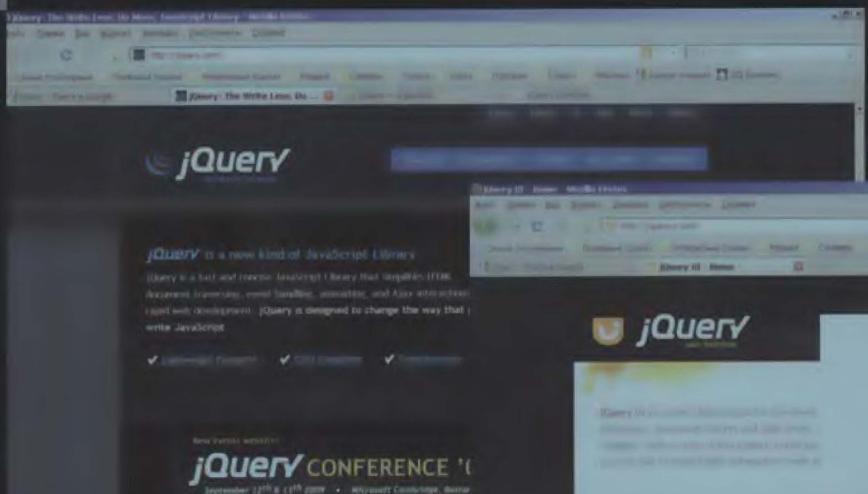


jQuery

Новый стиль программирования *на JavaScript*

Н.А. Прохоренок

- Библиотека jQuery:
 - селекторы, манипуляция элементами, обработка событий
 - эффекты и анимация, обработка данных формы
 - вспомогательные функции и свойства
- Технология AJAX и jQuery
- Библиотека jQuery UI



jQuery

Новый стиль программирования *на* JavaScript

Н.А. Прохоренок



Издательский дом "Вильямс"
Москва • Санкт-Петербург • Киев
2010

ББК 32.973.26-018.2.75

П84

УДК 681.3.07

Издательский дом “Вильямс”

Зав. редакцией *A.B. Слепцов*

По общим вопросам обращайтесь в Издательский дом “Вильямс” по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Прохоренок, Н.А.

П84 jQuery. Новый стиль программирования на JavaScript. :— М. : ООО “И.Д. Вильямс”, 2010. — 272 с. : ил.

ISBN 978-5-8459-1603-7 (рус.)

ББК32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения Издательского дома “Вильямс”.

Copyright © 2010 by WilliamsPublishing House.

All rights reserved including the right of reproduction in whole or in part in any form.

Научно-популярное издание

Николай Анатольевич Прохоренок

jQuery

Новый стиль программирования на JavaScript

Литературный редактор *Е.Д. Давидян*

Верстка *О.В. Романенко*

Художественный редактор *В.Г. Павлютин*

Корректор *Л.А. Гордиенко*

Подписано в печать 28.10.2009. Формат 70x100/16

Гарнитура Times. Печать офсетная

Усл. печ. л. 21,93. Уч.-изд. л. 11,76.

Тираж 1000 экз. Заказ № 19626.

Отпечатано по технологии СтР
в ОАО “Печатный двор” им. А. М. Горького
197110, Санкт-Петербург, Чкаловский пр., 15.

ООО “И. Д. Вильямс”, 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1603-7 (рус.)

© Издательский дом “Вильямс”, 2010

Оглавление

Введение	9
Глава 1. Основы jQuery	11
Глава 2. Селекторы	17
Глава 3. Перебор элементов	25
Глава 4. Изменение коллекции элементов	29
Глава 5. Манипуляции с элементами	37
Глава 6. Обработка событий	57
Глава 7. Эффекты и анимация	79
Глава 8. Обработка данных формы	91
Глава 9. Вспомогательные функции и свойства	103
Глава 10. Основы технологии AJAX	117
Глава 11. Поддержка AJAX в jQuery	135
Глава 12. Библиотека jQuery UI	161
Предметный указатель	260

Содержание

Введение	9
Глава 1. Основы jQuery	11
1.1. Подключение библиотеки jQuery	12
1.2. Определение готовности документа	12
1.3. Обработка конфликтных ситуаций	14
1.4. Форматы функции \$()	15
Глава 2. Селекторы	17
2.1. Часто используемые селекторы	17
2.2. Группирование селекторов	18
2.3. Привязка к элементам документа	18
2.4. Привязка к параметрам тегов	19
2.5. Псевдоклассы	20
Глава 3. Перебор элементов	25
3.1. Доступ к элементу по индексу	25
3.2. Метод each()	27
Глава 4. Изменение коллекции элементов	29
4.1. Добавление и фильтрация элементов	29
4.2. Привязка поиска к другим элементам	30
4.3. Метод map()	35
Глава 5. Манипуляции с элементами	37
5.1. Изменение содержимого элементов	37
5.2. Добавление содержимого перед элементом или после него	39
5.3. Вложение элементов	40
5.4. Перемещение и клонирование элементов	41
5.5. Очистка содержимого и удаление элемента	42
5.6. Замена элемента	43
5.7. Изменение атрибутов CSS	44
5.8. Управление классами стилей	48
5.9. Доступ к параметрам тегов	50
5.10. Вычисление положения элементов	54

Глава 6. Обработка событий	57
6.1. События документа	57
6.2. События мыши	59
6.3. События клавиатуры	63
6.4. События формы	64
6.5. Универсальные обработчики событий	66
6.6. Методы live() и die()	72
6.7. Всплытие событий	74
6.8. Действия по умолчанию и их отмена	76
Глава 7. Эффекты и анимация	79
7.1. Управление отображением элемента	79
7.2. Изменение прозрачности элемента	81
7.3. Создание анимации	83
7.4. Прерывание анимации	87
7.5. Управление очередью анимаций	89
Глава 8. Обработка данных формы	91
8.1. Текстовое поле и поле ввода пароля	91
8.2. Поле для ввода многострочного текста	93
8.3. Список с возможными значениями	94
8.4. Флажок и переключатели	97
8.5. Обработка щелчка на кнопке	99
8.6. Получение всех значений формы	100
Глава 9. Вспомогательные функции и свойства	103
9.1. Функция \$.each() — перебор элементов	103
9.2. Функция \$.grep() — поиск в массиве	104
9.3. Функция \$.map() — преобразование массива	105
9.4. Функция \$.inArray() — поиск элемента в массиве	106
9.5. Функция \$.merge() — объединение массивов	106
9.6. Функция \$.makeArray() — создание массива элементов	106
9.7. Функция \$.unique() — удаление повторяющихся элементов	107
9.8. Функция \$.trim() — удаление пробельных символов	109
9.9. Функции \$.data() и \$.removeData() — работа с данными	109
9.10. Свойство \$.browser — определение типа и версии браузера	111
9.11. Свойство \$.boxModel — определение блочной модели	112
9.12. Создание собственных модулей	114
Глава 10. Основы технологии AJAX	117
10.1. Обмен данными с помощью тега <iframe>	118
10.2. Объект XMLHttpRequest	119
10.3. Обмен данными в текстовом формате	122
10.4. Обмен данными в формате XML	127
10.5. Обмен данными в формате JSON	131

Глава 11. Поддержка AJAX в jQuery	135
11.1. Метод load()	135
11.2. Функция \$.getJSON()	139
11.3. Функция \$.getScript()	142
11.4. Функция \$.get()	143
11.5. Функция \$.post()	146
11.6. Функция \$.ajax()	152
11.7. Глобальные обработчики событий AJAX	156
Глава 12. Библиотека jQuery UI	161
12.1. Модуль UI Draggable — перемещение элементов	161
12.2. Модуль UI Droppable — “сбрасывание” элементов	172
12.3. Модуль UI Sortable — сортировка элементов	179
12.4. Модуль UI Selectable — выделение элементов	190
12.5. Модуль UI Resizable — изменение размеров	195
12.6. Модуль UI Accordion — компонент “Аккордеон”	201
12.7. Модуль UI Tabs — панель с вкладками	208
12.8. Модуль UI Dialog — диалоговые окна	215
12.9. Модуль UI Datepicker — календарь	224
12.10. Модуль UI Progressbar — индикатор хода процесса	239
12.11. Модуль UI Slider — шкала с бегунком	242
12.12. Модуль UI Effects — визуальные эффекты	248
12.12.1. Плавное изменение цвета	248
12.12.2. Управление классами стилей	249
12.12.3. Методы, позволяющие использовать эффекты	251
12.12.4. Эффекты	253
Несколько слов в заключение	258
Предметный указатель	260

Введение

Добро пожаловать в мир jQuery!

jQuery — это JavaScript-библиотека, обеспечивающая кроссбраузерную поддержку приложений (работает в Internet Explorer 6.0+, Mozilla Firefox 2+, Safari 3.0+, Opera 9.0+ и Chrome). Автор библиотеки Джон Резиг (John Resig) впервые представил свое творение в январе 2006 года на компьютерной конференции в Нью-Йорке, а в августе того же года была выпущена первая стабильная версия библиотеки. За прошедшие годы библиотека претерпела множество изменений и на текущий день содержит функционал, полезный для максимально широкого круга задач. Она имеет небольшой размер (например, размер минимизированной версии — 55,9 Кбайт, а сжатой версии — 19 Кбайт) и не засоряет глобальное пространство имен тривиальными идентификаторами.

Потрясающие возможности механизма селекторов, позволяющие легко получить доступ к любому элементу объектной модели документа, сделали библиотеку jQuery очень популярной. Судите сами. Чтобы получить ссылку на DOM-элемент с помощью JavaScript, обычно используется метод `getElementById()`. Например, изменим HTML-код элемента с идентификатором `div1`.

```
document.getElementById("div1").innerHTML = "Новый текст";
```

Код на jQuery, выполняющий то же самое действие, будет в два раза короче.

```
$("#div1").html("Новый текст");
```

Конечно, ради одной этой строки не имеет смысла подключать целую библиотеку. Но все дело в том, что функционал селекторов далеко не ограничивается одним идентификатором. Возможности селекторов можно сравнить разве что с регулярными выражениями языка Perl. В качестве примера изменим цвет текста во всех элементах A, в параметре `href` которых содержится ссылка на HTML-документ, причем элемент A должен быть расположен внутри элемента DIV, имеющего стилевой класс `cls1`.

```
 $("div.cls1 a[href$='.html']").css("color", "red");
```

Попробуйте выполнить аналогичную операцию с помощью JavaScript. Для этого понадобится далеко не одна строка кода.

Еще одной отличительной особенностью библиотеки jQuery является возможность составлять цепочки из вызовов методов, так как большинство методов jQuery возвращает объект, с которым можно производить дальнейшие манипуляции.

```
 $("#message") // Получили ссылку на элемент с id=message
   .html("Сообщение") // Изменили текст внутри элемента
   .parent() // Получили ссылку на родительский элемент
   .css("background-color", "#ffff4dd") // Задали цвет фона
   .width(300) // Ширина
   .height(200) // Высота
   // Плавно отобразили элемент за счет изменения прозрачности
   .fadeIn(3000);
```

Библиотека jQuery не оставила без внимания и технологию AJAX, позволяющую обмениваться данными с сервером без перезагрузки веб-страницы. В главе 10 мы изучим

базовые свойства и методы объекта XMLHttpRequest, а в главе 11 рассмотрим очень удобный интерфейс доступа к AJAX, предоставляемый библиотекой jQuery.

Большой популярности jQuery способствовали также дополнительные модули, реализующие готовые компоненты или добавляющие новую функциональность. В главе 12 мы рассмотрим библиотеку визуальных компонентов пользовательского интерфейса jQuery UI. Эта библиотека предоставляет готовые решения, которые может использовать практически любой разработчик, даже не владея основами jQuery и JavaScript. Без особого труда можно создать компонент “Аккордеон”, панель с вкладками, различные пользовательские диалоговые окна, вставить в веб-страницу календарь. Библиотека jQuery UI добавляет возможность перемещения и изменения размеров любых элементов с помощью мыши, позволяет сортировать и выделять элементы, а также предоставляет множество визуальных эффектов, которые сделают ваш сайт более привлекательным.

Благодаря своей универсальности библиотека jQuery будет полезна практически любому разработчику. Она подходит новичку, так как позволяет забыть о проблеме с кросбраузерностью приложения (вышел новый веб-браузер — сменил версию библиотеки и все опять работает). Библиотека идеальна для профессионалов, так как позволяет сократить код минимум в три раза. А это в свою очередь позволит написать очень сложный код с минимальными усилиями и потерей времени.

Желаю приятного прочтения и надеюсь, что эта книга станет верным спутником в вашей повседневной деятельности и в лучшую сторону изменит ваш стиль программирования на JavaScript.

Ваши замечания, пожелания, а также сообщения о замеченных опечатках можете оставить на форуме сайта <http://wwwadmin.ru/>. Скачать исходные коды листингов из книги можно со страницы <http://wwwadmin.ru/javascript/jquery/>.

От издательского дома “Вильямс”

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш веб-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: info@williamspublishing.com
WWW: <http://www.williamspublishing.com>

Наши почтовые адреса:

в России: 127055, г. Москва, ул. Лесная, д. 43, стр. 1
в Украине: 03150, Киев, а/я 152

Глава 1

Основы jQuery

В этой книге предполагается, что все обсуждаемые файлы сохраняются в кодировке UTF-8. Поэтому, прежде чем изучать возможности библиотеки jQuery, рассмотрим, чем отличается UTF-8 от других кодировок и как правильно сохранить файл в этой кодировке. Итак, приступим.

Все символы, которые мы видим на экране монитора, внутри компьютера хранятся в виде чисел. Каждому символу соответствует определенное число (код символа). Для ответа на вопрос, как должен выглядеть символ, представленный определенным кодом, предназначены таблицы соответствий, которые называются *кодировками*. Кодировки могут быть одно- и многобайтовыми.

В однобайтовых кодировках символ кодируется одним байтом. Первые 7 бит позволяют закодировать 128 символов, соответствующих кодировке ASCII. В число этих символов входят цифры, буквы латинского алфавита, знаки препинания и некоторые служебные символы (например, перенос строки, табуляция и т.д.). Коды этих символов одинаковы практически во всех однобайтовых кодировках. Восьмой бит предназначен для кодирования символов национальных алфавитов. Таким образом, однобайтовые кодировки позволяют закодировать всего 256 символов. Для кодирования букв русского языка разработано пять кодировок — windows-1251 (cp1251), cp866, iso8859-5, koi8-r и mac-cyrillic. Сложность заключается в том, что код одной и той же русской буквы в этих кодировках может быть разным. Из-за этого возникает множество проблем.

В кодировке UTF-8 один символ может кодироваться несколькими байтами. Первые 128 символов соответствуют кодировке ASCII и кодируются всего одним байтом. Остальные символы кодируются переменным количеством байтов — от двух до шести (на практике — до четырех). Буквы русского алфавита и некоторых других европейских языков кодируются двумя байтами. Иными словами кодировка UTF-8 позволяет закодировать символы всех существующих алфавитов и способна заменить все кодировки сразу. Сайт может быть на русском или на любом другом языке, а кодировка будет одна и та же. Этой кодировкой мы и будем пользоваться.

При *'сохранении* файлов в кодировке UTF-8 следует учитывать, что использовать приложение Блокнот для этого нельзя, так как при сохранении в начало файла будут вставлены служебные символы, называемые сокращенно BOM (Byte Order Mark, метка порядка байтов). Для кодировки UTF-8 эти символы являются необязательными и не позволят нам в дальнейшем, например, установить заголовки ответа сервера. Для работы с кодировкой UTF-8 необходимо установить на компьютер программу Notepad++. Скачать программу можно абсолютно бесплатно с веб-страницы <http://notepad-plus.sourceforge.net/ru/site.htm>. Из двух вариантов (zip-архив и инсталлятор) советую выбрать именно инсталлятор, так как в этом случае при установке можно будет выбрать язык интерфейса программы. Процедура установки Notepad++ предельно проста и в комментариях не нуждается. При создании нового документа в меню Кодировки следует установить флажок Кодировать в UTF-8 (без BOM).

Примечание

Тот факт, что мы будем использовать кодировку UTF-8, отнюдь не означает, что библиотека jQuery может работать только с этой кодировкой. В своих проектах вы можете использовать любую другую кодировку. Однако следует учитывать, что запросы AJAX по умолчанию выполняются в кодировке UTF-8. При использовании других кодировок придется выполнять перекодирование.

1.1. Подключение библиотеки jQuery

Прежде чем использовать библиотеку jQuery, ее необходимо вначале скачать с сайта <http://jquery.com/>, разместить на своем сервере, а затем подключить к HTML-документу. Подключение производится с помощью тега `<script>`, в параметре `src` которого указывается абсолютный или относительный путь к библиотеке.

```
<script type="text/javascript" src="Путь к библиотеке jQuery">  
</script>
```

Сам тег `<script>` должен быть размещен в разделе HEAD HTML-документа.

Библиотеку jQuery можно загрузить и с сайта <http://ajax.googleapis.com/>. В этом случае подключение будет выглядеть так.

```
<script type="text/javascript"  
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js">  
</script>
```

Если посетитель ранее заходил на другой сайт, на котором библиотека jQuery также подгружалась с сайта <http://ajax.googleapis.com/>, то веб-браузер не будет повторно загружать библиотеку и использует данные, сохраненные в кеше. Таким образом, скорость работы вашего сайта может увеличиться. В этом и заключается преимущество данного метода. Однако если сайт <http://ajax.googleapis.com/> будет недоступен, то возможны проблемы.

Примечание

Подробную информацию об этом способе можно получить на странице <http://code.google.com/intl/ru-RU/apis/ajaxlibs/documentation/index.html#jquery>.

1.2. Определение готовности документа

Обычно большинство действий производится над элементами HTML-документа. По этой причине выполнять скрипты необходимо только после полной загрузки документа. Без библиотеки jQuery для этой цели обычно используется событие `onload` объекта `window`.

```
window.onload = function() {  
    alert("Документ полностью загружен");  
}
```

Однако это событие возникает только после *полной* загрузки самого HTML-документа, а также всех других элементов, например изображений. Загрузка изображений (или баннеров с другого домена) может занимать определенное время, в течение которого выполнение скриптов как бы подвисает. Библиотека jQuery избавляет нас от потери времени и предоставляет возможность выполнения скриптов сразу после формирования

структурой документа, не дожидаясь загрузки других элементов. Обработать это событие можно с помощью метода `ready()`.

```
jQuery(document).ready(function() {
    alert("Документ доступен для выполнения скриптов");
});
```

Функция `jQuery()` имеет псевдоним `$()`. Используя этот псевдоним, можно обработать событие следующим образом.

```
$(document).ready(function() {
    alert("Документ доступен для выполнения скриптов");
});
```

Если в качестве параметра функции `jQuery()` указать ссылку на функцию, то она также будет выполнена сразу после формирования структуры документа.

```
jQuery(function() {
    alert("Документ доступен для выполнения скриптов");
});
```

Этот код можно еще сократить.

```
$(function() {
    alert("Документ доступен для выполнения скриптов");
});
```

Необходимо также заметить, что в программе может быть несколько вызовов методов `ready()`. В этом случае выполнение методов происходит в порядке их объявления внутри программы.

Рассмотрим последовательность событий на примере. Для этого создадим HTML-документ и объявим несколько методов `ready()`, а также обработаем событие `.onload` (листинг 1.1).

Листинг 1.1. Последовательность обработки событий

```
<html>
<head>
<title>Последовательность обработки событий</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
window.onload = function() {
    alert("Событие onload");
}
$(document).ready(function() {
    alert("Метод ready(). Вызов 1");
});
$(document).ready(function() {
    alert("Метод ready(). Вызов 2");
});
$(function() {
    alert("Вызов функции $()");
});
</script>
</head>
<body>
```

```
Пример последовательности обработки событий
</body>
</html>
```

При выполнении этого примера будет иметь место следующая последовательность событий.

```
Метод ready(). Вызов 1
Метод ready(). Вызов 2
Вызов функции $()
Событие onload
```

Как видно из примера, событие `onload` возникает самым последним, а методы `ready()` и функция `$()` выполняются в порядке их объявления в программе.

1.3. Обработка конфликтных ситуаций

Как вы уже знаете, функция `jQuery()` имеет псевдоним `$()`. В некоторых других библиотеках (например, в `Prototype`) также объявлена функция `$()`. Если использовать такие библиотеки одновременно, то возникнет конфликт имен. Библиотека `jQuery` позволяет избежать этого конфликта. Для этого необходимо вначале подключить конфликтную библиотеку, а затем библиотеку `jQuery`. Сразу после подключения следует вызвать функцию `noConflict()`.

```
jQuery.noConflict();
```

В этом случае библиотека `jQuery` освободит функцию `$()` для использования другой библиотекой.

Если результат выполнения функции `noConflict()` присвоить какой-либо переменной, то ее имя можно использовать вместо функции `$()` (листинг 1.2).

