениями элементарных типов. Например, когда вы читаете значение числа JavaScript, метод `getMember()` возвращает объект `java.lang.Double`, а не элементарное значение `double`.

При написании функций JavaScript, которые вызываются из Java, имейте в виду, что аргументы, переданные из Java, – это либо объекты JavaScript, полученные из обертки Java – JSObject, либо JavaObject. LiveConnect просто не

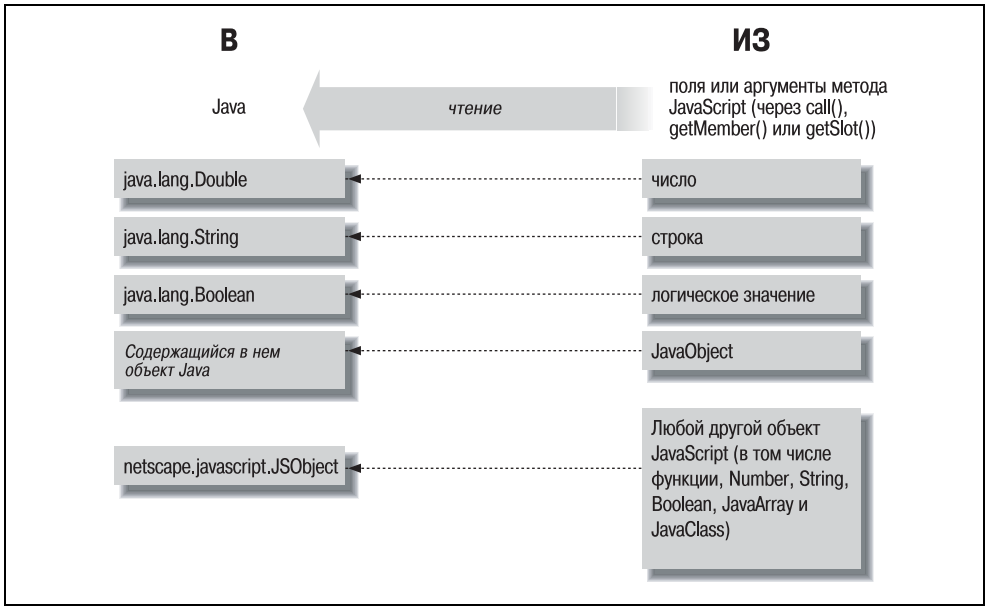

Рис. 22.5. *Преобразования данных, выполняемые при чтении значений JavaScript из кода Java*

разрешает Java передавать в качестве аргументов метода элементарные значения. Мы уже знаем из этой главы, что объекты JavaObject ведут себя не совсем так, как другие объекты. Например, экземпляр *java.lang.Double* ведет себя не так, как элементарное числовое значение JavaScript или даже объект JavaScript Number. При работе со свойствами JavaScript, значения которых установлены в Java, необходимо соблюдать определенные предосторожности.

Один из способов избежать всех трудностей с преобразованием данных состоит в том, чтобы использовать для взаимодействия кода Java с JavaScript метод `eval()` класса JSObject. Для этого код Java должен преобразовать все аргументы методов и значения свойств в строковый формат. Затем строка для исполнения может быть без изменений передана в JavaScript, где строковый формат данных преобразуется в нужные значения JavaScript.