Листинг 1.2. Обработка ситуации конфликта имен

```
<html>
<head>
<title>Обработка конфликтных ситуаций</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script type="text/javascript">
function $() {
    alert("Функция $() из другой библиотеки");
}
</script>
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
var jq = jQuery.noConflict();
</script>
</head>
<body>
<div id="div1"></div>
<script type= "text/javascript">
$(); // Вызов функции из другой библиотеки
jq("#div1").html("Пример использования библиотеки jQuery");
</script>
</body>
</html>
```

В этом примере мы присвоили результат выполнения функции `noConflict()` переменной `jq`. Теперь можно использовать это имя в качестве названия функции.

```
jq("#div1").html("Пример использования библиотеки jQuery");
```

Если результат нигде не сохранять, то обращение выглядит следующим образом.

```
jQuery("#div1").html("Пример использования библиотеки jQuery");
```

Использовать функцию `$()` в данном случае уже нельзя, так как будет вызвана функция из другой библиотеки. Если же необходимо использовать функцию `$()` применительно к библиотеке `jQuery`, то можно поступить следующим образом.

```
(function($) {
    $("#div1").html("Пример использования библиотеки jQuery");
}) (jQuery);
```

Этим способом обычно пользуются при создании расширений (плагинов). Можно также воспользоваться следующим кодом.

```
jQuery(function($) {
    $("#div1").html("Пример использования библиотеки jQuery");
});
```

1.4. Форматы функции `$()`

Основную функциональность библиотеки `jQuery` выполняет функция `jQuery()`, которая имеет псевдоним `$()`. Это единственные идентификаторы, которые доступны в глобальной области видимости. Все остальные функции находятся в области имен библиотеки `jQuery`. Функция `$()` поддерживает несколько форматов:

- `$(<Функция>);`
- `$(<Элемент объектной модели документа>);`
- `$(<HTML-текст>);`
- `$(<Селектор>[, <Контекст>]).`

Если в качестве параметра указать функцию, то она будет выполнена сразу после формирования структуры документа.

```
$(function() {
    alert('Документ доступен для выполнения скриптов');
});
```

Второй формат функции позволяет указать элемент объектной модели документа. В качестве примера зададим цвет фона для всего документа после загрузки:

```
$(document).ready(function() {
    $(document.body).css("background-color", "silver");
});
```

Третий формат функции позволяет создавать новые элементы, которые затем можно будет вставить в определенное место в HTML-документе. Рассмотрим это на примере (листинг 1.3).

Листинг 1.3. HTML-текст в качестве параметра функции \$()

```
<html>
<head>
<title>HTML-текст в качестве параметра функции $()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
$(document).ready(function() {
    $("<div>Вставленный текст</div>").appendTo("#div1");
});
</script>
</head>
<body>
<div id="div1">Строка 1</div>
<div id="div2">Строка 2</div>
</body>
</html>
```

Обратите внимание на следующую строку.

```
$("<div>Вставленный текст</div>").appendTo("#div1");
```

Здесь мы в функции \$() формируем HTML-текст, а затем с помощью метода appendTo() вставляем его после элемента, имеющего идентификатор div1 (id="div1"). При выполнении получим в окне веб-браузера следующий результат.

Строка 1
Вставленный текст
Строка 2

Четвертый формат функции \$() является наиболее часто используемым. Он позволяет получить ссылку на элемент или коллекцию элементов по указанному селектору. В качестве этого параметра указываются селекторы, которые являются частью стандарта CSS, а также некоторые специальные селекторы. Вторым параметром функции \$() может быть указан контекст. Например, ссылка на XML-документ, полученный с сервера в результате AJAX-запроса. Если параметр не указан, то контекстом является текущий документ.

Глава 2

Селекторы

Как вы уже знаете, в качестве параметра функции `$()` могут быть указаны селекторы, которые являются частью стандарта CSS. Рассмотрим различные возможные селекторы подробно.

2.1. Часто используемые селекторы

Перечислим наиболее часто используемые селекторы.

- `*` — коллекция всех тегов

```
var n = $("*").size(); // Количество тегов
```

- Тег — коллекция всех тегов, имеющих указанное имя

```
$("p").css("backgroundColor", "red");  
// Делаем фон всех абзацев красным
```

- `#Идентификатор` — ссылка на элемент с указанным идентификатором

```
$("#id1").css("backgroundColor", "red");  
// Делаем фон красным для элемента с id="id1"
```

(Если элементов с одинаковым идентификатором несколько, то будет возвращена ссылка только на первый элемент)

- Тег`#Идентификатор` — ссылка на элемент с указанным идентификатором, который расположен в определенном теге

```
 $("p#id1").css("backgroundColor", "red");  
// Делаем фон красным для абзаца с id="id1"
```

(Если абзацев с одинаковым идентификатором несколько, то будет возвращена ссылка только на первый абзац. Если идентификатор находится в другом теге, то он будет проигнорирован)

- `.Класс` — коллекция элементов, имеющих указанный класс

```
 $(".cls2").css("backgroundColor", "red");  
// Делаем фон красным для всех элементов с class="cls2"
```

- Тег.`Класс` — коллекция элементов, имеющих указанный класс в определенном теге

```
 $("p.cls2").css("backgroundColor", "red");  
// Делаем фон красным для всех абзацев с class="cls2"
```

Примечание

Для ускорения выборки не следует указывать название тега перед идентификатором. Например, пишите `#id1` вместо `p#id1`. Если поиск производится по стилевому классу, то, наоборот, следует указать название тега. Например, пишите `p.cls1` вместо `.cls1`.

Если название идентификатора (или класса) содержит специальные символы (например, точку или квадратные скобки), то их необходимо экранировать двумя слешами (\ \).

```
$("#div1\\.index\\[5\\\"]").html("Текст");
```

В этом примере получаем ссылку на следующий элемент.

```
<div id="div1.index[5]"></div>
```

2.2. Группирование селекторов

Если необходимо, например, применить один стиль к разным элементам, то можно указать селекторы через запятую.

```
$("#id1, div").addClass("newClass");
```

В этом примере для элемента с идентификатором id1, а также для всех тегов <div> назначается стилевой класс newClass.

2.3. Привязка к элементам документа

Выполнить поиск элемента с учетом заданного отношения к другому элементу HTML-документа можно следующими способами.

- Элемент1 Элемент2 — находим Элемент2, который располагается внутри контейнера Элемент1

```
 $("div a").css("color", "red");
```

Цвет текста ссылки станет красным, если тег <a> находится внутри тега <div>.

```
<div><a href="link.html">Ссылка</a></div>
```

- Элемент1 > Элемент2 — находим Элемент2, который является дочерним для Элемент1

```
 $("div > a").css("color", "red");
```

Цвет текста ссылки станет красным, если тег <a> находится внутри тега <div> и не вложен в другой тег.

```
<div>
  <a href="link1.html">Ссылка 1</a><br>
  <span><a href="link2.html">Ссылка 2</a></span>
</div>
```

В этом примере только первая ссылка станет красного цвета, так как вторая ссылка расположена внутри тега .

- Элемент1 + Элемент2 — находим Элемент2, который является соседним для Элемент1 и следует сразу после него.

```
 $("div + a").css("color", "red");
```

Цвет текста ссылки станет красным, если тег <a> следует сразу после тега <div>.

```
<div>Текст</div><a href="link.html">Ссылка</a>
```

- Элемент1 ~ Элемент2 — находим Элемент2, который следует после Элемент1, причем необязательно сразу.

```
 $("div ~ a").css("color", "red");
```

Цвет текста ссылки станет красным, если тег `<a>` следует после тега `<div>`.

```
<div>Текст</div>
<span>Текст</span><br>
<a href="link1.html">Ссылка 1</a><br>
<a href="link2.html">Ссылка 2</a><br>
<span><a href="link3.html">Ссылка 3</a></span><br>
<a href="link4.html">Ссылка 4</a><br>
```

В этом примере ссылки 1, 2 и 4 станут красного цвета. Ссылка 3 не станет красного цвета, так как расположена внутри тега ``.

При необходимости можно составлять выражения из нескольких селекторов

```
$( "div span a" ).css("color", "red");
```

Цвет текста ссылки станет красным, если тег `<a>` расположен внутри тега ``, а тот в свою очередь вложен в тег `<div>`

```
<div>
<a href="link1.html">Ссылка 1</a><br>
<span>
<a href="link2.html">Ссылка 2</a><br>
</span>
</div>
```

В этом примере только ссылка 2 будет красного цвета.

2.4. Привязка к параметрам тегов

Для привязки к параметрам тегов применяются следующие селекторы.

- [Параметр] — элементы с указанным параметром

```
$( "a[id]" ).css("color", "red");
```

Цвет текста ссылки станет красным, если тег `<a>` имеет параметр `id`.

```
<a id="link1" href="link1.html">Ссылка 1</a>
```

- [Параметр='Значение'] — коллекция элементов, у которых параметр точно равен значению

```
$( "a[href='link1.html']" ).css("color", "red");
```

Цвет текста ссылки станет красным, если параметр `href` тега `<a>` имеет значение `"link1.html"`.

- [Параметр`!=`'Значение'] — коллекция элементов, у которых параметр не равен значению

```
$( "a[href!='link1.html']" ).css("color", "red");
```

Цвет текста ссылки станет красным, если параметр `href` тега `<a>` не имеет значение `"link1.html"`.

- [Параметр`^=`'Значение'] — коллекция элементов, у которых параметр начинается с указанного значения

```
$( "a[href^='li']" ).css("color", "red");
```

Цвет текста ссылки станет красным, если значение параметра href тега <a> начинается с "li".

- [Параметр\$='Значение'] — коллекция элементов, у которых параметр оканчивается указанным значением

```
$("a[href$='.html']").css("color", "red");
```

Цвет текста ссылки станет красным, если значение параметра href тега <a> оканчивается на расширение ".html".

- [Параметр*='Значение'] — коллекция элементов, у которых параметр содержит указанный фрагмент значения

```
$("a[href*='link']").css("color", "red");
```

Цвет текста ссылки станет красным, если значение параметра href тега <a> содержит фрагмент "link".

Если необходимо сделать привязку сразу к нескольким параметрам, то используется следующий формат.

```
[Параметр='Значение'] [Параметр='Значение']
```

Пример:

```
$("a[href='link1.html'][id*='link']").css("color", "red");
```

Цвет текста ссылки станет красным, если значение параметра href тега <a> равно "link1.html", а параметр id содержит фрагмент "link".

2.5. Псевдоклассы

В библиотеке jQuery используются следующие псевдоклассы, определенные в CSS.

- :nth-child(N) — элемент, являющийся N-м дочерним элементом своего родительского элемента. Нумерация элементов начинается с 1. В качестве параметра можно указать индекс или два специальных слова:

◊ even — все четные элементы;

◊ odd — все нечетные элементы.

Выделим все нечетные пункты списка шрифтом красного цвета.

```
$("ul li:nth-child(odd)").css("color", "red");
```

В качестве параметра можно также указывать следующее значение.

<Сколько элементов>n+<Начальная позиция>

Выделим каждый третий пункт списка, начиная со второго.

```
$("ul li:nth-child(3n+2)").css("color", "red");
```

В итоге будут выделены пункты 2-й, 5-й, 8-й и т.д.

- :first-child — элемент, являющийся первым дочерним элементом своего родительского элемента. Выделим первый пункт списка шрифтом красного цвета.

```
$("ul li:first-child").css("color", "red");
```

- `:last-child` — элемент, являющийся последним дочерним элементом своего родительского элемента. Выделим последний пункт списка шрифтом красного цвета.

```
 $("ul li:last-child").css("color", "red");
```

- `:only-child` — элемент, являющийся единственным дочерним элементом своего родительского элемента.

```
 $("ul li:only-child").css("color", "red");
```

Пункт списка будет выведен шрифтом красного цвета, если других пунктов нет.

- `:empty` — пустой элемент.

```
 $("div:empty").text("Пустой DIV");
```

Во всех пустых тегах `<div>` выводим текст "Пустой DIV".

```
<div>Текст</div>
<div>
</div>
<div></div>
```

В этом примере текст будет выведен только в последнем теге `<div>`.

- `:not (Селектор)` — элемент, не соответствующий селектору.

```
 $("a:not (#link1)").css("color", "red");
```

Все ссылки в документе станут красного цвета, за исключением элемента с идентификатором `link1`.

- `:has (Селектор)` — все элементы, содержащие хотя бы один элемент, соответствующий селектору.

```
 $("div:has(a)").css("background-color", "red");
```

Выделяем все теги `<div>`, в которых содержится хотя бы одна ссылка.

Для привязки к элементам формы предназначены следующие псевдоклассы.

- `:input` — все элементы формы (теги `<input>`, `<select>`, `<textarea>`, `<button>`).
- `:text` — все текстовые поля (`type="text"`).
- `:password` — все поля для ввода паролей (`type="password"`).
- `:radio` — все переключатели (`type="radio"`).
- `:checkbox` — все поля для установки флагов (`type="checkbox"`).
- `:submit` — все кнопки для отправки формы (`type="submit"`).
- `:reset` — все кнопки очистки формы (`type="reset"`).
- `:button` — все обычные кнопки (`type="button"`, а также теги `<button>`).
- `:file` — все поля для отправки файлов (`type="file"`).
- `:enabled` — активный элемент.
- `:disabled` — неактивный элемент. В качестве примера выведем в активном текстовом поле текст **Активный элемент**, а в неактивном — текст **Неактивный элемент**.

```

$(document).ready(function() {
    $("#txt1")[0].disabled = true;
    $(":text:enabled").val("Активный элемент");
    $(":text:disabled").val("Неактивный элемент");
});

<input id="txt1" type="text"><br>
<input id="txt2" type="text">

```

После формирования структуры документа делаем неактивным элемент с идентификатором txt1. Затем, в зависимости от активности элемента, выводим соответствующее сообщение с помощью метода val().

- :checked — отмеченный элемент (например, установленный флагок). Выведем количество установленных флагков.

```
alert($(":checkbox:checked").length);
```

- :selected — все выбранные элементы списка (тег <select>). Выведем текст пункта списка после его выбора.

```

$(document).ready(function() {
    $("#sel1").change(function() {
        alert($("#sel1 option:selected").text());
    });
});

<select id="sel1">
<option>Пункт 1</option>
<option>Пункт 2</option>
<option>Пункт 3</option>
<option>Пункт 4</option>
</select>

```

Рассмотрим другие псевдоклассы.

- :even — все четные элементы (элементы нумеруются с 0).
- :odd — все нечетные элементы (элементы нумеруются с 0). В качестве примера назначим разные стилевые классы четным и нечетным строкам таблицы.

```

$("table tr:even").addClass("clsEven");
$("table tr:odd").addClass("clsOdd");

```

- :eq(Индекс) и :nth(Индекс) — элемент с указанным индексом (элементы нумеруются с 0). Выделим третий пункт списка.

```
 $("ul li:eq(2)").css("color", "red");
```

Выделим первый пункт списка.

```
 $("ul li:nth(0)").css("color", "red");
```

- :gt(Индекс) — элементы с индексом, большим, чем указан (элементы нумеруются с 0). Выделим все пункты списка, кроме первого.

```
 $("ul li:gt(0)").css("color", "red");
```

- :lt(Индекс) — элементы с индексом, меньшим, чем указан (элементы нумеруются с 0). Выделим первые два пункта списка.

- ```
$("ul li:lt(2)").css("color", "red");
```
- `:first` — первый элемент.
  - `:last` — последний элемент.
  - `:parent` — все элементы, у которых есть дочерние элементы (включая текст).
  - `:contains('Текст')` — все элементы, которые содержат указанный текст в любом месте. Выделим пункты списка, которые содержат фрагмент "Пункт 1".  

```
$("ul li:contains('Пункт 1')").css("color", "red");
```
- В этом случае будут выделены пункты, содержащие также текст "Пункт 11", "Пункт 12" и т.д.
- `:visible` — все видимые элементы (элементы с атрибутом `display` равным `block` или `inline`, а также элементы с атрибутом `visibility` равным `visible`).
  - `:hidden` — все невидимые элементы (элементы с атрибутом `display` равным `none`, элементы с атрибутом `visibility` равным `hidden`, а также скрытые элементы форм (`type="hidden"`)).
  - `:header` — все заголовки (теги `<h1>`...`<h6>`). Выделим все заголовки шрифтом белого цвета на черном фоне.  

```
$(":header").css({ backgroundColor:"black", color:"white" });
```
  - `:animated` — все элементы, с которыми выполняется анимация. Прервем все анимации.  

```
$(":animated").stop(true, true);
```



## Глава 3

# Перебор элементов

В предыдущих главах мы научились получать коллекцию элементов с помощью функции `$()` и механизма селекторов. Большинство методов библиотеки jQuery позволяет изменить какое-либо свойство сразу всех элементов коллекции. Например, выведем во всех тегах `<p>` текст “Привет, мир!”

```
$("p").text("Привет, мир!");
```

Это, конечно, удобно. Но иногда необходимо получить доступ к какому-либо отдельному элементу коллекции. Прежде чем обратиться к конкретному элементу, следует с помощью свойства `length` узнать количество элементов в коллекции.

```
alert($("p").length);
```

Кроме того, можно воспользоваться методом `size()`.

```
alert($("p").size());
```

Теперь, когда мы знаем количество элементов, можно приступить к перебору всех элементов коллекции.

## 3.1. Доступ к элементу по индексу

Получить доступ к элементу можно, указав его индекс в квадратных скобках. Нумерация начинается с нуля. В качестве примера выведем текст “Привет, мир!” только во втором теге `<p>`.

```
var elems = $("p");
if (elems.size() >= 2) {
 elems[1].innerHTML = "Привет, мир!";
}
else {
 alert("Элемента для вывода нет!");
}
```

Обратите внимание на тот факт, что доступ по индексу возвращает ссылку на DOM-элемент. Поэтому изменить содержимое можно с помощью свойства `innerHTML` объекта `document`. Применить метод jQuery `text()` можно только после нахождения элемента.

```
$(elems[1]).text("Привет, мир!");
```

Вместо квадратных скобок можно использовать метод `get()`. Метод имеет следующий формат.

```
get([<Индекс элемента>])
```

Если индекс элемента не указан, то возвращается массив DOM-элементов.

```
var elems = $("p").get();
alert("Количество DOM-элементов " + elems.length);
```

Указание параметра позволяет получить доступ к конкретному элементу. Нумерация элементов начинается с 0. В качестве примера во всех тегах `<p>` выведем индекс текущего абзаца.

```
var elems = $("p");
for (var i=0, count=elems.length; i<count; i++) {
 elems.get(i).innerHTML = i;
}
```

Все рассмотренные способы позволяли получить доступ к конкретному DOM-элементу. Если необходимо получить доступ к элементу коллекции jQuery, то следует использовать метод `eq()`. Нумерация элементов начинается с 0. Переделаем наш предыдущий пример и используем метод jQuery `html()` вместо свойства `innerHTML` объекта `document`.

```
var elems = $("p");
for (var i=0, count=elems.length; i<count; i++) {
 elems.eq(i).html(i);
}
```

Метод `slice()` позволяет получить срез коллекции. Возвращает новую коллекцию элементов. Имеет следующий формат.

```
slice(<Начальная позиция>[, <Конечная позиция>])
```

Следует учитывать, что нумерация позиций начинается с 0, а конечная позиция не попадает в диапазон возвращаемых значений. Выделим первую ссылку в коллекции.

```
$("a").slice(0,1).css("color", "red");
```

А теперь выделим только вторую и третью ссылки.

```
$("a").slice(1,3).css("color", "red");
```

Если второй параметр не указан, то будут возвращены элементы от начальной позиции до конца коллекции. Выделим все ссылки, начиная с третьей.

```
$("a").slice(2).css("color", "red");
```

В случае указания отрицательного значения позиции будут отсчитываться с конца коллекции. Выделим две последние ссылки.

```
$("a").slice(-2).css("color", "red");
```

Пронумеруем все ссылки в документе с помощью метода `slice()`.

```
var elem = $("a");
for (var i=0, count=elem.size(); i<count; i++) {
 elem.slice(i,i+1).html("Ссылка " + i);
}
```

С помощью метода `index()` можно получить индекс элемента на странице. Если элемент не найден, то возвращается значение -1. Нумерация элементов начинается с нуля. Метод имеет следующий формат.

```
index(<DOM-элемент>)
```

При щелчке на абзаце выведем его индекс.

```
$(p).click(function() {
 alert("Индекс: " + $("p").index(this));
});
```

```
<p>Абзац 1</p>
<p>Абзац 2</p>
<p>Абзац 3</p>
<p>Абзац 4</p>
```

## 3.2. Метод each()

Метод `each()` позволяет перебрать все элементы коллекции без использования циклов. Имеет следующий синтаксис.

```
each(<Функция обратного вызова>)
```

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции> ([<Индекс> [, <Элемент DOM>]]) {
 // ...
}
```

Все параметры являются необязательными. Если в параметре `<Индекс>` указать переменную, то на каждой итерации в нее будет передаваться текущий индекс элемента в коллекции. Нумерация начинается с 0. Выведем индекс элемента во всех тегах `<div>`.

```
$("div").each(function(i) {
 this.innerHTML = "DIV с индексом " + i;
});
```

Текущий элемент доступен внутри функции через указатель `this`. Обратите внимание, указатель `this` ссылается на текущий элемент объектной модели документа, а не на элемент коллекции jQuery. По этой причине применять методы jQuery для этого элемента нельзя. В последнем примере для вывода текста внутри тегов `<div>` мы использовали свойство `innerHTML` объекта `document`, а не метод из библиотеки jQuery. Если необходимо использовать методы из библиотеки jQuery, то предварительно следует найти элемент с помощью функции `$()`. В качестве примера заменим свойство `innerHTML` на jQuery-метод `html()`.

```
$("div").each(function(i) {
 $(this).html("DIV с индексом " + i);
});
```

Если в параметре `<Элемент DOM>` указать переменную, то на каждой итерации в нее будет передаваться ссылка на текущий элемент DOM. Эту переменную можно использовать вместо указателя `this`.

```
$("div").each(function(i, d) {
 $(d).html("DIV с индексом " + i);
});
```

Если внутри функции вернуть значение `false`, то выполнение метода `each()` будет остановлено. При индексе, равном двум, прервем выполнение метода.

```
$("div").each(function(i) {
 $(this).html("DIV с индексом " + i);
 if (i == 2) {
 return false;
 }
});
```

Во всех этих примерах мы использовали анонимную функцию. Если необходимо вызвать обычную функцию, то ее имя указывается в качестве параметра метода each() без скобок.

```
function myFunc(i, d) {
 $(d).html("DIV с индексом " + i);
 if (i == 2) {
 return false;
 }
}
$(document).ready(function() {
 $("div").each(myFunc);
});
```

В качестве примера использования метода each() получим URL-адреса всех ссылок в документе (листинг 3.1).

### Листинг 3.1. Получение URL-адресов всех ссылок

```
<html>
<head>
<title>Получение URL-адресов всех ссылок</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
$(document).ready(function() {
 $("a").each(function() {
 $("#div1").append($(this).attr("href") + "
");
 });
});
</script>
</head>
<body>
Ссылка 1

Ссылка 2

Ссылка 3

Все URL-адреса:

<div id="div1"></div>
</body>
</html>
```

После загрузки структуры документа находим коллекцию всех тегов `<a>` (`$("a")`). Далее с помощью метода `each()` перебираем все элементы. В качестве параметра указываем анонимную функцию. На каждой итерации внутри анонимной функции доступен указатель `(this)` на текущий элемент коллекции. Находим текущий элемент (`$(this)`) и с помощью метода `attr()` получаем значение параметра `href` (`attr("href")`). Для вывода результата находим элемент с идентификатором `div1` (`$("#div1")`) и с помощью метода `append()` добавляем результат в конец к имеющемуся значению элемента.

## Глава 4

# Изменение коллекции элементов

Библиотека jQuery позволяет не только получать коллекцию элементов, а затем манипулировать ими, но и изменять эту первоначальную коллекцию в любой момент времени. Множество методов предоставляют возможность добавления новых элементов, фильтрации элементов, а также передвижения по дереву DOM. Рассмотрим эти методы подробно.

## 4.1. Добавление и фильтрация элементов

Для добавления и фильтрации элементов коллекции предназначены следующие методы.

- `add(<Выражение>)` — добавляет элементы в коллекцию. В качестве параметра можно указать HTML-элемент, селектор, коллекцию DOM-элементов или объект jQuery. Возвращает новую коллекцию элементов. Создадим элемент, затем добавим к нему новый HTML-элемент и выведем полученную коллекцию в конец содержимого всех тегов `<div>`.

```
$("Текст").add("Новый элемент")
.appendTo("div");
```

Найдем коллекцию всех тегов `<span>`, добавим к ним элемент с идентификатором `div1`, а затем выведем текст во всех элементах коллекции.

```
$("span").add("#div1").html("Новый текст");
```

Передадим коллекцию DOM-элементов и объект jQuery.

```
$("span").add($(".div").get()).html("Новый текст"); // DOM
$("span").add($(".div")); // jQuery
```

- `not(<Параметр>)` — позволяет удалить определенные элементы из коллекции. В качестве параметра может быть указан селектор, DOM-элемент, массив DOM-элементов или объект jQuery. Получим коллекцию всех тегов `<div>`, удалим из коллекции элемент с идентификатором `div1`, а затем выделим оставшиеся элементы коллекции.

```
$("div").not("#div1").css("background-color", "red");
```

Удалим из коллекции первый элемент, передав в качестве параметра DOM-элемент.

```
$("div").not($(".div:first").get(0))
.css("background-color", "red");
```

Теперь исключим из коллекции все четные элементы, передав массив DOM-элементов.

```
$("div").not($(".div:even").get())
.css("background-color", "red");
```

Такого же результата можно достичь, если передать объект jQuery.

```
$("div").not($("div:even"))
.css("background-color", "red");
```

- `filter()` — позволяет ограничить коллекцию дополнительным условием. В отличие от метода `not()`, не удаляет элементы, соответствующие селектору, а наоборот, оставляет только их. Метод имеет два формата.

```
filter(<Селектор>)
filter(<Функция обратного вызова>)
```

В первом формате метода передается селектор. Для примера выделим элемент с определенным идентификатором.

```
$("div").filter("#div1").css("background-color", "red");
```

Второй формат метода позволяет ограничить набор произвольным условием. В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Индекс>]) {
 // ...
}
```

Текущий элемент доступен внутри функции через указатель `this`. Обратите внимание, указатель `this` ссылается на текущий элемент объектной модели документа, а не на элемент коллекции jQuery. Текущий индекс элемента в коллекции доступен через параметр `<Индекс>`. Чтобы оставить элемент в наборе, необходимо в функции обратного вызова вернуть значение `true`. Выделим ссылку с определенным текстом.

```
$("a").css("color", "silver")
.filter(function() {
 if (this.innerHTML == "Ссылка 2") {
 return true;
 }
}).css("color", "red");
```

## 4.2. Привязка поиска к другим элементам

Привязать поиск элемента к другим элементам HTML-документа можно с помощью следующих методов.

- `find(<Селектор>)` — находит элементы, которые располагаются внутри контейнера. Возвращает новую коллекцию. Найдем все теги `<div>`, затем внутри них все теги `<span>` и изменим текст внутри этих тегов.

```
$("div").find("span").html("Новый текст");
```

```
<div>
<u>Текст 1</u>
Текст 2
</div>
```

- `children([<Селектор>])` — возвращает все дочерние узлы. Выделим все элементы внутри тегов `<div>`.

```
$("div").children().css("color", "red");
```

```
<div>
<u>Выделенный текст 1</u>
Текст 2
</div>
```

В результате будет выделен текст, который расположен между тегами `<u>` и `<b>`.

Если в качестве необязательного параметра указать селектор, то будут возвращены дочерние узлы, которые соответствуют селектору.

```
$("div").children("b").css("color", "red");
```

```
<div>
<u>Выделенный текст 1</u>
Текст 2
</div>
```

В этом примере будет выделен только фрагмент Текст 2, так как второй тег `<b>` расположен внутри тега `<u>`.

- `parent ([<Селектор>])` — находит контейнер, в который вложен элемент. Получим любые теги, в которые вложены теги `<span>`, и выведем новый текст вместо их содержимого.

```
$("span").parent().html("Это элемент-родитель");
```

```
<div>Текст</div>
Текст
<u>Текст</u>
```

Результат:

```
<div>Это элемент-родитель</div>
Это элемент-родитель
<u>Текст</u>
```

Если в качестве необязательного параметра указать селектор, то будут возвращены узлы, которые ему соответствуют. Найдем все теги `<div>`, в которые вложены теги `<span>`, и выведем новый текст вместо их содержимого.

```
$("span").parent("div").html("Это элемент-родитель");
```

```
<div>Текст</div>
Текст
<u>Текст</u>
```

Результат будет выглядеть следующим образом.

```
<div>Это элемент-родитель</div>
Текст
<u>Текст</u>
```

При щелчке на кнопке отобразим скрытый элемент, который находится в одном контейнере с ней, а затем удалим эту кнопку.

```
$(":button").click(function() {
 $(this) // Находим элемент
 .parent() // Получаем элемент-родитель
```

```

 .find("span:first") // Находим первый тег
 .show() // Отображаем его
 .end() // Возвращаемся к элементу-родителю
 .find(":button:first") // Находим внутри него первую кнопку
 .remove(); // Удаляем кнопку
};

<div>
<input type="button" value="Показать описание"> Термин 1

Описание термина 1
</div>
<div>
<input type="button" value="Показать описание"> Термин 2

Описание термина 2
</div>

```

- `closest(<Селектор>)` — возвращает ближайший родительский элемент, который соответствует селектору.

```
$("#span1").closest("div").html("Это элемент-родитель");
```

```
<div><div>
Текст
</div></div>
```

Результат:

```
<div><div>Это элемент-родитель</div></div>
```

- `parents([<Селектор>])` — возвращает все контейнеры, в которые вложен элемент. Если в качестве необязательного параметра указать селектор, то будут возвращены узлы, которые ему соответствуют. Получим все теги, в которые вложен элемент с идентификатором `span1`, и выведем их названия.

```
var elem = $("#div1");
$("#span1").parents().each(function() {
 elem.append(this.tagName + " ");
});
```

```
<div><u>
Текст
</u></div>
<div id="div1"></div>
```

Результат:

```
b u div
```

- `siblings([<Селектор>])` — находит все узлы, которые расположены до или после элемента на одном уровне вложенности. Если в качестве необязательного параметра указать селектор, то будут возвращены узлы, которые ему соответствуют.

```
 $("div").siblings("a").css("color", "red");
```

```
Ссылка 1

Текст

<div>Текст</div>
Текст

```

```
Ссылка 2

Ссылка 3

Ссылка 4

```

Цвет текста ссылки станет красным, если тег `<a>` следует до или после тега `<div>`. В этом примере ссылки 1, 2 и 4 станут красного цвета. Ссылка 3 не станет красного цвета, так как расположена внутри тега `<span>`.

- `next ([<Селектор>])` — находит узел, который является соседним для элемента и следует сразу после него. Если в качестве необязательного параметра указать селектор, то будут возвращены узлы, которые ему соответствуют. Выделим содержимое тега, который следует сразу за элементом с идентификатором `span1`.

```
$("#span1").next().css("color", "red");
```

```
Текст
Текст1
<u>Текст2</u>
```

В результате будет выделен фрагмент `Текст1`.

- `nextAll ([<Селектор>])` — возвращает все узлы, которые следуют после элемента на том же уровне вложенности. Если в качестве необязательного параметра указать селектор, то будут возвращены узлы, которые ему соответствуют. Получим все названия тегов, которые следуют после элемента с идентификатором `span1`.

```
var elem = $("#div1");
$("#span1").nextAll().each(function() {
 elem.append(this.tagName + " ");
});
```

```
Текст
Текст
<u>Текст</u>
<div id="div1"></div>
```

Результат:

```
b u div
```

Обратите внимание: мы не получили второй тег `<b>`, так как он расположен внутри тега `<u>`.

- `prev ([<Селектор>])` — находит узел, который является соседним для элемента и находится сразу перед ним. Если в качестве необязательного параметра указать селектор, то будут возвращены узлы, которые ему соответствуют. Выделим содержимое тега, который находится перед элементом с идентификатором `span1`.

```
$("#span1").prev().css("color", "red");
```

```
<u>Текст2</u>
Текст1
Текст
```

В результате будет выделен фрагмент `Текст1`.

- `prevAll([<Селектор>])` — возвращает все узлы, которые находятся перед элементом на том же уровне вложенности. Если в качестве необязательного параметра указать селектор, то будут возвращены узлы, которые ему соответствуют. Получим все названия тегов, которые находятся перед элементом с идентификатором `span1`.

```
var elem = $("#div1");
$("#span1").prevAll().each(function() {
 elem.append(this.tagName + " ");
});
```

```
<u>Текст2</u>
Текст1
Текст
<div id="div1"></div>
```

Результат:

```
b u
```

Обратите внимание: мы не получили первый тег `<b>`, так как он расположен внутри тега `<u>`.

- `contents()` — находит дочерние узлы (включая текстовые) в коллекции элементов или в содержимом документа, если он представляет собой фрейм. Переместим все текстовые узлы в конец элемента с идентификатором `div1`.

```
$("div").contents().not("[nodeType=1]").appendTo("#div1");
```

```
<div>Текстовый узел1 узел элемента Текстовый
узел2</div>
<div id="div1"></div>
```

- `andSelf()` — объединяет предыдущую коллекцию с новой.

```
$("div").children("a")
// Объединяем коллекцию тегов <div>
// с коллекцией тегов <a>, являющихся
// дочерними элементами тегов <div>
.andSelf()
.css({ "border-style": "dotted", margin: "20px" });
```

```
<div>

Ссылка 1

Ссылка 2

Ссылка 3

Ссылка 4

</div>
```

В этом примере содержимое тега `<div>`, а также ссылки 1, 2 и 4 будут обведены пунктирной рамкой. Ссылка 3 не будет обведена, так как расположена внутри тега `<span>`.

- `end()` — позволяет вернуться к предыдущей коллекции. В качестве примера получим коллекцию всех тегов `<div>`, внутри которой найдем элемент с идентификатором `span1` и выведем в нем соответствующий текст. Затем вернемся к пре-

дыущей коллекции всех тегов `<div>`, в которой найдем элемент с идентификатором `span2` и также выведем в нем соответствующий текст.

```
$("div")
 //Находим элемент с id="span1"
 .find("#span1").html("Это span1")
 // Возвращаемся на предыдущую коллекцию
 .end()
 //Находим элемент с id="span2"
 .find("#span2").html("Это span2");
```

- `is(<Селектор>)` — проверяет коллекцию или элемент на соответствие селектору. В качестве параметра нельзя указывать сложносоставные селекторы (например, условия `+`, `~`, `и` `>`). Возвращает `true`, если элемент соответствует селектору, и `false` — в противном случае. Выполним подтверждающее сообщение, если элемент с идентификатором `link1` расположен внутри тега `<div>`, а в противном случае — название тега, в который вложен элемент.

```
var elem = $("#link1").parent();
if (elem.is("div")) {
 alert("Элемент вложен в тег <div>");
}
else {
 alert("Элемент вложен в тег " + elem[0].tagName);
}
```

## 4.3. Метод map()

Метод `map()` позволяет создать новую коллекцию элементов jQuery на основе имеющегося набора. Имеет следующий синтаксис.

```
map(<Функция обратного вызова>);
```

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Индекс>[, <DOM-элемент>]]) {
 // ...
}
```

Текущий элемент доступен внутри функции через указатель `this`. Обратите внимание, указатель `this` ссылается на текущий элемент объектной модели документа, а не на элемент коллекции jQuery. Если в параметре `<DOM-элемент>` указана переменная, то текущий элемент также будет доступен через эту переменную. Текущий индекс элемента в коллекции доступен через параметр `<Индекс>`. Выделим все четные ссылки в документе.

```
 $("a").map(function(ind, elem) {
 if (ind%2 == 0) {
 return elem;
 }
 else {
 return null;
 }
}).css("color", "red");
```

Чтобы добавить текущий элемент в новый набор, необходимо вернуть его внутри функции обратного вызова. Добавить можно сразу несколько элементов, вернув их как массив. Если в качестве значения вернуть null, то элемент не будет добавлен в новую коллекцию.

В листинге 3.1 мы получали URL-адреса всех ссылок в документе с помощью метода `each()`. Переделаем наш пример и используем для этой цели метод `map()`.

```
var links = $("a").map(function() {
 return $(this).attr("href");
}).get().join(",");
$("#div1").text(links);
```

Как видно из примера, мы можем создавать коллекцию не только элементов, но и значений. Преобразовать эту коллекцию в обычный массив позволяет метод `get()`. С помощью метода JavaScript `join()` преобразуем массив в строку, где все элементы массива будут перечислены через запятую.

В качестве еще одного примера использования метода `map()` выделим все ссылки с определенным текстом (листинг 4.1).

#### Листинг 4.1. Выделение ссылок в зависимости от текста

```
<html>
<head>
<title>Функция map()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
$(document).ready(function() {
 $("a").map(function() {
 var txt = $(this).text();
 if (txt == "Ссылка 1" || txt == "Ссылка 3") {
 return this;
 }
 else {
 return null; // Удаляем элемент
 }
 }).css("color", "red");
});
</script>
</head>
<body>
Ссылка 1

Ссылка 2

Ссылка 3

Ссылка 4
</body>
</html>
```

В этом примере будут выделены все ссылки, содержащие текст Ссылка 1 и Ссылка 3.

## Глава 5

# Манипуляции с элементами

Получать коллекцию элементов и осуществлять доступ к отдельному ее элементу мы уже научились. Теперь рассмотрим различные методы, позволяющие изменять свойства и значения элементов коллекции. Некоторые методы мы уже использовали в наших примерах для вывода результата или изменения свойств элементов.

## 5.1. Изменение содержимого элементов

Наиболее часто используемой операцией является изменение содержимого элементов. Вне библиотеки jQuery для этой цели используется свойство `innerHTML` объекта `document`. Например, чтобы вставить новый фрагмент в элемент с идентификатором `div1`, можно воспользоваться следующим кодом.

```
document.getElementById("div1").innerHTML = "Новый текст";
```

Библиотека jQuery также позволяет использовать это свойство. Для этого находим элемент с помощью функции `$()`, а затем преобразуем коллекцию элементов jQuery в набор DOM-элементов.

```
$("#div1")[0].innerHTML = "Новый текст";
```

Для этой цели можно также воспользоваться методом `get()`.

```
$("#div1").get(0).innerHTML = "Новый текст";
```

В библиотеке jQuery для изменения содержимого элементов предназначены следующие методы.

- `text([<Значение>])` — позволяет получить или задать текст элемента. Если параметр не указан, то метод возвращает текстовое значение без тегов. При наличии нескольких элементов в коллекции будут возвращены все значения в виде строки. Если необходимо получить значение только первого элемента из коллекции, то можно воспользоваться методом `eq()`.

```
alert($("#div").eq(0).text());
```

Можно также ограничить набор с помощью селектора `:first`.

```
alert($("#div:first").text());
```

При указании строки в качестве параметра она заменит содержимое всех элементов коллекции. В случае наличия в строке специальных символов они будут заменены на HTML-эквиваленты.

```
$("#div").text("Новое значение");
```

В этом примере строка в исходном HTML-коде будет выглядеть следующим образом.

```
Новое значение
```

- `html([<Значение>])` — позволяет получить или задать HTML-код элемента. Если параметр не указан, то метод возвращает значение вместе с тегами. При наличии нескольких элементов в коллекции будет возвращено значение только первого элемента.

```
alert($("#div").html());
```

При указании HTML-кода в качестве параметра он заменит содержимое всех элементов коллекции, и объектная модель документа будет обновлена. Выведем HTML-код во всех тегах `<div>`.

```
$("div").text(" Новое значение ");
```

Если необходимо изменить значение только первого элемента из коллекции, то можно воспользоваться методом `eq()` или селектором `:first`.

```
$("div:first").html(" Новое значение ");
```

- `append(<Выражение>)` — добавляет `<Выражение>` в конец содержимого выбранного элемента. В качестве параметра может быть указан HTML-код, DOM-элемент или коллекция элементов jQuery. Для примера добавим HTML-код в конец элемента.

```
$("#div1").append(" <u>Новый текст</u> ");
```

```
<div id="div1">
 Текст
</div>
```

Результат будет выглядеть следующим образом.

```
<div id="div1">
 Текст
 <u>Новый текст</u></div>
```

Теперь добавим DOM-элемент.

```
$("#div1").append($(". <u>Новый текст</u> ").get(0));
```

А теперь добавим созданный элемент коллекции jQuery.

```
$("#div1").append($(". <u>Новый текст</u> "));
```

- `prepend(<Выражение>)` — добавляет `<Выражение>` в начало содержимого выбранного элемента. В качестве параметра может быть указан HTML-код, DOM-элемент или коллекция элементов jQuery. Метод полностью идентичен методу `append()`, за исключением того, что добавляет `<Выражение>` не в конец элемента, а в его начало. Для примера добавим HTML-код в начало элемента.

```
$("#div1").prepend(" <u>Новый текст</u> ");
```

```
<div id="div1">
 Текст
</div>
```

Результат будет выглядеть следующим образом.

```
<div id="div1"><u>Новый текст</u>

 Текст
</div>
```

- `appendTo(<Селектор>)` — добавляет коллекцию элементов jQuery в конец всех элементов, соответствующих указанному селектору. Для примера добавим HTML-код в конец элемента с идентификатором `div1`.

```
$("<u>Новый текст</u>").appendTo("#div1");
```

Результат будет таким же, как и при использовании метода `append()`.

```
$("#div1").append("<u>Новый текст</u>");
```

Как видно из примера, мы поменяли параметры местами и использовали метод `append()` вместо метода `appendTo()`.

- `prependTo(<Селектор>)` — добавляет коллекцию элементов jQuery в начало всех элементов, соответствующих указанному селектору. Для примера добавим HTML-код в начало элемента с идентификатором `div1`.

```
$("<u>Новый текст</u>").prependTo("#div1");
```

Результат будет таким же, как и при использовании метода `prepend()`.

```
$("#div1").prepend("<u>Новый текст</u>");
```

## 5.2. Добавление содержимого перед элементом или после него

В предыдущем разделе мы рассмотрели изменение внутреннего содержимого элемента. Библиотека jQuery предоставляет также методы, которые позволяют добавить какое-либо содержимое перед элементом или после него. Рассмотрим эти методы.

- `after(<Выражение>)` — добавляет `<Выражение>` после всех элементов коллекции. В качестве параметра может быть указан HTML-код, DOM-элемент или коллекция элементов jQuery. Для примера добавим HTML-код после элемента.

```
$("#div1").after("<u>Новый текст</u>");
```

```
<div id="div1">Текст</div>
```

Результат будет выглядеть следующим образом.

```
<div id="div1">Текст</div><u>Новый текст</u>
```

Теперь добавим DOM-элемент.

```
$("#div1").after($("<u>Новый текст</u>").get(0));
```

А теперь добавим созданный элемент коллекции jQuery.

```
$("#div1").after($("<u>Новый текст</u>"));
```

- `before(<Выражение>)` — добавляет `<Выражение>` перед всеми элементами коллекции. В качестве параметра может быть указан HTML-код, DOM-элемент или коллекция элементов jQuery. Метод полностью идентичен методу `after()`, за исключением того, что добавляет `<Выражение>` не после элемента, а перед ним. Для примера добавим HTML-код перед элементом.

```
$("#div1").before("<u>Новый текст</u>");
```

```
<div id="div1">Текст</div>
```

Результат будет выглядеть следующим образом.

```
<u>Новый текст</u><div id="div1">Текст</div>
```

- `insertAfter(<Селектор>)` — добавляет коллекцию элементов jQuery после всех элементов, соответствующих указанному селектору. Для примера добавим HTML-код после элемента с идентификатором `div1`.

```
$("<u>Новый текст</u>").insertAfter("#div1");
```

Результат будет таким же, как и при использовании метода `after()`.

```
$("#div1").after("<u>Новый текст</u>");
```

- `insertBefore(<Селектор>)` — добавляет коллекцию элементов jQuery перед всеми элементами, соответствующими указанному селектору. Для примера добавим HTML-код перед элементом с идентификатором `div1`.

```
$("<u>Новый текст</u>").insertBefore("#div1");
```

Результат будет таким же, как и при использовании метода `before()`.

```
$("#div1").before("<u>Новый текст</u>");
```

## 5.3. Вложение элементов

Все элементы коллекции можно разместить в каком-либо другом элементе. Для вложения элементов предназначены следующие методы.

- `wrapInner(<HTML-элемент или DOM-элемент>)` — вкладывает внутреннее содержимое каждого элемента коллекции в другой элемент. Для примера выделим содержимое всех тегов `<div>`.

```
$("div").wrapInner(" ");
```

```
<div>Текст 1</div>
<div>Текст 2</div>
```

Результат будет выглядеть следующим образом.

```
<div>Текст 1</div>
<div>Текст 2</div>
```

Такого же эффекта можно достичь, передав в качестве параметра DOM-элемент.

```
$("div").wrapInner(document.createElement("b"));
```

Можно также передать элемент, созданный с помощью функции `$()`.

```
$("div").wrapInner($(" "));
```

- `wrap(<HTML-элемент или DOM-элемент>)` — полностью вкладывает каждый элемент коллекции в другой элемент.

```
$("div").wrap(" ");
```

```
<div>Текст 1</div>
<div>Текст 2</div>
```

Результат выполнения таков:

```
<div>Текст 1</div>
<div>Текст 2</div>
```

- `wrapAll(<HTML-элемент или DOM-элемент>)` — собирает все элементы коллекции в одном месте и вкладывает их в другой элемент.

```
$("div").wrapAll("<u></u>");

Какой-то текст 1
<div>Текст 1</div>
Какой-то текст 2
<div>Текст 2</div>
Какой-то текст 3
<div>Текст 3</div>
```

Результат будет выглядеть следующим образом.

```
Какой-то текст 1
<u><div>Текст 1</div><div>Текст 2</div><div>Текст 3</div></u>
Какой-то текст 2
Какой-то текст 3
```

Как видно из примера, все элементы коллекции были размещены после первого элемента и расположены внутри тега `<u>`.

## 5.4. Перемещение и клонирование элементов

Если в качестве параметра методов `before()`, `prepend()`, `append()` и `after()` указать коллекцию существующих элементов jQuery, то они будут перемещены. Куда будут вставлены элементы, зависит от конкретного метода: `before()` (перед элементом), `prepend()` (в начало содержимого), `append()` (в конец содержимого), `after()` (после элемента). Для примера найдем все ссылки на странице и добавим их в конец элемента.

```
$("#div1").append($(".a"));

<div id="div1">
Текст
</div>
1

2

```

В результате все ссылки будут перемещены в конец содержимого элемента с идентификатором `div1`, и мы получим следующий HTML-код.

```
<div id="div1">
Текст12
</div>


```

Такого же эффекта можно достичь при помощи методов `insertBefore()`, `prependTo()`, `appendTo()` и `insertAfter()`. Куда будут вставлены элементы, зависит от конкретного метода: `insertBefore()` (перед элементом), `prependTo()` (в начало содержимого), `appendTo()` (в конец содержимого), `insertAfter()` (после элемента). Для примера найдем все ссылки на странице и добавим их перед элементом.

```
$("a").insertBefore($("#div1"));

<div id="div1">
Текст</div>

1

2

```

В результате все ссылки будут размещены непосредственно перед элементом с идентификатором div1, и мы получим следующий HTML-код.

```
12<div id="div1">
Текст</div>


```

Библиотека jQuery позволяет создавать копии существующих элементов (клонировать). Для этого предназначен метод `clone([true])`. Если в качестве параметра указано значение `true`, то будут также клонированы и обработчики событий. Создадим копию первой ссылки в документе, а затем выведем ее после элемента с идентификатором div1.

```
$("a").eq(0).clone().insertAfter("#div1");

Ссылка
<div id="div1">Текст</div>
```

Результат будет выглядеть следующим образом.

```
Ссылка
<div id="div1">Текст</div>Ссылка
```

## 5.5. Очистка содержимого и удаление элемента

Для очистки содержимого и удаления элемента применяются следующие методы.

- `empty()` — удаляет все подэлементы текущего элемента.

```
<div id="div1">Этот текст будет удален</div>
<input type="button" value="Очистить"
onclick="$('div1').empty();">
<input type="button" value="Вставить"
onclick="$('div1').html('Новый текст');" >
```

Как видно из примера, после удаления содержимого элемента с идентификатором div1 сам элемент все еще остается доступным для манипуляций.

- `remove([<Селектор>])` — полностью удаляет элементы из объектной модели документа.

```
<div id="div1">Этот элемент будет полностью уда-
лен</div>
<input type="button" value="Удалить"
onclick="$('div1').remove();">
<input type="button" value="Количество элементов"
onclick="alert($('#div1').size());">
```

Данный пример демонстрирует отсутствие элемента после щелчка на кнопке Удалить. Щелкнув на кнопке Количество элементов в первый раз, мы получим число 1, а если щелкнуть на ней после удаления элемента, то получим число 0.

Если коллекция состоит более, чем из одного элемента, то будут удалены все элементы. Метод `remove()` позволяет задать дополнительное условие, которому должны соответствовать удаляемые элементы. В качестве примера удалим все ссылки с расширением .php.

```
$("a").remove("[href$=' .php']");

```

## 5.6. Замена элемента

Для полной замены одного элемента на другой предназначены два метода.

- `replaceWith(<Выражение>)` — заменяет все элементы коллекции на `<Выражение>`. В качестве параметра может быть указан HTML-код, DOM-элемент или коллекция элементов jQuery. Для примера заменим элемент с идентификатором `div1` на новый элемент при щелчке на нем.

```
$("#div1").click(function() {
 $("#div1").replaceWith("<u>Новый элемент</u>");
});

```

```
<div id="div1">Нажмите здесь</div>

```

Теперь заменим на DOM-элемент.

```
$("#div1").replaceWith($(".<u>Новый элемент</u>").get(0));

```

А теперь заменим на созданный элемент коллекции jQuery.

```
$("#div1").replaceWith($(".<u>Новый элемент</u>"));

```

Если указать существующую коллекцию элементов jQuery, то элементы будут перемещены.

```
$("#div1").replaceWith($(".p"));

```

```
<div id="div1">Нажмите здесь</div>

```

Какой-то текст  
`<br>`

```
<p>Абзац1</p>

```

```
<p>Абзац2</p>

```

В результате все абзацы будут перемещены вместо элемента с идентификатором `div1`, и мы получим следующий HTML-код.

```
<p>Абзац1</p><p>Абзац2</p>

```

Какой-то текст  
`<br>`

- `replaceAll(<Селектор>)` — вставляет созданный элемент вместо всех элементов, соответствующих указанному селектору. Заменим элемент с идентификатором `div1` на другой элемент.

```
$(".<u>Новый элемент</u>").replaceAll("#div1");

```

Результат будет таким же, как и при использовании метода `replaceWith()`.

```
$("#div1").replaceWith("<u>Новый элемент</u>");

```

Как видно из примера, мы поменяли параметры местами и использовали метод `replaceWith()` вместо метода `replaceAll()`.

## 5.7. Изменение атрибутов CSS

Для изменения атрибутов стилей предназначены следующие методы.

- `css()` — позволяет получить или установить значение отдельных атрибутов CSS. Метод имеет несколько форматов.

```
css(<Название атрибута>)
css(<Название атрибута>, <Значение>)
css(<Объект с атрибутами>)
css(<Название атрибута>, <Функция обратного вызова>)
```

Первый формат метода позволяет получить значение указанного атрибута для первого элемента коллекции. Значение возвращается в виде строки.

Составное название атрибута может быть таким, как принято в CSS (например, `background-color`) или в JavaScript (например, `backgroundColor`). В качестве примера выведем цвет фона первого тега `<div>`.

```
alert($(".div").css("background-color"));
```

```
<div id="div1" style="background-color:red">Текст</div>
<div id="div2" style="background-color:blue">Текст</div>
```

Результат выполнения этого примера зависит от браузера. Так браузер Opera возвращает значение "#ff0000", в то время как браузер Internet Explorer 7 — название цвета ("red"). Иными словами, Internet Explorer возвращает значение, указанное в атрибуте стиля. Если указать значение "#ff0000":

```
<div id="div1" style="background-color:#ff0000">Текст</div>,
то он вернет именно это значение. В случае, если цвет фона элемента не задан, то мы получим значение "transparent".
```

Второй формат метода `css()` позволяет задать значение указанного атрибута для всех элементов коллекции. Сделаем цвет фона всех абзацев красным.

```
$(".p").css("background-color", "#ff0000");
```

Метод `css()` возвращает указатель на коллекцию jQuery. По этой причине мы можем изменить еще несколько атрибутов, составив цепочку вызовов функции `css()`. Сделаем цвет фона всех абзацев красным, а цвет текста — белым.

```
$(".p").css("backgroundColor", "red").css("color", "#ffffff");
```

Третий формат метода `css()` позволяет задать сразу несколько атрибутов за один вызов функции. Для этого атрибуты и значения должны быть указаны следующим образом.

```
{
Атрибут1: "Значение1",
Атрибут2: "Значение2",
...
АтрибутN: "ЗначениеN"
}
```

Если название атрибута содержит дефис, то название необходимо заключить в кавычки. Для примера сделаем цвет фона всех заголовков первого уровня черным, цвет текста — белым, а также зададим размер и название шрифта.

```
$("h1").css(
{
 backgroundColor: "#000000",
 color: "#ffffff",
 "font-size": "18pt",
 "font-family": "Tahoma"
}
);
```

Такой объект можно сохранить в переменной, а затем передать его в качестве параметра метода `css()`.

```
var obj = { fontSize: "18pt", fontFamily: "Tahoma" }
$("h1").css(obj);
```

Четвертый формат метода `css()` позволяет выборочно задать значение указанного атрибута, в зависимости от каких-либо условий. В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Индекс>]) {
 // ...
}
```

На каждой итерации в функцию передается ссылка (`this`) на текущий DOM-элемент. Чтобы задать значение атрибуту, необходимо внутри функции обратного вызова вернуть новое значение. Если в параметре указать переменную, то через нее будет доступен индекс текущего элемента в коллекции. В качестве примера сделаем так, чтобы все ссылки, имеющие абсолютный URL-адрес, были выделены красным цветом.

```
$("a").css("color", function() {
 var pattern = /^http:\/\//i;
 if (pattern.test(this.href)) {
 return "red";
 }
 else {
 return "black";
 }
});
```

- `width()` и `height()` — позволяют получить ширину и высоту первого элемента коллекции соответственно. Возвращают значение в виде числа. Для примера выведем ширину и высоту доступной области окна браузера.

```
var w = $(window).width();
var h = $(window).height();
$("#div1").html("Ширина: " + w + "
Высота: " + h);
```

В результате будет выведено, например, следующее.

```
Ширина: 1022
Высота: 734
```

В качестве еще одного примера выведем ширину и высоту элемента с помощью методов `width()` и `height()`, а для сравнения выведем также значения соответствующих атрибутов стиля.

```
var elem = $("#div1");
$("#div1").append("Значения методов:
");
$("#div1").append("Ширина: " + elem.width() + "
");
$("#div1").append("Высота: " + elem.height() + "
");
$("#div1").append("Значения атрибутов стиля:
");
$("#div1").append("Ширина: " + elem.css("width") + "
");
$("#div1").append("Высота: " + elem.css("height") + "
");

<div id="div1"
style="width:300px;height:200px; border:dotted;">
</div>
```

В результате будет выведено следующее:

Значения методов:

Ширина: 300

Высота: 200

Значения атрибутов стиля:

Ширина: 300px

Высота: 200px

Как видно из примера, значения атрибутов стиля содержат единицы измерения, тогда как методы возвращают значение в виде числа.

- `width(<Значение>)` и `height(<Значение>)` — позволяют задать ширину и высоту всех элементов коллекции соответственно. Возвращают объект jQuery. Зададим ширину всех тегов `<div>`, а затем укажем цвет фона.

```
$("div").width(50).css("background-color", "#ff0000");
```

Значением параметра может быть не только число, но и строка.

```
$("#div1").width("500px");
```

```
$("#div2").width("90%");
```

- `innerWidth()` и `innerHeight()` — возвращают соответственно ширину и высоту первого элемента коллекции. Методы учитывают значение атрибута `padding`, но не учитывают толщину линии рамки.
- `outerWidth([true])` и `outerHeight([true])` — возвращают соответственно ширину и высоту первого элемента коллекции. Методы учитывают значение атрибута `padding` и толщину линии рамки. Если в качестве параметра указать значение `true`, то методы будут также учитывать значение атрибута `margin`.

Рассмотрим возможность изменения ширины и высоты элемента путем задания произвольных числовых значений, которые можно вводить в текстовые поля (листинг 5.1).

### Листинг 5.1. Методы `width()` и `height()`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Методы width() и height()</title>
```

```

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
function showValue() {
 var elem1 = $("#div1");
 var msg = "Высота: " + elem1.height() + "
inner: Ширина: ";
 msg += elem1.innerWidth() + " Высота: " + elem1.innerHeight();
 msg += "
outer: Ширина: " + elem1.outerWidth(true) + " Высота: ";
 msg += elem1.outerHeight(true);
 $("#div2")
 .html("Ширина: " + elem1.width() + " ")
 .append(msg);
}
$(document).ready(function() {
 $("button").click(function() {
 var w = $("#txt1").val();
 var h = $("#txt2").val();
 var elem = $("#div1");
 if (w != '') {
 w = parseInt(w);
 if (!isNaN(w)) {
 elem.width(w);
 $("#txt1").val("");
 }
 else alert("Необходимо ввести число!");
 }
 if (h != '') {
 h = parseInt(h);
 if (!isNaN(h)) {
 elem.height(h);
 $("#txt2").val("");
 }
 else alert("Необходимо ввести число!");
 }
 showValue();
 });
 showValue();
});
//-->
</script>
<style>
#div1 type="text/css"{
 padding:5px; margin:5px;
 width:300px; height:200px;
 border: black 2px dotted;
}
</style>
</head>
<body>
<div id="div1"></div>

Текущие значения:

<div id="div2"></div>

Ширина:
<input type="text" id="txt1">

Высота:
<input type="text" id="txt2">


```

```
<input type="button" value="Задать новые значения">
</body>
</html>
```

Если ввести значение и щелкнуть на кнопке Задать новые значения, будет вызван обработчик события `onclick`. С помощью метода `val()` мы получаем значения, введенные в текстовые поля, и сохраняем их в переменных. Далее проверяем значения на пустоту. Если поле не пустое, то пытаемся преобразовать строку в число. Прежде чем присвоить новое значение, проверяем, не вернул ли метод `parseInt()` значение `Nan`, с помощью метода `isNaN()`. Далее задаем новое значение и очищаем поле ввода. Если в поле была введена строка, а не число, то выводим предупреждающее сообщение. Затем обновляем данные в элементе с идентификатором `div2` при помощи функции `showValue()`. Обратите внимание на следующие строки.

```
$("#div2")
.html("Ширина: " + elem1.width() + " ")
.append(msg);
```

Здесь мы вначале получаем элемент, а затем выводим ширину с помощью метода `html()`. После изменения содержимого элемента метод `html()` возвращает ссылку на полученный ранее элемент. По этой причине мы можем производить с ним дальнейшие манипуляции. В данном случае мы добавили в конец элемента значения других методов. Создание таких цепочек вызовов — характерная особенность библиотеки `jQuery`. Практически все методы возвращают объект `jQuery`, если, конечно, не возвращают какого-либо другого значения. Так, например, метод `width()` без указания параметра возвращает значение в виде числа, а при указании параметра — объект `jQuery`.

Обратите также внимание на расположение методов в цепочке. Мы можем указывать каждый метод в отдельной строке. Более того, можно между вызовами методов в цепочки вставлять комментарии.

```
$("#div2")
// Выводим ширину элемента
.html("Ширина: " + elem1.width() + " ")
// Выводим остальные значения
.append(msg);
```

## 5.8. Управление классами стилей

Библиотека `jQuery` позволяет не только изменять определенные атрибуты стиля, но и управлять классами. Для этого предназначены следующие методы.

- `addClass(<Название класса>)` — добавляет указанный класс всем элементам коллекции. В качестве параметра можно указать несколько названий классов через пробел.
- `removeClass(<Название класса>)` — удаляет указанный класс у всех элементов коллекции. В качестве параметра можно указать несколько классов через пробел. Если название класса не указано, то будут удалены все классы.
- `toggleClass(<Название класса> [, <Условие>])` — добавляет указанный класс всем элементам коллекции, если он не был определен ранее, или удаляет указанный класс, если он был добавлен ранее. Если в необязательном параметре

<Условие> задано значение `true`, то указанный класс будет добавлен, а если задано значение `false`, то указанный класс будет удален.

Все указанные методы в качестве значения возвращают коллекцию элементов jQuery. Рассмотрим возможность изменения класса при наведении курсора на элемент (листинг 5.2).

### Листинг 5.2. Методы `addClass()`, `removeClass()` и `toggleClass()`

```
<html>
<head>
<title>Методы addClass(), removeClass() и toggleClass()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
$(document).ready(function() {
 $("#div1").hover(
 function() { // Если навести курсор
 $(this).removeClass().addClass("cls2");
 },
 function() { // Если убрать курсор
 $(this).removeClass().addClass("cls1");
 }
);
 $("#div2").hover(
 function() { // Если навести курсор
 $(this).toggleClass("cls2");
 },
 function() { // Если убрать курсор
 $(this).toggleClass("cls2");
 }
);
});
</script>
<style type="text/css">
.cls1 { color:#000000; text-decoration:none; }
.cls2 { color:#ff0000; text-decoration:underline; }
</style>
</head>
<body>

Наведите курсор на этот текст</div>

Наведите курсор на этот текст</div>
</body>
</html>


```

В этом примере мы определили два класса `cls1` и `cls2`. Класс `cls1` описывает стиль элемента по умолчанию. Класс `cls2` предназначен для описания стиля элемента при наведении курсора. После формирования структуры документа мы добавляем обработчики событий с помощью метода `hover()`. В первом параметре указывается функция, которая будет вызвана при наведении курсора. Во втором параметре указывается функция, которая будет вызвана, если курсор убрать с элемента. Внутри этих обработчиков текущий DOM-элемент доступен через указатель `this`. Если курсор будет наведен на первую надпись, то удаляем все классы, а затем добавляем класс `cls2`.

```
$(this).removeClass().addClass("cls2");
```

Если убрать курсор с элемента, то проделываем обратную операцию. Удаляем все классы и добавляем класс по умолчанию.

Вторая надпись демонстрирует применение метода `toggleClass()`. Если навести курсор, то метод проверит, был ли класс `cls2` добавлен ранее. В случае отсутствия класса он будет добавлен, в противном случае удален. По этой причине во всех событиях мы указываем одну и ту же строку.

```
$(this).toggleClass("cls2");
```

Метод `hasClass(<Название класса>)` позволяет определить, был ли указанный класс добавлен ранее. Возвращает `true`, если класс был добавлен ранее.

```
function checkClass(cls) {
 if ($("#div1").hasClass(cls)) {
 alert("Определен");
 }
 else {
 alert("Нет");
 }
}
checkClass("cls2"); // Нет
$("#div1").addClass("cls2");
checkClass("cls2"); // Определен
checkClass("cls1"); // Определен

<div id="div1" class="cls1">Текст</div>
```

При первой проверке получим сообщение, что класс `cls2` не определен. Далее добавляем класс и опять проверяем. В этом случае получим сообщение, что класс определен для элемента с идентификатором `div1`. Обратите внимание на тот факт, что добавление нового класса не удаляет другие классы, определенные ранее.

## 5.9. Доступ к параметрам тегов

Для доступа к параметрам тегов предназначены следующие методы.

- `attr()` — позволяет получить или установить значение отдельных параметров тегов. Метод имеет несколько форматов.

```
attr(<Название параметра>)
attr(<Название параметра>, <Значение>)
attr(<Объект с параметрами>)
attr(<Название параметра>, <Функция обратного вызова>)
```

Первый формат метода позволяет получить значение указанного параметра для первого элемента коллекции. Если параметр не найден, то возвращается значение `undefined`. Получим значение параметра `id` первого тега `<div>`.

```
alert($("#div").attr("id")); // div1

<div id="div1">Текст</div>

<div id="div2">Текст</div>
```

Второй формат метода `attr()` позволяет задать значение указанного параметра для всех элементов коллекции. В качестве примера создадим текстовое поле, в котором по умолчанию будет подсказка пользователю. Если поле получит фо-

кус ввода, то подсказка должна исчезнуть. Если в поле не было введено значение, то при потере фокуса подсказка должна опять появиться в поле.

```
$("#txt1").focus(function() {
 var elem = $(this);
 if (elem.attr("value") == "[Введите текст]")
 elem.attr("value", "");
})
$("#txt1").blur(function() {
 var elem = $(this);
 if (elem.attr("value") == "")
 elem.attr("value", "[Введите текст]");
})
});
```

```
<input type="text" id="txt1" value="[Введите текст]">
```

Задать значение параметрам, которые в HTML указываются без значений, можно одним из двух способов.

```
$("#btn1").attr("disabled", true);
$("#btn1").attr("disabled", "disabled");
```

Третий формат метода `attr()` позволяет задать сразу несколько параметров за один вызов метода. Для этого параметры и значения должны быть указаны следующим образом.

```
{
Параметр1: "Значениe1",
Параметр2: "Значениe2",
...
ПараметрN: "ЗначениeN"
}
```

Зададим URL-адрес и текст подсказки всем ссылкам.

```
 $("a").attr(
{
 href: "link1.html",
 title: "Нажми меня"
});
);
```

Четвертый формат метода `attr()` позволяет выборочно задать значение указанного параметра, в зависимости от каких-либо условий. В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Индекс>]) {
 // ...
}
```

На каждой итерации в функцию передается ссылка (`this`) на текущий DOM-элемент. Чтобы задать значение параметру, необходимо внутри функции обратного вызова вернуть новое значение. Если в параметре указать переменную, то через нее будет доступен индекс текущего элемента в коллекции. В качестве примера сделаем так, чтобы все ссылки, имеющие абсолютный URL-адрес, открывались в новом окне.

```
$("a").attr("target", function() {
 var pattern = /^http:\/\//i;
 if (pattern.test(this.href)) {
 return "_blank";
 }
 else {
 return "_self";
 }
});
```

- `removeAttr(<Название параметра>)` — удаляет указанный параметр у всех элементов коллекции. В качестве примера реализуем возможность установки и снятия всех флагжков при щелчках на соответствующих кнопках.

```
<input type="checkbox" checked>Флажок 1

<input type="checkbox" checked>Флажок 2

<input type="button" value="Снять все флагжи"
 onclick="$(':checkbox:checked').removeAttr('checked');">
<input type="button" value="Установить все флагжи"
 onclick="$(':checkbox').attr('checked', true);">
```

- `val()` — позволяет получить или установить значение параметра `value`. Метод имеет несколько форматов.

```
val()
val(<Значение>)
val(<Массив>)
```

Первый формат метода позволяет получить значение параметра `value` для первого элемента коллекции. Выведем значение текстового поля.

```
alert($(".text").val());
```

Если элемент позволяет выбрать несколько значений, то метод возвращает массив значений. Для примера получим все значения выбранных пунктов списка с множественным выбором..

```
var vals = $("#sel1").val() || [];
alert(vals.join(", "));
```

Следует заметить, что метод `val()` не дает представления, какой из флагжков или переключателей установлен, так как возвращает значение первого элемента. Чтобы получить значение выбранного переключателя или установленного флагжа, необходимо воспользоваться селектором `:checked`. Если элемент не найден, то возвращается значение `undefined`.

```
alert($(".radio:checked").val());
```

Второй формат метода `val()` позволяет задать значение всем элементам коллекции. Выведем текст в первом текстовом поле.

```
$(".text:first").val("Новое значение");
```

Третий формат метода предназначен для установки флагжков, выбора переключателя и выделения пунктов списка. В качестве параметра необходимо передать массив значений, которые должны быть установлены. Рассмотрим пример выбора переключателя.

```
$(":radio").val(["male"]);
```

Если найден переключатель со значением "male", то он будет выбран. Рассмотрим пример установки нескольких флагов и выделения пунктов списка с множественным выбором.

```
$(":checkbox").val(["1", "3", "4"]);
$("select").val(["2", "value3"]);
```

В качестве примера рассмотрим различные варианты получения значений и текста пункта списка с множественным выбором (листинг 5.3).

### Листинг 5.3. Получение значений списка с множественным выбором

```
<html>
<head>
<title>Получение значений списка с множественным выбором</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
$(document).ready(function() {
 $("#btn1").click(function() { // Способ 1
 var vals = $("#sel1").val() || [];
 var count = vals.length;
 if (count > 0) {
 var elem = $("#div1").empty();
 for (var i=0; i<count; i++) {
 elem.append(vals[i] + " => " +
 $("#sel1 option[value='" + vals[i] + "']").text() +
 "
");
 }
 } else {
 $("#div1").html("Ничего не выделено");
 }
 });
 $("#btn2").click(function() { // Способ 2
 var sels = $("#sel1 option:selected");
 var count = sels.size();
 if (count > 0) {
 var elem = $("#div1").empty();
 for (var i=0; i<count; i++) {
 elem.append(sels.eq(i).val() + " => " +
 sels.eq(i).text() + "
");
 }
 } else {
 $("#div1").html("Ничего не выделено");
 }
 });
 $("#btn3").click(function() { // Способ 3
 var elem = $("#div1").empty();
 $("#sel1 option:selected").each(function() {
 elem.append(this.value + " => " + this.text + "
");
 });
 if (elem.text() == "") {
 elem.html("Ничего не выделено");
 }
 });
});
```

```

 }
 });
});
</script>
</head>
<body>
<select id="sel1" size="5" multiple>
<option value="1">Элемент1</option>
<option value="2">Элемент2</option>
<option value="3">Элемент3</option>
<option value="4">Элемент4</option>
<option value="5">Элемент5</option>
</select>
<div id="div1"></div>
<input type="button" value="Способ 1" id="btn1">

<input type="button" value="Способ 2" id="btn2">

<input type="button" value="Способ 3" id="btn3">
</body>
</html>

```

Итак, в этом примере представлено три способа получения значений. После щелчка на первой кнопке запустится соответствующий обработчик. С помощью строки

```
var vals = $("#sel1").val() || [];
```

мы получаем все значения выделенных пунктов списка. Если ни один пункт не выбран, то присваиваем переменной пустой массив. Далее получаем количество элементов массива и перебираем его с помощью цикла `for`. На каждой итерации цикла находим элемент, значение которого совпадает со значением выбранного пункта списка, и получаем текст этого пункта. У этого способа есть свой недостаток. Если внутри списка существуют пункты с повторяющимися значениями, то можно получить текст другого пункта. Однако повторяющееся значение — редко встречающееся явление, практически всегда значения пунктов являются уникальными.

Следующий способ демонстрирует применение селектора `:selected`, а также возможность перебора элементов коллекции с помощью цикла `for` и метода `eq()`. На каждой итерации мы передаем в метод `eq()` индекс элемента в коллекции и получаем значение и текст пункта.

Наконец, последний способ позволяет полностью обойтись без циклов. Перебор всех элементов осуществляется с помощью метода `each()`. На каждой итерации внутри функции обратного вызова доступна ссылка `(this)` на текущий DOM-элемент. По этой причине мы можем воспользоваться свойствами `value` и `text` для получения значения и текста пункта соответственно.

## 5.10. Вычисление положения элементов

Для вычисления положения элемента предназначены методы `offset()` и `position()`. Методы возвращают объект с двумя свойствами:

- `top` — смещение сверху;
- `left` — смещение слева.

Выведем положение элемента при щелчке на нем (листинг 5.4).

#### **Листинг 5.4. Методы offset() и position()**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Методы offset() и position()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("p").one("click", function() {
 var elem = $(this);
 var obj1 = elem.offset();
 var obj2 = elem.position();
 var msg = "Offset top: " + obj1.top + " left: " + obj1.left;
 msg += " Position top: " + obj2.top + " left: " + obj2.left;
 elem.html(msg);
 });
})
//-->
</script>
</head>
<body>
<p>Абзац 1</p>
<div style="margin:20px;padding:15px;border: black 2px dotted;">
<p style="margin:10px;">Абзац 2</p>
</div>
Щелкните на абзаце
</body>
</html>
```

Для получения или установки значения отступа прокрутки предназначены два метода.

- scrollTop([<Значение>]) — значение отступа прокрутки сверху. Если параметр не указан, то метод возвращает числовое значение, а если указан, то задает новое значение для всех элементов коллекции.
- scrollLeft([<Значение>]) — значение отступа прокрутки слева. Если параметр не указан, то метод возвращает числовое значение, а если указан, то задает новое значение для всех элементов коллекции.

Выведем значение отступа прокрутки после щелчка на элементе (листинг 5.5).

#### **Листинг 5.5. Методы scrollTop() и scrollLeft()**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Методы scrollTop() и scrollLeft()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
```

```
<!--
$(document).ready(function() {
 $("#div1").click(function() {
 var elem = $("body");
 $(this)
 .html("top: " + elem.scrollTop() + " left: " +
 elem.scrollLeft());
 elem.scrollTop(0).scrollLeft(0);
 });
//-->
</script>
<style type="text/css">
#div1 {
 margin:20px;padding:15px;
 width:1500px; height:1500px;
 border: black 2px dotted;
 background-color:silver;
}
</style>
</head>
<body>
<div id="div1">Переместите полосы прокрутки и щелкните на сером фоне
чтобы увидеть значения

После щелчка значения будут приравнены к 0</div>
</body>
</html>
```

## Глава 6

# Обработка событий

При взаимодействии пользователя с веб-страницей происходят события. События — это своего рода извещения системы о том, что пользователь выполнил какое-либо действие или внутри самой системы возникло некоторое условие. События возникают в результате щелчка на элементе, перемещения мыши, нажатия клавиши клавиатуры, изменения размеров окна, окончания загрузки веб-страницы и т.д. Зная, какие события может генерировать тот или иной элемент веб-страницы, можно написать функцию для обработки этого события. Например, при отправке данных формы возникает событие `onsubmit`. При наступлении этого события можно проверить данные, введенные пользователем, и, если они не соответствуют ожидаемым, прервать отправку данных.

## 6.1. События документа

В самом начале книги мы уже рассматривали возможность выполнения скриптов сразу после формирования структуры документа, не дожидаясь загрузки других элементов. Как вы уже знаете, обработать это событие можно с помощью метода `ready()`.

```
$(document).ready(function() {
 alert("Документ доступен для выполнения скриптов");
});
```

Если необходимо выполнить какие-либо действия только после полной загрузки самого HTML-документа, а также всех других элементов (например, изображений), то можно воспользоваться методом `load()`. Синтаксис метода таков.

```
load(<Функция обратного вызова>)
```

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>]) {
 // ...
}
```

Выведем подтверждающее сообщение и тип события после полной загрузки веб-страницы.

```
$(window).load(function(e) {
 alert("Документ полностью загружен");
 alert("Тип события " + e.type);
});
```

В результате выполнения получим два сообщения.

Документ полностью загружен

Тип события load

Для выполнения каких-либо действий непосредственно перед выгрузкой документа предназначен метод `unload()`. Метод имеет следующий синтаксис.

```
unload(<Функция обратного вызова>)
```

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>]) {
 // ...
}
```

Выведем сообщение перед выгрузкой документа.

```
$(window).unload(function() {
 alert("Заходите еще!");
});
```

В этом разделе рассмотрим еще два полезных метода.

- `resize(<Функция обратного вызова>)` — выполняется при изменении размера окна.
- `scroll(<Функция обратного вызова>)` — выполняется при прокручивании содержимого элемента страницы, документа, окна или фрейма.

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>]) {
 // ...
}
```

Обработать прокрутку содержимого окна и вывести сообщение можно так.

```
$(window).scroll(function() {
 alert("Прокрутка содержимого окна");
});
```

В качестве примера использования метода `resize()` получим ширину и высоту окна после изменения его размера (листинг 6.1).

### Листинг 6.1. Вывод ширины и высоты окна при изменении размера

```
<html>
<head>
<title>Вывод ширины и высоты окна при изменении размера</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
,<!--
$(document).ready(function() {
 $(window).resize(function() {
 var elem = $(window);
 $("#div1").html("Ширина: " + elem.width() + "
")
 .append("Высота: " + elem.height());
 });
//-->
</script>
```

```
</head>
<body><div id="div1"></div>
</body>
</html>
```

При изменении размера окна в элементе с идентификатором `div1` будет выведена ширина и высота окна.

Метод `error(<функция обратного вызова>)` вызывается при наличии ошибки на странице или в коде JavaScript. В параметре `<функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции> ([<Сообщение> [, <URL> [, <Номер строки>]]]) {
 // ...
}
```

Если ошибка произошла в объекте `window`, то в функции обратного вызова будут доступны три параметра:

- `<Сообщение>` — текст, описывающий ошибку;
- `<URL>` — полный URL-адрес документа с ошибкой;
- `<Номер строки>` — номер строки с ошибкой.

Чтобы скрыть ошибки JavaScript от пользователей, необходимо внутри функции обратного вызова вернуть значение `true`.

```
$(window).error(function() {
 return true;
});
```

С помощью метода `error()` можно обработать ошибку загрузки изображения.

```
$("#img").error(function() {
 $(this).attr("src", "noimage.gif");
});
```

#### Обратите внимание

Для корректной работы этого примера необходимо разместить скрипт на сервере. В противном случае обработки события не будет.

## 6.2. События мыши

Для обработки событий мыши в jQuery предназначены следующие методы.

- `click([<функция обратного вызова>])` — выполняется при щелчке на элементе или веб-странице. Если параметр не указан, то это позволит имитировать щелчок на элементе. Событию `onclick` предшествуют два события: `onmousedown` и `onmouseup`. Выведем сообщение после щелчка на любой кнопке.

```
$(".button").click(function() {
 alert("Вы нажали на кнопку");
});
```

- `toggle()` — позволяет чередовать несколько функций при щелчке на элементе или веб-странице. Функции вызываются по очереди. При первом щелчке вызыва-

ется первая функция, при втором — вторая, при третьем — третья. Если третья функция не определена, то вызывается опять первая функция, и т.д. Метод `toggle()` имеет следующий формат.

```
toggle(<Функция обратного вызова 1>,
 <Функция обратного вызова 2>[, ...
 ...,
 <Функция обратного вызова N>])
```

Сделаем цвет текста абзаца меняющимся после каждого щелчка мыши.

```
$("p").toggle(
 function() {
 $(this).css("color", "red");
 },
 function() {
 $(this).css("color", "blue");
 },
 function() {
 $(this).css("color", "green");
 }
).click();
```

Попробуйте щелкнуть несколько раз на абзаце. Цвет текста будет последовательно меняться с красного на синий, с синего — на зеленый, с зеленого — опять на красный. Обратите внимание на последнюю строку примера. После назначения обработчика события вызывается событие `onclick` с помощью метода `click()` без параметров. По этой причине с самого начала цвет абзаца будет красным.

- `dblclick([<Функция обратного вызова>])` — выполняется при двойном щелчке на элементе или веб-странице. Если параметр не указан, то это позволит имитировать двойной щелчок на элементе. Событию `ondblclick` предшествуют три события: `onmousedown`, `onmouseup` и `onclick`. Выведем сообщение при двойном щелчке на любом абзаце.

```
$("p").dblclick(function() {
 alert("Вы сделали двойной щелчок");
});
```

- `mousedown(<Функция обратного вызова>)` — выполняется при нажатии кнопки мыши на элементе или странице. Событие `onmousedown` возникает перед событиями `onmouseup` и `onclick`.
- `mouseup(<Функция обратного вызова>)` — выполняется при отпускании ранее нажатой кнопки мыши. Событие `onmouseup` возникает между событиями `onmousedown` и `onclick`. Выведем сообщения при нажатии и отпусканнии кнопки мыши над абзацем.

```
$("p").mousedown(function() {
 $("#div1").append("Событие onmousedown" + "
");
}).mouseup(function() {
 $("#div1").append("Событие onmouseup" + "
");
});
```

```
<p>Щелкни меня</p>
<div id="div1"></div>
```

- `mousemove(<Функция обратного вызова>)` — выполняется при любом перемещении мыши. Выведем сообщение при перемещении курсора над абзацем.
- ```
$("p").mousemove(function() {
    $("#div1").append("Событие onmousemove" + "<br>");
});
```
- `mouseover(<Функция обратного вызова>)` — выполняется при наведении курсора мыши на элемент.
 - `mouseout(<Функция обратного вызова>)` — выполняется при выходе курсора мыши за пределы элемента. При наведении курсора на ссылку сделаем цвет ссылки красным и выведем сообщение, а при уходе курсора со ссылки — изменим ее цвет на черный и выведем сообщение.
- ```
$("a").mouseover(function() {
 $(this).css("color", "red");
 $("#div1").append("Событие onmouseover" + "
");
}).mouseout(function() {
 $(this).css("color", "black");
 $("#div1").append("Событие onmouseout" + "
");
});
```
- `hover()` — позволяет обработать наведение и уход курсора с элемента с помощью одного метода. Имеет следующий формат.
- ```
hover(<Функция обратного вызова для события onmouseover>,
      <Функция обратного вызова для события onmouseout>)
```

Переделаем наш предыдущий пример и используем метод `hover()`.

```
$("a").hover(function() {
    $(this).css("color", "red");
    $("#div1").append("Событие onmouseover" + "<br>");
}, function() {
    $(this).css("color", "black");
    $("#div1").append("Событие onmouseout" + "<br>");
});
```

Использование метода `hover()` имеет еще одно преимущество. Если внутри блока существует другой блок, то отдельные функции будут вызываться при входе во внутренний блок, а также при выходе из него. При использовании метода `hover()` эти перемещения будут игнорироваться (листинг 6.2).

Листинг 6.2. Особенности использования метода `hover()`

```
<html>
<head>
<title>Особенности использования метода hover()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#div1").mouseover(function() {
        $("#log1").append("Событие onmouseover<br>");
    });
});
```

```

$( "#div1" ).mouseout(function() {
    $("#log1").append("Событие onmouseout<br>") ;
});
$("#div2" ).hover(
    function() {
        $("#log2").append("Событие onmouseover<br>") ;
    },
    function() {
        $("#log2").append("Событие onmouseout<br>") ;
    }
);
//-->
</script>
<style type="text/css">
.cls1 {
    border:2px solid black; background-color:green; padding:30px;
}
.cls2 {
    border:2px solid black; background-color:silver;
}
</style>
</head>
<body>
<b>Отдельные обработчики:</b><br>
<div class="cls1" id="div1">
<div class="cls2">
Строка 1<br>
Строка 2
</div>
</div>
<div id="log1"></div>
<b>Функция hover():</b><br>
<div class="cls1" id="div2">
<div class="cls2">
Строка 1<br>
Строка 2
</div>
</div>
<div id="log2"></div>
</body>
</html>

```

Для первого вложенного блока мы определили отдельные обработчики событий. Если переместить курсор мыши через весь блок посередине, получим следующую последовательность событий.

Событие onmouseover
Событие onmouseout
Событие onmouseover
Событие onmouseout
Событие onmouseover
Событие onmouseout

Для второго вложенного блока обработчики событий назначены с помощью метода `hover()`. При том же самом действии мы получим только два события.

Событие `onmouseover`

Событие `onmouseout`

Во всех приведенных в этом разделе методах мы еще не разобрались с параметром, который можно им передать. В качестве параметра `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>]) {  
    // ...  
}
```

Элемент, вызвавший событие, доступен внутри функции через указатель `this`. Обратите внимание на то, что указатель `this` ссылается на элемент объектной модели документа, а не на элемент коллекции jQuery. Если в параметре `<Объект event>` указана переменная, то через нее можно обратиться к свойствам объекта `event`. Напишем обработчик щелчка мыши для двух элементов сразу. При нажатии выведем название тега в элементе с идентификатором `div1`.

```
$( "p, div" ).click(function(e) {  
    $("#div1")  
        .append("Элемент " + e.srcElement.tagName + "<br>");  
});
```

6.3. События клавиатуры

Для обработки событий клавиатуры в jQuery предназначены следующие методы.

- `keydown(<Функция обратного вызова>)` — выполняется при нажатии клавиши клавиатуры.
- `keypress(<Функция обратного вызова>)` — выполняется при нажатии клавиши клавиатуры. Аналогичен методу `keydown()`, но возвращает значение кода символа в кодировке Unicode. Событие генерируется постоянно, пока пользователь не отпустит клавишу.
- `keyup(<Функция обратного вызова>)` — выполняется при отпускании ранее нажатой клавиши клавиатуры.

В качестве параметра `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>]) {  
    // ...  
}
```

Если в параметре `<Объект event>` указана переменная, то через нее можно обратиться к свойствам объекта `event`. Например, чтобы получить код нажатой клавиши, можно воспользоваться свойством `keyCode`. В качестве примера получим код нажатой клавиши и продемонстрируем последовательность прохождения событий (листинг 6.3).

Листинг 6.3. События клавиатуры

```
<html>
<head>
<title>События клавиатуры</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $(":text").keyup(function(e){
        $("#div1").append("keyup " + e.keyCode + "<br>");
    });
    $(":text").keydown(function(e){
        $("#div1").append("keydown " + e.keyCode + "<br>");
    });
    $(":text").keypress(function(e){
        $("#div1").append("keypress " + e.keyCode + "<br>");
    });
});
//-->
</script>
</head>
<body>
<input type="text">
<div id="div1"></div>
</body>
</html>
```

Выделите текстовое поле и нажмите какую-либо клавишу на клавиатуре. Например, нажмем клавишу <A> при русской раскладке клавиатуры. В итоге получим следующее сообщение.

```
keydown 70
keypress 1072
keyup 70
```

6.4. События формы

Для обработки событий формы предназначены такие методы.

- `focus([<Функция обратного вызова>])` — выполняется при получении фокуса элементом формы. Если параметр не указан, то элемент получит фокус ввода.
- `blur([<Функция обратного вызова>])` — выполняется при потере фокуса элементом формы. Если параметр не указан, то элемент потеряет фокус ввода.
- `change(<Функция обратного вызова>)` — выполняется при изменении данных в текстовом поле и перемещении фокуса на другой элемент формы либо при отправке данных формы.
- `submit([<Функция обратного вызова>])` — выполняется при отправке данных формы. Если параметр не указан, то форма будет отправлена.

- `select([<Функция обратного вызова>])` — выполняется при выделении содержимого элемента. Если параметр не указан, то содержимое элемента будет полностью выделено.

В качестве параметра `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции> ([<Объект event>]) {
    // ...
}
```

Элемент, вызвавший событие, доступен внутри функции через указатель `this`. Обратите внимание на то, что указатель `this` ссылается на элемент объектной модели документа, а не на элемент коллекции jQuery. Если в параметре `<Объект event>` указана переменная, то через нее можно обратиться к свойствам объекта `event`.

Продемонстрируем обработку различных событий формы (листинг 6.4).

Листинг 6.4. События формы

```
<html>
<head>
<title>События формы</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $(":text").focus(function() {
        $(this).css("background-color", "#ffffbef");
    }).blur(function() {
        $(this).css("background-color", "#ffffff");
    });
    $("#sel1").change(function(){
        alert("Значение выбранного пункта " + $(this).val());
    });
    $("form").submit(function(){
        if (window.confirm("Отправить данные формы?")) {
            return true;
        }
        else return false;
    });
    ":text").select(function(){
        alert("Выделен фрагмент");
    });
    $("#btn1").click(function(){
        $("#txt1").select();
    });
    $("#btn2").click(function(){
        $("#txt1").focus();
    });
    $("#btn3").click(function(){
        $("#txt1").blur();
    });
    $("#btn4").click(function(){
        $("#frm").submit();
    });
});
```

```

});  

//-->  

</script>  

</head>  

<body>  

<form id="frm">  

<select id="sel1">  

<option value="1">Пункт 1</option>  

<option value="2">Пункт 2</option>  

<option value="3">Пункт 3</option>  

<option value="4">Пункт 4</option>  

</select>  

<input type="text" id="txt1">  

<input type="submit" value="Отправить">  

</form>  

<input type="button" value="Выделить поле" id="btn1">  

<input type="button" value="Установить фокус на поле" id="btn2">  

<input type="button" value="Снять фокус с поля" id="btn3">  

<input type="button" value="Отправить форму" id="btn4">  

</body>  

</html>

```

Итак, мы создали форму с тремя элементами. При выборе пункта из списка будет отображено сообщение со значением выбранного пункта. Если выделить текстовое поле, то изменится цвет фона, а если убрать фокус с поля, то цвет фона опять станет белым. Щелчок на кнопке Отправить вызывает отображение диалогового окна, с помощью которого можно прервать отправку данных формы. Под формой расположены четыре кнопки. Первая кнопка позволяет выделить содержимое текстового поля. После выделения будет отображено сообщение о выделении фрагмента, так как мы определили соответствующий обработчик. Такое же сообщение можно получить, если с помощью мыши выделить фрагмент в поле. Вторая кнопка позволяет установить фокус ввода на текстовое поле, а третья — его снять. И наконец, четвертая кнопка предназначена для отправки данных формы. Щелчок на этой кнопке приводит к выводу диалогового окна для подтверждения отправки данных.

6.5. Универсальные обработчики событий

В предыдущих разделах мы рассмотрели обработчики конкретных событий. Библиотека jQuery позволяет также назначить обработчики для разных событий (в том числе и собственных событий) одним методом. Для назначения, вызова и удаления обработчиков предназначены следующие методы.

- `bind(<Тип события> [, <Данные>], <Функция обратного вызова>)` — позволяет назначить обработчик для всех элементов коллекции. В параметре `<Тип события>` могут быть указаны следующие значения:
 - ◊ события документа: `load`, `unload`, `resize`, `scroll`, `error`;
 - ◊ события мыши: `click`, `dblclick`, `mousedown`, `mouseup`, `mousemove`, `mouseover`, `mouseout`, `mouseenter`, `mouseleave`;
 - ◊ события клавиатуры: `keydown`, `keypress`, `keyup`;
 - ◊ события формы: `focus`, `blur`, `change`, `select`, `submit`.

Почти все эти события мы уже рассматривали в предыдущих разделах. Например, назначить обработчик события click для всех кнопок можно так.

```
$(".button").bind("click", function() {
    alert("Вы нажали на кнопку");
});
```

Нерассмотренными остались два события мыши. Событие mouseenter выполняется при наведении курсора мыши на элемент, а событие mouseleave — при уходе курсора мыши за пределы элемента. Использование этих событий отличается от обработки событий mouseover и mouseout. Если внутри блока существует другой блок, то событие mouseover будет вызываться при входе во внутренний блок, а событие mouseout — при выходе из него. При использовании событий mouseenter и mouseleave эти перемещения будут игнорироваться (листинг 6.5).

Листинг 6.5. События mouseenter и mouseleave

```
<html>
<head>
<title>События mouseenter и mouseleave</title>
<meta http-equiv="Content-Type"
      content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#div1").bind("mouseover", function() {
        $("#log1").append("Событие mouseover<br>");
    });
    $("#div1").bind("mouseout", function() {
        $("#log1").append("Событие mouseout<br>");
    });
    $("#div2").bind("mouseenter", function() {
        $("#log2").append("Событие mouseenter<br>");
    });
    $("#div2").bind("mouseleave", function() {
        $("#log2").append("Событие mouseleave<br>");
    });
});
//--
</script>
<style type="text/css">
.cls1 {
    border:2px solid black; background-color:green;
    padding:30px;
}
.cls2 {
    border:2px solid black; background-color:silver;
}
</style>
</head>
<body>
<b>События mouseover и mouseout:</b><br>
```

```
<div class="cls1" id="div1">
<div class="cls2">
Строка 1<br>
Строка 2
</div>
</div>
<div id="log1"></div>
<b>События mouseenter и mouseleave:</b><br>
<div class="cls1" id="div2">
<div class="cls2">
Строка 1<br>
Строка 2
</div>
</div>
<div id="log2"></div>
</body>
</html>
```

Для первого вложенного блока мы определили обработчики событий `mouseover` и `mouseout`. Если переместить курсор мыши через весь блок посередине, то получим следующую последовательность событий.

Событие mouseover
Событие mouseout
Событие mouseover
Событие mouseout
Событие mouseover
Событие mouseout

Для второго вложенного блока назначены обработчики событий `mouseenter` и `mouseleave`. При том же действии получим только два события.

Событие mouseenter
Событие mouseleave

В параметре `<Тип события>` может быть указана комбинация событий через пробел. Изменим класс ссылки при наведении курсора мыши, а при уходе курсора удалим класс.

```
$( "a" ).bind("mouseenter mouseleave", function() {
    $(this).toggleClass("cls1");
});
```

Кроме того, с помощью метода `bind()` можно назначать собственные события. Вызов собственных событий осуществляется при помощи методов `trigger()` и `triggerHandler()`. Создадим собственное событие, а затем вызовем его.

```
$( "#div1" ).bind("myEvent", function() {
    alert("Обработка собственного события");
});
$( "#div1" ).trigger("myEvent"); // Вызов события
```

В качестве параметра `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>[, <Аргументы>]]) {
    // ...
}
```

Элемент, вызвавший событие, доступен внутри функции через указатель `this`. Обратите внимание на то, что указатель `this` ссылается на элемент объектной модели документа, а не на элемент коллекции jQuery. Если в параметре <Объект event> указана переменная, то через нее можно обратиться к свойствам объекта `event`. В параметре <Аргументы> можно указать несколько переменных через запятую. Значения в эти переменные можно передать с помощью второго параметра методов `trigger()` и `triggerHandler()`.

```
$("#div1").bind("myEvent", function(e, msg) {
    alert("Обработка собственного события. " + msg);
});
$("#div1").trigger("myEvent", ["Наше сообщение"]);
```

С помощью необязательного параметра <Данные> можно передать значения в функцию-обработчик. Для этого передаваемые параметры и значения должны быть указаны следующим образом.

```
{
Параметр1: "Значение1",
Параметр2: "Значение2",
...
ПараметрN: "ЗначениеN"
}
```

Получить значение переданного параметра внутри функции можно так.

```
<Значение> = <Объект event>.data.<Параметр>
```

Рассмотрим пример.

```
function f_click(e) {
    var str = "msg1 = " + e.data.msg1 + "\n";
    str += "msg2 = " + e.data.msg2;
    alert(str);
    return false;
}
$("a").bind(
    "click", // Событие
    {msg1: "Сообщение 1", msg2: "Сообщение 2"}, // Наши данные
    f_click // Ссылка на функцию
);
```

При щелчке на ссылке будет выведено сообщение со значениями переданных данных.

```
msg1 = Сообщение 1
msg2 = Сообщение 2
```

Для одного события можно назначить несколько обработчиков. В этом случае они будут вызываться в той последовательности, в которой были определены.

```
$("#btn1").bind("click", function() {
    alert("Первый обработчик");
});
$("#btn1").bind("click", function() {
    alert("Второй обработчик");
});
```

При щелчке на элементе с идентификатором `btn1` будут последовательно выведены два сообщения.

Первый обработчик
Второй обработчик

- `one(<Тип события>[, <Данные>], <Функция обратного вызова>)` — позволяет назначить обработчик для всех элементов коллекции. Метод идентичен `bind()`, но событие срабатывает только один раз, после чего обработчик будет автоматически удален.

```
$(":button").one("click", function() {
    alert("Вы щелкнули на кнопке");
});
```

После первого щелчка на кнопке будет выведено указанное сообщение, а после второго обработчик вызван уже не будет.

- `trigger(<Тип события>[, <Массив данных>])` — вызывает обработчик указанного события.

```
$(document).bind("click", function() { // Обработчик
    alert("Событие onclick");
});
$(document).trigger("click"); // Вызываем обработчик
```

Если указан массив данных, то он будет передан в функцию обратного вызова в обработчике. Каждый параметр будет соответствовать одному элементу массива.

```
$("#div1").bind("myEvent", function(e, msg1, msg2) {
    $("#div2").append("Событие myEvent. " + msg1 + " " + msg2)
        .append("<br>");
});
$("p").click(function() {
    $("#div1").trigger("myEvent", ['Сообщение1',
    'Сообщение2']);
});
```

<p>Нажмите здесь</p>

В результате выполнения этого кода получим сообщение.

Событие myEvent Сообщение1 Сообщение2

- `triggerHandler(<Тип события>[, <Массив данных>])` — вызывает обработчик указанного события. В отличие от метода `trigger()`, событие вызывается только для первого элемента коллекции. Кроме того, при использовании метода `triggerHandler()` не происходит действие по умолчанию.

```
 $("form").bind("submit", function(e, msg) {
    if (msg) {
        alert("Форма отправлена не будет. " + msg);
    }
    else {
        alert("Форма будет отправлена");
    }
});
```

70

Глава 6

```

});  

$(":button").bind("click", function() {  

    $("form").triggerHandler("submit", ["Наши данные"]);  

});  
  

<form action="file.php"><input type="text">  

<input type="submit" value="Отправить">  

</form>  

<input type="button" value="Вызвать обработчик">

```

В данном примере щелчок на кнопке **Вызвать обработчик** приводит к запуску обработчика события `onclick`. Внутри обработчика мы вызываем событие `onsubmit` и передаем свои данные. На основании этих данных мы можем определить, вызвано ли событие при щелчке на кнопке **Отправить** или оно вызвано нами искусственно. Если переменная `msg` определена, то вызов события был произведен с помощью метода `triggerHandler()`. В этом случае после вывода сообщения данные формы отправлены не будут. Если щелкнуть на кнопке **Отправить**, то переменная `msg` будет не определена, и после вывода сообщения данные формы будут отправлены.

- `unbind([<Тип события>[, <Название функции>]])` — удаляет обработчики событий для всех элементов коллекции.

```

$("#btn1").bind("click", function() {  

    alert("Вы нажали на кнопку");  

});  

$("#btn2").bind("click", function() {  

    $("#btn1").unbind("click");  

});  
  

<input type="button" value="Нажми меня" id="btn1">  

<input type="button" value="Удалить обработчик" id="btn2">

```

Если во втором параметре указать название функции, то будет удален только обработчик с таким названием.

```

function f_click1() {  

    alert("Функция f_click1()");  

}  

function f_click2() {  

    alert("Функция f_click2()");  

}  
  

$("#btn1").bind("click", f_click1);  

$("#btn1").bind("click", f_click2);  

$("#btn2").bind("click", function() {  

    $("#btn1").unbind("click", f_click2);  

});  
  

<input type="button" value="Нажми меня" id="btn1">  

<input type="button" value="Удалить обработчик" id="btn2">

```

В этом примере мы назначили два обработчика события `click` для первой кнопки. Если щелкнуть на кнопке **Нажми меня**, то будут выведены два сообщения.

Функция f_click1()
Функция f_click2()

После щелчка на кнопке Удалить обработчик обработчик события с названием f_click2 будет удален. Теперь после щелчка на первой кнопке мы получим только одно сообщение.

Функция f_click1()

Если параметры не указаны, то будут удалены все обработчики событий.

```
$("#btn1").unbind();
```

6.6. Методы live() и die()

Назначение обработчиков событий с помощью метода bind() имеет один существенный недостаток. Если мы назначили обработчик, а затем добавили новые элементы, то для этих элементов обработчики событий назначены не будут.

```
$(".button").bind("click", function() {
    var html = '<input type="button" value="Новая кнопка">';
    $(this).after("<br>" + html);
});
```

```
<input type="button" value="Кнопка">
```

В этом примере мы назначили обработчик события click для всех кнопок. После щелчка на кнопке добавляется новая кнопка. Если попробовать на ней щелкнуть, то никаких изменений не произойдет.

Для динамического назначения обработчиков событий предназначен метод live(). Переделаем наш предыдущий пример и используем метод live() вместо bind().

```
$(".button").live("click", function() {
    var html = '<input type="button" value="Новая кнопка">';
    $(this).after("<br>" + html);
});
```

В этом случае, если щелкнуть на новой кнопке, появится еще одна кнопка. Щелчок на этой кнопке приведет к созданию еще одной кнопки и т.д.

Обратите внимание

Метод live() доступен начиная с версии jQuery 1.3. В более ранних версиях можно использовать модуль Live Query.

Метод live() имеет следующий формат.

```
live(<Тип события>, <Функция обратного вызова>)
```

В параметре <Тип события> могут быть указаны следующие значения:

- события мыши: click, dblclick, mousedown, mouseup, mousemove, mouseover, mouseout;
- события клавиатуры: keydown, keypress, keyup.

Кроме того, с помощью метода live() можно назначать собственные события. Вызов собственных событий осуществляется при помощи методов trigger() и triggerHandler(). Создадим собственное событие, а затем вызовем его.

```
$("#div1").live("myEvent", function() {
    alert("Обработка собственного события");
});
$("button").click(function() {
    $("#div1").trigger("myEvent"); // Вызов события
});
```

В качестве параметра *<Функция обратного вызова>* указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>[, <Аргументы>]]) {
    // ...
}
```

Элемент, вызвавший событие, доступен внутри функции через указатель `this`. Обратите внимание на то, что указатель `this` ссылается на элемент *объектной модели документа*, а не на элемент коллекции jQuery. Если в параметре *<Объект event>* указана переменная, то через нее можно обратиться к свойствам объекта `event`. В параметре *<Аргументы>* можно указать несколько переменных через запятую. Значения в эти переменные можно передать с помощью второго параметра методов `trigger()` и `triggerHandler()`.

```
$("#div1").bind("myEvent", function(e, msg) {
    alert("Обработка собственного события. " + msg);
});
$("button").click(function() {
    $("#div1").trigger("myEvent", ["Наше сообщение"]);
});
```

Удалить обработчик события можно с помощью метода `die()`. Формат метода:
`die(<Тип события>[, <Название функции>])`

Обратите внимание

Метод `die()` доступен начиная с версии jQuery 1.3.

Рассмотрим пример.

```
$("#btn1").live("click", function() {
    alert("Вы нажали на кнопку");
});
$("#btn2").click(function() {
    $("#btn1").die("click"); // Удаляем обработчик
});
```



```
<input type="button" value="Нажми меня" id="btn1">
<input type="button" value="Удалить обработчик" id="btn2">
```

Если во втором параметре указать название функции, то будет удален только обработчик с таким названием.

```
function f_click1() {
    alert("Функция f_click1()");
}
function f_click2() {
    alert("Функция f_click2()");
}
```

```

$( "#btn1" ).live("click", f_click1);
$( "#btn1" ).live("click", f_click2);
$( "#btn2" ).click(function() {
    $( "#btn1" ).die("click", f_click2);
}); ~~~~~

<input type="button" value="Нажми меня" id="btn1">
<input type="button" value="Удалить обработчик" id="btn2">

```

В этом примере мы назначили два обработчика события click для первой кнопки. Если щелкнуть на кнопке **Нажми меня**, то будут выведены два сообщения.

Функция f_click1()

Функция f_click2()

В результате щелчка на кнопке **Удалить обработчик** обработчик события с названием f_click2 будет удален. Теперь после щелчка на первой кнопке получим только одно сообщение.

Функция f_click1()

6.7. Всплытие событий

Что же такое “всплытие” событий? Давайте рассмотрим следующий пример (листинг 6.6).

Листинг 6.6. Всплытие событий

```

<html>
<head>
<title>Всплытие событий</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("*").click(function(e) {
        $("#div1")
            .append("Элемент " + $(this)[0].tagName + "<br>");
    });
//-->
</script>
</head>
<body>
<p>Щелкните мышью <span style="color:red">здесь</span></p>
<div id="div1"></div>
</body>
</html>

```

В этом примере мы написали обработчик события onclick для всех элементов страницы. Попробуем щелкнуть левой кнопкой мыши на слове **здесь**. В итоге, вместо одного события, получим целую последовательность событий.

Элемент SPAN
Элемент P
Элемент BODY
Элемент HTML

Иными словами, событие onclick последовательно передается очередному элементу-родителю. Для тега элементом-родителем является абзац. Для абзаца элементом-родителем является само тело документа, а для тела документа элементом-родителем является тег <html>. Такое прохождение событий и называется *всплытием событий*.

Иногда всплытие событий необходимо прервать. Для этого внутри функции обратного вызова следует вернуть значение false. Продемонстрируем это на примере (листинг 6.7).

Листинг 6.7. Прерывание всплытия событий

```
<html>
<head>
<title>Прерывание всплытия событий</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("*").click(function(e) {
        $("#div1")
            .append("Элемент " + $(this)[0].tagName + "<br>");
        return false;
    });
//-->
</script>
</head>
<body>
<p>Щелкните мышью <span style="color:red">здесь</span></p>
<div id="div1"></div>
</body>
</html>
```

Попробуем теперь щелкнуть левой кнопкой мыши на слове здесь. В итоге, вместо четырех событий, получим только одно.

Элемент SPAN

Кроме возврата значения false, для прерывания всплытия событий можно воспользоваться методом stopPropagation() объекта event (листинг 6.8).

Листинг 6.8. Прерывание всплытия с помощью метода stopPropagation()

```
<html>
<head>
<title>Прерывание всплытия с помощью метода stopPropagation()
</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
```

```

$(document).ready(function() {
    $("*").click(function(e) {
        $("#div1")
            .append("Элемент " + $(this)[0].tagName + "<br>");

        if (e.stopPropagation) e.stopPropagation();
        else e.cancelBubble = true;
    });
}); //-->
</script>
</head>
<body>
<p>Щелкните мышью <span style="color:red">здесь</span></p>
<div id="div1"></div>
</body>
</html>

```

6.8. Действия по умолчанию и их отмена

Для многих событий назначены действия по умолчанию, т.е. действия, которые веб-браузер выполняет в ответ на возникшие в документе события. Например, при щелчке на гиперссылке действием по умолчанию будет переход по указанному URL-адресу, щелчок на кнопке **Submit** приводит к отправке данных формы и т.д.

Иногда действия по умолчанию необходимо прервать. Например, при отправке данных формы можно проверить их на соответствие ожидаемым и, если они не соответствуют установленным требованиям, прервать отправку. Для этого внутри функции обработчика необходимо вернуть значение `false`.

В листинге 6.9 приведен пример проверки правильности ввода почтового адреса в поле `E-mail` и прерывания перехода по гиперссылке при обнаружении ошибки.

Листинг 6.9. Прерывание действий по умолчанию

```

<html>
<head>
<title>Прерывание действий по умолчанию</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#frm").submit(function() {
        var p = /^[a-z0-9_\.\-]+@[a-z0-9\-\-]+\.\+[a-z]{2,4}$/i;
        if (p.test($("#email").val())) {
            if (window.confirm("Отправить данные формы?")) {
                return true;
            }
            else return false;
        }
        else {
            alert("E-mail введен не правильно");
            return false;
        }
    });
});

```

```

    });
    $("a").click(function() {
        alert("Перехода по ссылке не будет");
        return false;
    });
    $("#email").focus(function() {
        $(this).css("background-color", "#ffffbef");
    }).blur(function() {
        $(this).css("background-color", "#ffffff");
    });
});
//-->
</script>
</head>
<body>
<form action="file.php" method="GET" id="frm">
E-mail:<br>
<input type="text" name="email" id="email"><br>
<input type="submit" value="Отправить">
</form>
<a href="file.html">Нажмите для перехода по ссылке</a><br><br>
</body>
</html>

```

Кроме возврата значения `false`, для отмены действий по умолчанию можно воспользоваться методом `preventDefault()` объекта `event` (листинг 6.10).

Листинг 6.10. Прерывание событий с помощью метода `preventDefault()`

```

<html>
<head>
<title>Прерывание событий с помощью метода preventDefault()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
function f.preventDefault(e) {
    (e.preventDefault) ? e.preventDefault() : e.returnValue = false;
}
$(document).ready(function() {
    $("#frm").submit(function(e) {
        var p = /^[a-z0-9_\.-]+@[a-z0-9\.-]+\.[a-z]{2,4}$/i;
        if (p.test($("#email").val())) {
            if (window.confirm("Отправить данные формы?")) {
                return true;
            }
            else f.preventDefault(e);
        }
        else {
            alert("E-mail введен не правильно");
            f.preventDefault(e);
        }
    });
    $("a").click(function(e) {

```

```
    alert("Перехода по ссылке не будет");
    f.preventDefault(e);
});
$("#email").focus(function() {
    $(this).css("background-color", "#ffffbef");
}).blur(function() {
    $(this).css("background-color", "#ffffff");
});
});
//-->
</script>
</head>
<body>
<form action="file.php" method="GET" id="frm">
E-mail:<br>
<input type="text" name="email" id="email"><br>
<input type="submit" value="Отправить">
</form>
<a href="file.html">Нажмите для перехода по ссылке</a><br><br>
</body>
</html>
```

Глава 7

Эффекты и анимация

Базовый набор методов библиотеки jQuery позволяет плавно отображать и скрывать элементы, управлять прозрачностью, а также создавать произвольную анимацию за счет изменения различных числовых атрибутов. Множество других эффектов реализуется с помощью дополнительных модулей.

7.1. Управление отображением элемента

Для управления отображением элементов предназначены следующие методы.

- `show([<Продолжительность>[, <Функция обратного вызова>]])` — отображает элемент.
- `hide([<Продолжительность>[, <Функция обратного вызова>]])` — скрывает элемент.
- `toggle([<Продолжительность>[, <Функция обратного вызова>]])` — позволяет чередовать сокрытие и отображение элементов. Если элемент скрыт, то он будет отображен, и наоборот.

```
$(":button").click(function() {
    $("p:first").toggle();
});
```

```
<p>Абзац</p>
<input type="button" value="Скрыть или отобразить абзац">
```

После первого щелчка на кнопке абзац будет скрыт. Если щелкнуть второй раз, то абзац будет отображен.

- `toggle(<Условие>)` — если в качестве параметра указано значение `true`, то элемент будет отображен (с помощью метода `show()`), а если указано значение `false`, то он будет скрыт (с помощью метода `hide()`).
- `slideDown([<Продолжительность>[, <Функция обратного вызова>]])` — показывает элемент, спуская его сверху.
- `slideUp([<Продолжительность>[, <Функция обратного вызова>]])` — скрывает элемент, поднимая его снизу вверх.
- `slideToggle([<Продолжительность>[, <Функция обратного вызова>]])` — позволяет чередовать сокрытие и отображение элементов. Если элемент скрыт, то он будет отображен (спущен сверху), и наоборот (поднят снизу вверх).

```
$("#btn1").click(function() {
    $("div").slideDown(1000);
});
$("#btn2").click(function() {
```

```

        $("div").slideUp(1000);
    });
    $("#btn3").click(function() {
        $("div").slideToggle(1000);
    });

<div style="background-
color:green;height:150px;display:none;">
Скрытый элемент</div>
<input type="button" value="Отобразить элемент" id="btn1">
<input type="button" value="Скрыть элемент" id="btn2">
<input type="button" value="Отобразить или скрыть элемент"
id="btn3">

```

В необязательном параметре *<Продолжительность>* может быть указано время выполнения анимации в миллисекундах или следующие значения:

- *fast* — 200 миллисекунд;
- *normal* — 400 миллисекунд;
- *slow* — 600 миллисекунд.

Отобразим все абзацы.

```
$( "p" ).show();
```

Скроем все абзацы.

```
$( "p" ).hide("slow");
$( "p" ).hide(600);
```

Во втором параметре может быть указана функция, которая будет вызвана после окончания анимации. Указывается ссылка на функцию следующего формата:

```
function <Название функции>() {
    // ...
}
```

Внутри функции обратного вызова доступен указатель (*this*) на текущий DOM-элемент. При щелчке на ссылке вначале скроем ее, а затем по окончании анимации отобразим.

```
$( "a" ).click(function() {
    $(this).hide(600, function() {
        $(this).show("normal");
    });
    return false; // Прерываем переход по ссылке
});
```

В качестве примера сокрытия и отображения элементов создадим меню, в котором можно отобразить не более одного элемента (листинг 7.1).

Листинг 7.1. Сокрытие и отображение элементов

```

<html>
<head>
<title>Сокрытие и отображение элементов</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--

```

```

$(document).ready(function() {
    $("a.menu").click(function() {
        // Находим элемент, расположенный сразу после ссылки
        var elem = $(this).nextAll("div.elem:first");
        if (elem.is(":hidden")) { // Если элемент скрыт
            // Скрываем видимые элементы
            $("div.elem:visible").hide();
            // Отображаем элемент, расположенный сразу после ссылки
            elem.show();
        }
        else { // Если элемент был отображен ранее
            elem.hide(); // Скрываем элемент
        }
        return false; // Прерываем переход по ссылке
    });
});
//-->
</script>
<style type="text/css">
.menu { color:black; }
.elem { display:none; }
</style>
</head>
<body>
<a href="#" class="menu">Отобразить или скрыть элемент 1</a><br>
<div class="elem">Скрытый элемент 1</div>
<a href="#" class="menu">Отобразить или скрыть элемент 2</a><br>
<div class="elem">Скрытый элемент 2</div>
<a href="#" class="menu">Отобразить или скрыть элемент 3</a><br>
<div class="elem">Скрытый элемент 3</div>
<a href="#" class="menu">Отобразить или скрыть элемент 4</a><br>
<div class="elem">Скрытый элемент 4</div>
</body>
</html>

```

Изначально все теги `<div>`, имеющие класс `elem`, скрыты. При щелчке на ссылке, имеющей класс `menu`, находим тег `<div>`, который следует сразу после ссылки. Далее с помощью метода `is()` проверяем, является ли элемент скрытым. Если да, то скрываем все видимые теги `<div>`, а затем отображаем элемент, а если нет, то просто скрываем его. И наконец, прерываем переход по ссылке, возвращая значение `false`.

7.2. Изменение прозрачности элемента

Для изменения прозрачности элементов предназначены следующие методы.

- `fadeIn([<Продолжительность>[, <Функция обратного вызова>]])` — показывает элемент, изменяя его прозрачность.
- `fadeOut([<Продолжительность>[, <Функция обратного вызова>]])` — скрывает элемент, изменяя его прозрачность.
- `fadeTo(<Продолжительность>, <Прозрачность>[, <Функция обратного вызова>])` — позволяет изменить степень прозрачности элемента. Во втором параметре может быть указано вещественное число от 0 до 1 включительно.

Сделаем изображение вначале полупрозрачным. При наведении указателя мыши полностью “проявим” изображение, а как только указатель выйдет за его пределы, установим значение прозрачности равным 0.4.

```
$("#img1").css("opacity", 0.4).hover(  
    function() {  
        $(this).stop().fadeTo(600, 1);  
    },  
    function() {  
        $(this).stop().fadeTo(600, 0.4);  
    }  
);  
  

```

Обратите внимание на метод `stop()`. С его помощью мы досрочно прерываем анимацию. Это позволяет избежать неприятных эффектов при слишком быстром перемещении курсора.

В параметре *<Продолжительность>* указывается время выполнения анимации в миллисекундах или следующие значения:

- `fast` — 200 миллисекунд;
- `normal` — 400 миллисекунд;
- `slow` — 600 миллисекунд.

Рассмотрим пример.

```
$("#btn1").click(function() {  
    $("div").fadeIn(4000);  
});  
$("#btn2").click(function() {  
    $("div").fadeOut(4000);  
});  
  
<div  
style="width:500px; height:150px; background-color:green;  
display:none;">  
Текст</div>  
<input type="button" value="Отобразить" id="btn1">  
<input type="button" value="Скрыть" id="btn2">
```

В параметре *<Функция обратного вызова>* может быть указана функция, которая будет вызвана после окончания анимации. Указывается ссылка на функцию следующего формата.

```
function <Название функции>() {  
    // ...  
}
```

Внутри функции обратного вызова доступен указатель `(this)` на текущий DOM-элемент. В качестве примера при наведении указателя мыши скроем элемент, а после окончания анимации отобразим его.

```
 $("div").mouseover(function() {  
    $(this).fadeOut(200, function() {  
        $(this).fadeIn(600);  
    });
```

```
) ;  
  
<div style="width:100px;height:100px;background-color:black;">  
    </div><br>  
<div style="width:100px;height:100px;background-color:black;"></div>
```

Если для одного элемента указано несколько анимаций, то они будут выполняться последовательно. Иными словами, пока не закончится первая анимация, вторая не начнется. По этой причине предыдущий пример можно переписать короче.

```
$(".div").mouseover(function() {  
    $(this).fadeOut(200).fadeIn(600);  
});
```

7.3. Создание анимации

Практически все методы, которые мы рассматривали в двух предыдущих разделах, используют метод `animate()`. Этот метод позволяет создавать произвольную анимацию за счет изменения числовых атрибутов (например, `height`, `width`, `opacity` и др.).

Примечание

Плавное изменение значений атрибутов, задающих цвет, рассматривается в главе 12, в разделе 12.12.1.

Метод `animate()` имеет два формата.

```
animate(<Изменяемые атрибуты>, <Опции>)  
animate(<Изменяемые атрибуты> [, <Продолжительность> [, <Эффект>  
    [, <Функция обратного вызова>]]])
```

В первом параметре атрибуты и значения должны быть указаны следующим образом.

```
{  
Атрибут1: "Значение1",  
Атрибут2: "Значение2",  
...  
АтрибутN: "ЗначениеN"  
}
```

Если название атрибута содержит дефис, то такое название необходимо указывать в стиле JavaScript (например, `fontSize` вместо `font-size`). В качестве значения могут быть указаны:

- число с абсолютными единицами измерения (например, `px`, `in` и др.);
- число с относительными единицами измерения `%` и `em`;
- `hide`, `show` или `toggle`.

Можно также определять значения атрибута относительно его текущей позиции, указав перед значением `+=` или `-=` (листинг 7.2).

Листинг 7.2. Перемещение элементов

```
<html>  
<head>  
<title>Перемещение элементов</title>
```

```

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#btn1").click(function() {
        $("#div1").animate({ left: "-=" + $("#txt1").val() }, 1000);
    });
    $("#btn2").click(function() {
        $("#div1").animate({ left: "+=" + $("#txt1").val() }, 1000);
    });
    $("#btn3").click(function() {
        $("#div1").animate({ top: "-=" + $("#txt1").val() }, 1000);
    });
    $("#btn4").click(function() {
        $("#div1").animate({ top: "+=" + $("#txt1").val() }, 1000);
    });
});
//-->
</script>
<style type="text/css">
.cls1 {
    position:absolute;
    top:50px;
    left:10px;
    width:100px;
    height:100px;
    background-color:black;
}
</style>
</head>
<body>
<input type="text" value="50px" id="txt1">
<input type="button" value="Влево" id="btn1">
<input type="button" value="Вправо" id="btn2">
<input type="button" value="Вверх" id="btn3">
<input type="button" value="Вниз" id="btn4"><br>
<div id="div1" class="cls1"></div>
</body>
</html>

```

В параметре *<Продолжительность>* может быть указано время выполнения анимации в миллисекундах или следующие значения:

- *fast* — 200 миллисекунд;
- *normal* — 400 миллисекунд;
- *slow* — 600 миллисекунд.

В качестве примера рассмотрим изменение ширины элементов (листинг 7.3).

Листинг 7.3. Изменение ширины элементов

```

<html>
<head>
<title>Изменение ширины элементов</title>

```

```

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#btn1").click(function() {
        $("#div1").animate({ width: "hide" }, 300);
    });
    $("#btn2").click(function() {
        $("#div1").animate({ width: "show" }, 300);
    });
    $("#btn3").click(function() {
        $("#div1").animate({ width: "toggle" }, "slow");
    });
});
//-->
</script>
<style type="text/css">
.cls1 {
    position:absolute;
    top:50px;
    left:10px;
    width:150px;
    height:100px;
    background-color:black;
}
</style>
</head>
<body>
<input type="button" value="Скрыть" id="btn1">
<input type="button" value="Отобразить" id="btn2">
<input type="button" value="Скрыть/Отобразить" id="btn3"><br>
<div id="div1" class="cls1"><div>
</body>
</html>

```

Если щелкнуть на кнопке Скрыть, то элемент свернется справа налево, а если затем щелкнуть на кнопке Отобразить, то он раскроется слева направо. Щелчок на кнопке Скрыть/Отобразить позволяет свернуть элемент (если он развернут) или развернуть (если он свернут).

В параметре <Эффект> могут быть указаны эффекты замедления из дополнительных модулей или два значения:

- swing — в начале идет ускорение, а в конце замедление (значение по умолчанию);
- linear — равномерная скорость движения.

В необязательном параметре <Функция обратного вызова> может быть указана ссылка на функцию, которая будет вызвана после окончания анимации. Внутри функции обратного вызова доступен указатель (*this*) на текущий DOM-элемент. Формат функции:

```

function <Название функции>() {
    // ...
}

```

В параметре *<Опции>* значения должны быть указаны следующим образом.

```
{  
    Опция1: "Значение1",  
    Опция2: "Значение2",  
    ...  
    ОпцияN: "ЗначениеN"  
}
```

Могут быть указаны следующие опции:

- duration — время выполнения анимации в миллисекундах или значения fast, normal и slow;
- easing — эффект замедления из дополнительных модулей или значения swing и linear;
- complete — ссылка на функцию, которая будет вызвана после окончания анимации;
- queue — если указано значение false, то текущая анимация будет выполнена параллельно с последующими анимациями. Если указано значение true (значение по умолчанию), то анимации, заданные для одного и того же элемента, выполняются последовательно (листинг 7.4).

Листинг 7.4. Параллельное и последовательное выполнение анимации

```
<html>  
<head>  
<title>Параллельное и последовательное выполнение анимации</title>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
<script src="jquery.js" type="text/javascript"></script>  
<script type="text/javascript">  
!--  
$(document).ready(function() {  
    $("#btn1").click(function() {  
        $("#div1")  
            .css({ width: "100px", height: "20px", fontSize: "10px" })  
            .animate({ width: "450px", height: "100px" },  
                { queue: false, duration: 2000 })  
            .animate({ fontSize: "32px" }, 2000);  
    });  
    $("#btn2").click(function() {  
        $("#div1")  
            .css({ width: "100px", height: "20px", fontSize: "10px" })  
            .animate({ width: "450px", height: "100px" },  
                { duration: 2000 })  
            .animate({ fontSize: "32px" }, 2000);  
    });  
});  
//-->  
</script>  
<style type="text/css">  
.cls1 {  
    position: absolute;  
    top: 50px;
```

```

    left:10px;
    width:100px;
    height:20px;
    background-color:white;
    border:black thin dotted;
    font-size:10px;
}
</style>
</head>
<body>
<input type="button" value="queue:false" id="btn1">
<input type="button" value="queue:true" id="btn2"><br>
<div id="div1" class="cls1">Текст<div>
</body>
</html>

```

7.4. Прерывание анимации

Если для одного элемента назначено несколько анимаций, то по умолчанию они выполняются по очереди. Пока не закончилась первая анимация, вторая не начнется. В случае, если анимация применена к разным элементам, то она будет выполняться одновременно. Для досрочного прерывания анимации предназначен метод `stop()`. Формат метода:

```
stop([<Очистка очереди> [, <Завершение анимации>]])
```

Если параметры не указаны, то прерывается только текущая анимация. Иными словами, если в очереди несколько анимаций, то прервется текущая анимация и сразу начнется следующая.

Параметр `<Очистка очереди>` позволяет очистить очередь анимаций. Для этого необходимо указать значение `true`. Тогда прервется не только текущая анимация, но и все остальные.

В случае применения метода `stop()` без указания параметра `<Завершение анимации>` текущая анимация будет прервана и объект остановится в промежуточном состоянии. Если необходимо, чтобы анимация была сразу завершена и объект находился в конечном состоянии, тогда во втором параметре необходимо указать значение `true`.

Пример использования метода `stop()` приведен в листинге 7.5.

Листинг 7.5. Прерывание анимации

```

<html>
<head>
<title>Прерывание анимации</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#btn1").click(function() {
        if ($("#chc1").is(":checked")) {
            $("#div1").stop().animate({ width: "0.1px" }, 3000);
        }
        else {

```

```

        $("#div1").animate({ width: "0.1px" }, 3000);
    }
});
$("#btn2").click(function() {
    if ($("#chc1").is(":checked")) {
        $("#div1").stop().animate({ width: "450px" }, 3000);
    }
    else {
        $("#div1").animate({ width: "450px" }, 3000);
    }
});
$("#btn3").click(function() {
    $("#div1").stop();
});
$("#btn4").click(function() {
    $("#div1").stop(true);
});
$("#btn5").click(function() {
    $("#div1").stop(true, true);
});
});
//-->
</script>
<style type="text/css">
.cls1 {
    position:absolute;
    top:50px;
    left:10px;
    width:450px;
    height:100px;
    background-color:black;
}
</style>
</head>
<body>
<input type="checkbox" id="chc1" checked> Прерывание анимации
<input type="button" value="Скрыть" id="btn1">
<input type="button" value="Отобразить" id="btn2">
<input type="button" value="Остановить" id="btn3">
<input type="button" value="Остановить все" id="btn4">
<input type="button" value="Остановить все и завершить"
id="btn5"><br>
<div id="div1" class="cls1"><div>
</body>
</html>

```

Если щелкнуть на кнопке **Скрыть**, а затем, не дожидаясь окончания анимации, щелкнуть на кнопке **Отобразить**, то при установленном флагажке текущая анимация будет прервана и начнется новая. Если флагажок сбросить, то следующая анимация будет выполнена только после окончания предыдущей.

Чтобы увидеть различные нюансы применения метода `stop()`, сбросьте флагажок и несколько раз щелкните на кнопках **Скрыть** и **Отобразить**, чтобы создать очередь анимаций. Если теперь щелкнуть на кнопке **Остановить**, то текущая анимация будет

прервана и сразу начнется следующая. Если щелкнуть на кнопке Остановить все, то текущая, а также все остальные анимации будут прерваны и объект остановится в промежуточном положении. Щелчок на кнопке Остановить все и завершить вызовет не только прерывание всех анимаций, но и полностью завершит текущую. Например, если ширина уменьшалась, то она сразу станет равна 0,1 px, а если увеличивалась, то ее значение станет равно 450 px.

7.5. Управление очередью анимаций

Для управления очередью анимаций предназначены следующие методы.

- `queue([<Название очереди>])` — возвращает массив функций в очереди. С помощью свойства `length` можно узнать количество анимаций в очереди.
`alert($("#div1").queue().length);`
- `queue([<Название очереди>,]<Функция обратного вызова>)` — добавляет функцию в очередь. В качестве параметра `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>() {  
    // ...  
    // Удаляем функцию из очереди после выполнения  
    $(this).dequeue();  
}
```

Элемент, вызвавший событие, доступен внутри функции через указатель `this`.

- `queue([<Название очереди>,]<Новая очередь>)` — заменяет очередь всех элементов коллекции новой очередью (массивом функций). Очистить очередь можно так.
`$("#div1").queue([]);`
- `dequeue([<Название очереди>])` — удаляет добавленную в очередь функцию после ее выполнения.

Необязательный параметр `<Название очереди>` во всех этих функциях по умолчанию имеет значение `fx`. Пример очистки очереди можно переписать следующим образом.

```
$("#div1").queue("fx", []);
```

Пример добавления функции в очередь приведен в листинге 7.6.

Листинг 7.6. Добавление функции в очередь

```
<html>  
<head>  
<title>Добавление функции в очередь</title>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
<script src="jquery.js" type="text/javascript"></script>  
<script type="text/javascript">  
!--  
$(document).ready(function() {  
    $("#btn1").click(function() {  
        var elem = $("#div1");  
        elem.css({ width:"100px", height:"20px", fontSize:"10px" });  
    })  
})
```

```
elem.text("Текст");
elem.animate({ width:"450px", height:"100px" }, 2000);
elem.queue(function() {
    $(this).text("Наш новый текст").dequeue();
});
});
//-->
</script>
<style type="text/css">
.cls1 {
    position:absolute;
    top:50px;
    left:10px;
    width:100px;
    height:20px;
    background-color:white;
    border:black thin dotted;
    font-size:10px;
}
</style>
</head>
<body>
<input type="button" value="Показать" id="btn1"><br>
<div id="div1" class="cls1">Текст</div>
</body>
</html>
```

Глава 8

Обработка данных формы

В этом разделе мы научимся обрабатывать данные, введенные пользователем в элементы формы. Обработка на стороне клиента позволит снизить нагрузку на сервер за счет отмены отправки данных формы при неправильно введенных значениях. Кроме того, в дальнейшем мы будем изучать технологию Ajax, которая позволяет отправлять данные без перезагрузки страницы. В этом случае заботиться о получении корректных данных формы необходимо самим.

8.1. Текстовое поле и поле ввода пароля

Создадим форму ввода адреса электронной почты и пароля и проверим правильность ввода перед отправкой данных на сервер (листинг 8.1). Если данные введены неправильно, то

- выводится сообщение об ошибке;
- поле выделяется розовым цветом;
- текст в поле выделяется;
- отправка формы прерывается.

Листинг 8.1. Проверка правильности ввода почтового адреса и пароля

```
<html>
<head>
<title>Проверка правильности ввода E-mail и пароля</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $(".text,:password").css("background-color", "#ffffff");
    $("form").submit(function() {
        var pole1 = $("#pole1");
        var pole2 = $("#pole2");
        pole1.css("background-color", "#ffffff");
        pole2.css("background-color", "#ffffff");
        var p = /^[a-z0-9_.\-\-]+@[a-z0-9\-\-]+\.\+[a-z]{2,4}$/i;
        if (!p.test(pole1.val())) {
            alert("Неверный адрес E-mail");
            pole1.css("background-color", "#ffe4e1").focus().select();
            return false;
        }
        p = /^[a-z0-9_.\-\-]{6,16}$/i;
        if (!p.test(pole2.val())) {
            alert("Неверный пароль");
            pole2.css("background-color", "#ffe4e1").focus().select();
        }
    })
})
```

```

        return false;
    }
    var msg = "Вы ввели следующие данные:\n\n E-mail: ";
    msg += pole1.val() + "\n Пароль: " + pole2.val();
    alert(msg);
    return true;
});
});
//-->
</script>
</head>
<body>
<form action="test.php" method="GET">
E-mail:<br>
<input type="text" name="pole1" id="pole1"><br>
Пароль:<br>
<input type="password" name="pole2" id="pole2"><br>
<input type="submit" value="Отправить">
</form>
</body>
</html>

```

В следующем примере (листинг 8.2) рассмотрим применение атрибутов `readOnly` и `disabled`. Создадим два текстовых поля и кнопку. Второе текстовое поле сделаем доступным только для чтения. Кроме того, изначально кнопка будет неактивна. При введении текста в первое поле кнопка становится активной, а сам текст копируется из первого поля во второе. Если щелкнуть на кнопке, то первое поле очищается и вызывается метод `keyup()`. За счет этого очищается второе поле, и кнопка деактивизируется. После кнопки добавим еще одно текстовое поле, в котором по умолчанию будет подсказка. При получении полем фокуса удаляем подсказку, а при потере фокуса, если поле осталось пустым, снова ее выводим.

Листинг 8.2. Атрибуты `readOnly` и `disabled`

```

<html>
<head>
<title>Атрибуты readOnly и disabled</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $(":text").css("background-color", "#ffffff");
    $("#pole2").attr("readonly", true);
    $("#btn1").attr("disabled", true);
    $("#pole1").keyup(function() {
        var pole1 = $("#pole1").val();
        var btn1 = $("#btn1");
        if (pole1.length == 0) {
            if (!btn1.is(":disabled")) btn1.attr("disabled", true);
        }
        else {
            if (btn1.is(":disabled")) btn1.removeAttr("disabled");

```

```

        }
        $("#pole2").val(pole1);
    });
    $("#btn1").click(function() {
        $("#pole1").val("").keyup();
    });
    $("#pole3").focus(function() {
        var pole3 = $("#pole3");
        if (pole3.val() == "[Подсказка]") pole3.val("");
    }).blur(function() {
        var pole3 = $("#pole3");
        if (pole3.val() == "") pole3.val("[Подсказка]");
    });
});
//-->
</script>
</head>
<body>
<input type="text" id="pole1"><br>
<input type="text" id="pole2"><br>
<input type="button" value="Очистить" id="btn1"><br><br>
<input type="text" id="pole3" value="[Подсказка]">
</body>
</html>

```

8.2. Поле для ввода многострочного текста

Рассмотрим возможность добавления слов из текстового поля в поле для ввода многострочного текста (листинг 8.3). Добавить слово можно с помощью кнопки **Добавить слово** или клавиши <Enter>. Так как по умолчанию нажатие <Enter> приводит к отправке данных формы, то событие прерывается за счет возврата значения `false`. При щелчке на кнопке **Значение поля** выводится текущее значение тега <textarea>.

Листинг 8.3. Добавление слов из текстового поля в поле <TEXTAREA>

```

<html>
<head>
<title>Добавление слов из текстового поля в поле TEXTAREA</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
function f_click() {
    var pole1 = $("#pole1");
    var pole2 = $("#pole2");
    var txt1 = pole1.val();
    var txt2 = pole2.val();
    if (txt1 != "") {
        pole2.val(txt2 + txt1 + "\n");
        pole1.val("").focus();
    }
    else {
        alert("Поле не заполнено!");
    }
}
-->
</script>

```

```

        pole1.focus();
    }
}
$(document).ready(function() {
    $("#pole1").focus();
    $("form").submit(function() {
        var val = $("#pole2").val();
        alert("Текущее значение: \n" + val);
        $("#pole1").focus();
        return false;
    });
    $("#pole1").keypress(function(e) {
        if (e.keyCode==13) {
            f_click();
            return false;
        }
    });
    $("#btn1").click(function() {
        f_click();
    });
});
//-->
</script>
</head>
<body>
<form action="test.php" method="GET">
Слово:<br>
<input type="text" name="pole1" id="pole1"><br>
<input type="button" value="Добавить слово" id="btn1"><br>
<textarea name="pole2" id="pole2" cols="15" rows="10">
</textarea><br>
<input type="submit" value=" Значение поля ">
</form>
</body>
</html>

```

8.3. СПИСОК С ВОЗМОЖНЫМИ ЗНАЧЕНИЯМИ

Пример, представленный в листинге 8.4, демонстрирует следующие возможности.

- Добавление нового пункта списка. При заполнении первого поля и нажатии клавиши <Enter> фокус ввода перемещается во второе поле. При заполнении второго поля и нажатии клавиши <Enter> введенные значения добавляются в список. Вместо клавиши <Enter> можно воспользоваться кнопкой Добавить.
- Получение всех выбранных значений из списка с возможностью множественного выбора.
- Применение взаимосвязанных списков и получение значения выбранного пункта. При выборе элемента в первом списке загружаются соответствующие элементы во второй список. При выборе элемента во втором списке выводится сообщение со значением выбранного пункта.

- Применение списков вместо гиперссылок. При выборе элемента списка загружается веб-страница, находящаяся по указанному в параметре value URL-адресу.

Листинг 8.4. Обработка списков

```

<html>
<head>
<title>Пример обработки списков</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
var arr = [];
arr[1] = ["Тема1 Элемент1", "Тема1 Элемент2"];
arr[2] = ["Тема2 Элемент1", "Тема2 Элемент2", "Тема2 Элемент3"];
var val = [];
val[1] = ["1", "2"];
val[2] = ["3", "4", "5"];
function f_click() {
    // Добавление пункта в список
    var pole1 = $("#pole1");
    var pole2 = $("#pole2");
    var text1 = pole1.val();
    var text2 = pole2.val();
    if (text1 != "" && text2 != "") {
        var sell = $("#select1").get(0);
        var i = sell.length++;
        sell.options[i].text = text1;
        sell.options[i].value = text2;
        pole1.val("");
        pole2.val("");
        pole1.focus();
    }
    else {
        alert("Поле не заполнено!");
        pole1.focus();
    }
}
$(document).ready(function() {
    // Добавление пункта в список
    $("#pole1").keypress(function(e) {
        if (e.keyCode==13) {
            $("#pole2").focus();
            return false;
        }
    });
    $("#pole2").keypress(function(e) {
        if (e.keyCode==13) {
            f_click();
            return false;
        }
    });
    $("#btn1").click(function() {
        f_click();
    });
})

```

```

});  

// Список со множественным выбором  

$("#btn2").click(function() {  

    var msg = "";  

    $("#select2 option:selected").each(function() {  

        msg += this.value + " - " + this.text + "\n";  

    });  

    if (msg == "") alert("Выбранных пунктов нет");  

    else alert(msg);  

});  

// Взаимосвязанные списки  

$("#select3").change(function() {  

    var index = this.value;  

    var count = arr[index].length;  

    var el = $("#select4").get(0);  

    el.length = count;  

    for (i=0; i<count; i++) {  

        el.options[i].value = val[index][i];  

        el.options[i].text = arr[index][i];  

    }  

});  

$("#select4").change(function() {  

    var msg = "";  

    $("#select4 option:selected").each(function() {  

        msg = "Значение: " + this.value;  

        msg += "\nТекст: " + this.text;  

    });  

    alert(msg);  

});  

// Переход на указанный сайт  

$("#select5").change(function() {  

    var url = $(this).val();  

    if (url != 0) {  

        window.location.href = url;  

    }
});  

//-->
</script>
</head>
<body>
<b>Добавление пункта в список:</b><br><br>
Текст пункта:<br>
<input type="text" id="pole1"><br>
Значение пункта:<br>
<input type="text" id="pole2"><br>
<select id="select1">
</select><br>
<input type="button" value="Добавить" id="btn1"><br><br>

<b>Список со множественным выбором:</b><br><br>
<select id="select2" size="5" multiple>
<option value="1" selected>Элемент1</option>
<option value="2">Элемент2</option>

```

```

<option value="3">Элемент3</option>
<option value="4">Элемент4</option>
<option value="5">Элемент5</option>
<option value="6">Элемент6</option>
</select><br>
<input type="button" value="Значения списка" id="btn2"><br><br>

<b>Взаимосвязанные списки:</b><br><br>
<select id="select3" size="5">
<option value="1">Тема1</option>
<option value="2">Тема2</option>
</select><br>
<select id="select4">
<option value="1" selected>Тема1 Элемент1</option>
<option value="2">Тема1 Элемент2</option>
</select><br><br>

<b>Переход на указанный сайт:</b><br><br>
<select id="select5">
<option value="0" selected>-----</option>
<option value="http://www.mail.ru/">Mail.ru</option>
<option value="http://www.rambler.ru/">Рамблер</option>
</select><br>
</body>
</html>

```

8.4. Флажок и переключатели

В листинге 8.5 рассматривается получение значений флажков и переключателей. Первая форма демонстрирует проверку установки флажка и получение выбранного переключателя в группе. Если флажок установлен, то выводим значение параметра value. Вторая форма демонстрирует поиск установленных флажков в группе, а также возможность установки и снятия всех флажков сразу.

Листинг 8.5. Обработка флажков и переключателей

```

<html>
<head>
<title>Обработка флажков и переключателей</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#btn1").click(function() {
        var msg = "";
        var elem = $("#check1");
        if (elem.is(":checked")) {
            msg = "Флажок установлен\n";
            msg += "Значение: " + elem.val() + "\n";
        }
        else {
            msg = "Флажок снят\n";
        }
    });
});

```

```

        }
        var value1 = $("input[id='radio1']:checked").val();
        if (value1 == "male") {
            msg += "Пол: Мужской\n";
        }
        else {
            msg += "Пол: Женский\n";
        }
        alert(msg);
    });
    $("#c0").click(function() {
        if ($("#c0").is(":checked")) {
            $("#frm2 :checkbox").attr("checked", true);
        }
        else {
            $("#frm2 :checkbox").removeAttr("checked");
        }
    });
    $("#btn2").click(function() {
        var msg = "";
        $("#frm2 input[id='c[]']:checked").each(function() {
            msg += "Значение: " + this.value + "\n";
        });
        if (msg == "") alert("Флажки не установлены");
        else alert(msg);
    });
});
//-->
</script>
</head>
<body>
<form action="test.php" method="GET" id="frm1">
<input type="checkbox" name="check1" id="check1" value="yes" checked>
Текст<br><br>
Укажите ваш пол:<br>
<input type="radio" name="radio1" id="radio1" value="male" checked>
Мужской
<input type="radio" name="radio1" id="radio1"
value="female">Женский
<br><br>
<input type="button" value="Вывести значения" id="btn1">
</form>
<form action="test.php" method="GET" id="frm2">
<input type="checkbox" name="c0" id="c0" value="1"> Отметить/Снять все<br>
<input type="checkbox" name="c[]" id="c[]" value="1"> Пункт 1<br>
<input type="checkbox" name="c[]" id="c[]" value="2"> Пункт 2<br>
<input type="checkbox" name="c[]" id="c[]" value="3"> Пункт 3<br>
<input type="button" id="btn2" value="Вывести значения">
</form>
</body>
</html>

```

8.5. Обработка щелчка на кнопке

В приведенном ниже примере (листинг 8.6) кнопка изначально неактивна. При вводе в текстовое поле кнопка активизируется. При щелчке на кнопке текст, введенный в текстовое поле, отображается на кнопке. Текстовое поле очищается, и кнопка вновь деактивизируется.

Листинг 8.6. Обработка щелчка на кнопке

```
<html>
<head>
<title>Обработка щелчка на кнопке</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#btn1").click(function() {
        var elem = $("#txt1");
        this.value = elem.val();
        elem.val("");
        $(this).attr("disabled", true);
    });
    $("#txt1").keyup(function() {
        var txt1 = $("#txt1").val();
        var btn1 = $("#btn1");
        if (txt1.length == 0) {
            if (!btn1.is(":disabled")) btn1.attr("disabled", true);
        }
        else {
            if (btn1.is(":disabled")) btn1.removeAttr("disabled");
        }
    });
});
//-->
</script>
</head>
<body>
<input type="text" id="txt1"><br>
<input type="button" value="Изменить текст на кнопке"
       id="btn1" disabled>
</body>
</html>
```

Обычная командная кнопка может быть вставлена в веб-страницу не только с помощью тега `<input>`, но и с помощью парного тега `<button>`. В качестве примера выведем порядковый индекс кнопки, на которой был сделан щелчок (листинг 8.7).

Листинг 8.7. Вывод порядкового индекса кнопки, на которой был сделан щелчок

```
<html>
<head>
<title>Вывод порядкового индекса нажатой кнопки</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

```

<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $(":button").click(function() {
        alert("Индекс: " + $(":button").index(this));
    });
//-->
</script>
</head>
<body>
<input type="button" value="Кнопка 1"><br>
<input type="button" value="Кнопка 2"><br>
<button>Кнопка 3</button><br>
<button>Кнопка 4</button>
</body>
</html>

```

8.6. Получение всех значений формы

Библиотека jQuery позволяет получить все данные формы за один раз и подготовить их для передачи на сервер. Для этого предназначены следующие методы.

- `serialize()` — получает все данные формы и составляет запрос, который можно отправить с помощью технологии AJAX. Метод позволяет обработать данные не только всей формы, но и отдельных элементов.

```

$( "#btn1" ).click(function() {
    alert($("#txt1").serialize());
});

```

```

<input type="text" name="txt1" id="txt1" value="Текст"><br>
<input type="button" id="btn1" value="Получить значение">

```

Результат:

```
txt1=%D0%A2%D0%B5%D0%BA%D1%81%D1%82
```

- `serializeArray()` — возвращает массив всех данных формы. Первым элементом является порядковый номер элемента внутри формы. Нумерация начинается с 0. Второй элемент представляет собой объект с двумя свойствами:
 - ◊ `name` — название поля в форме;
 - ◊ `value` — значение поля.

Метод позволяет обработать данные не только всей формы, но и отдельных элементов.

```

$( "#btn1" ).click(function() {
    var arr = $("#txt1").serializeArray();
    $("#div1")
        .html("Поле: "+arr[0].name+"<br>Значение: "+arr[0].value);
});

```

```
<input type="text" name="txt1" id="txt1" value="Текст"><br>
```

```
<input type="button" id="btn1" value="Получить значение"><br>
<div id="div1"></div>
```

Результат:

Поле: txt1

Значение: Текст

- `$.param(<Объект>)` — позволяет преобразовать объект в строку, предназначенную к отправке на сервер. Применяется как обычная функция, объявленная в пространстве имен библиотеки jQuery. Не требует предварительного нахождения коллекции элементов.

```
$("#btn1").click(function() {
    $("#div1").text($.param($("#txt1").serializeArray()));
});
```

```
<input type="text" name="txt1" id="txt1" value="Текст"><br>
<input type="button" id="btn1" value="Получить значение"><br>
<div id="div1"></div>
```

Результат:

txt1=%D0%A2%D0%B5%D0%BA%D1%81%D1%82

В качестве параметра можно передать коллекцию элементов. В качестве примера получим значения всех текстовых полей.

```
$("#btn1").click(function() {
    $("#div1").text($.param(":text"));
});
```

```
<input type="text" name="txt1" id="txt1" value="1"><br>
<input type="text" name="txt2" id="txt2" value="2"><br>
<input type="button" id="btn1" value="Получить значения"><br>
<div id="div1"></div>
```

Результат:

txt1=1&txt2=2

Функция позволяет обработать не только данные формы, но и обычные объекты.

```
var obj = {var1:"Текст", var2:5};
alert($.param(obj));
```

Результат:

var1=%D0%A2%D0%B5%D0%BA%D1%81%D1%82&var2=5

Пример обработки данных всей формы разными методами приведен в листинге 8.8.

Листинг 8.8. Методы `serialize()`, `serializeArray()` и `$.param()`

```
<html>
<head>
<title>Методы serialize(), serializeArray() и $.param()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
```

```

$(document).ready(function() {
    $("#btn1").click(function() {
        $("#div1").text($("#frm1").serialize());
        var arr = $("#frm1").serializeArray();
        var txt = "";
        for (var i=0, count=arr.length; i<count; i++) {
            txt += i + " > " + arr[i].name + " = ";
            txt += arr[i].value + "<br>";
        }
        $("#div2").html(txt);
        $("#div3").text($.param(arr));
    });
}); //-->
</script>
</head>
<body>
<form action="test.php" method="GET" id="frm1">
<input type="text" name="txt1" id="txt1" value="Текст"><br>
<input type="hidden" name="hdn1" id="hdn1" value="1">
<input type="checkbox" name="check1" id="check1" value="yes">
Текст<br>
Укажите ваш пол:<br>
<input type="radio" name="radio1" id="radio1" value="male"
checked>Мужской
<input type="radio" name="radio1" id="radio1"
value="female">Женский
<br>
<input type="checkbox" name="c[]" id="c[]" value="1"> Пункт 1<br>
<input type="checkbox" name="c[]" id="c[]" value="2"> Пункт 2<br>
<input type="checkbox" name="c[]" id="c[]" value="3"> Пункт 3<br>
</form>
<input type="button" id="btn1" value="Получить значения"><br><br>
<b>serialize():</b><br>
<div id="div1"></div>
<b>serializeArray():</b><br>
<div id="div2"></div>
<b>$ .param():</b><br>
<div id="div3"></div>
</body>
</html>

```

Результат при значениях по умолчанию:

```

serialize():
txt1=%D0%A2%D0%B5%D0%BA%D1%81%D1%82&hdn1=1&radio1=ma
serializeArray():
0 => txt1 = Текст
1 => hdn1 = 1
2 => radio1 = male
$.param():
txt1=%D0%A2%D0%B5%D0%BA%D1%81%D1%82&hdn1=1&radio1=ma

```

Глава 9

Вспомогательные функции и свойства

Кроме уже рассмотренных методов, библиотека jQuery предоставляет различные вспомогательные функции и свойства, которые расширяют возможности JavaScript. Для их использования нет необходимости создавать коллекцию элементов. Они применяются как обычные функции, объявленные в пространстве имен библиотеки jQuery.

9.1. Функция `$.each()` — перебор элементов

Функция `$.each()` предназначена для перебора массивов и объектов. В отличие от метода `each()`, она не требует предварительного нахождения коллекции элементов и применяется как обычная функция, объявленная в пространстве имен библиотеки jQuery. Функция имеет следующий синтаксис.

```
$.each(<Массив или объект>, <Функция обратного вызова>);
```

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Индекс или ключ>[, <Значение>]]) {  
    // ...  
}
```

Если параметры не указаны, то значение доступно через переменную `this`. В качестве примера выведем все элементы массива.

```
var arr = [ 1, 2, 3 ];  
$.each(arr, function() {  
    $("#div1").append(this + "<br>");  
});
```

Теперь выведем элементы массива, указав индекс и значение.

```
var arr = [ 1, 2, 3 ];  
$.each(arr, function(ind, val) {  
    $("#div1").append(ind + " => " + val + "<br>");  
});
```

Результат:

```
0 => 1  
1 => 2  
2 => 3
```

Перебрать элементы объекта можно следующим образом.

```
var obj = { "Один": 1, "Два": 2, "Три": 3 };  
$.each(obj, function(key, val) {  
    $("#div1").append(key + " => " + val + "<br>");  
});
```

Результат:

```
Один => 1
Два => 2
Три => 3
```

Если необходимо досрочно прервать выполнение функции `$.each()`, то внутри функции обратного вызова следует вернуть значение `false`.

```
var obj = { "Один": 1, "Два": 2, "Три": 3 };
$.each(obj, function(key, val) {
    $("#div1").append(key + " => " + val + "<br>");
    if (val == 2) {
        return false;
    }
});
```

Результат:

```
Один => 1
Два => 2
```

9.2. Функция `$.grep()` — поиск в массиве

Функция `$.grep()` позволяет произвести поиск в массиве. Возвращает новый массив, элементы которого соответствуют условию. Применяется как обычная функция, объявленная в пространстве имен библиотеки jQuety. Имеет следующий синтаксис.

```
<Массив2> = $.grep(<Массив1>, <Функция обратного вызова>[, <Инверсия>]);
```

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>(<Значение>[, <Индекс>]) {
    // ...
}
```

Если значение соответствует условию, то функция обратного вызова должна вернуть значение `true`, в противном случае — `false`. Предположим, есть массив, заполненный числами от 1 до 5. Создадим новый массив из исходного, в котором все элементы массива меньше или равны 3.

```
var arr = [ 1, 2, 3, 4, 5 ];
arr = $.grep(arr, function(val) {
    if (val <= 3) {
        return true;
    }
    else {
        return false;
    }
});
$("#div1").html(arr.join(", "));
```

Результат:

```
1, 2, 3
```

Если в функции `$.grep()` в параметре `<Инверсия>` указать значение `true`, то результат будет изменен на противоположный.

```

function check(val) {
    if (val <= 3) {
        return true;
    }
    else {
        return false;
    }
}
$(document).ready(function() {
    var arr = [ 1, 2, 3, 4, 5 ];
    arr = $.grep(arr, check, true);
    $("#div1").html(arr.join(", "));
});

```

Результат:

4, 5

9.3. Функция `$.map()` — преобразование массива

Функция `$.map()` позволяет изменить каждый элемент массива. Возвращает новый массив. Применяется как обычная функция, объявленная в пространстве имен библиотеки jQuery. Имеет следующий синтаксис.

`<Новый массив> = $.map(<Массив>, <Функция обратного вызова>);`

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```

function <Название функции>(<Значение>[, <Индекс>]) {
    // ...
}

```

Чтобы изменить текущее значение элемента массива, необходимо внутри функции обратного вызова вернуть новое значение. В качестве примера умножим все элементы массива на 2.

```

var arr = [ 1, 2, 3 ];
arr = $.map(arr, function(val) {
    return (val * 2);
});
$("#div1").html(arr.join(", "));

```

Результат:

2, 4, 6

Если в качестве значения вернуть `null`, то элемент не будет добавлен в новый массив. Для примера первый элемент массива удалим, второй элемент умножим на 2, а третий оставим без изменений.

```

var arr = [ 1, 2, 3 ];
arr = $.map(arr, function(val, ind) {
    if (ind == 1) {
        return (val * 2);
    }
    else if (ind == 0) {

```

```
        return null;
    }
    else {
        return val;
    }
);
$("#div1").html(arr.join(", "));
```

Результат:

```
4, 3
```

9.4. Функция `$.inArray()` — поиск элемента в массиве

Функция `$.inArray()` позволяет выполнить поиск элемента в массиве. Поиск производится с учетом регистра символов. Возвращает индекс первого найденного элемента или значение `-1`, если ничего не найдено. Применяется как обычная функция, объявленная в пространстве имен библиотеки jQuery. Имеет следующий синтаксис.

```
<Результат> = $.inArray(<Значение>, <Массив>);
```

Рассмотрим пример.

```
var arr = ["Один", "Два", 3, "Два"];
alert($.inArray("Два", arr)); // 1
alert($.inArray(4, arr)); // -1
```

9.5. Функция `$.merge()` — объединение массивов

Функция `$.merge()` позволяет объединить два массива в один. Возвращает новый массив. Применяется как обычная функция, объявленная в пространстве имен библиотеки jQuery. Формат функции:

```
<Новый массив> = $.merge(<Массив1>, <Массив2>);
```

Рассмотрим пример.

```
var arr = $.merge([1,2,3], [1,2,3]);
alert(arr.join(", "));
```

Результат:

```
1, 2, 3, 1, 2, 3
```

9.6. Функция `$.makeArray()` — создание массива элементов

Функция `$.makeArray()` создает массив из выбранных DOM-элементов. Применяется как обычная функция, объявленная в пространстве имен библиотеки jQuery. Формат функции:

```
<Массив> = $.makeArray(<DOM-элементы>);
```

Пример использования функции `$.makeArray()` приведен в листинге 9.1.

Листинг 9.1. Функция \$.makeArray()

```
<html>
<head>
<title>Функция $.makeArray()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#div1").click(function() {
        var arr = $.makeArray($(".p").get()).reverse();
        $("#div1").append(arr);
    });
//-->
</script>
</head>
<body>
<div id="div1">Нажмите здесь</div>
<p>Абзац1</p>
<p>Абзац2</p>
<p>Абзац3</p>
<p>Абзац4</p>
</body>
</html>
```

9.7. Функция \$.unique() — удаление повторяющихся элементов

Функция `$.unique()` позволяет удалить все повторяющиеся элементы из массива DOM-элементов. Возвращает новый массив. Применяется как обычная функция, объявленная в пространстве имен библиотеки jQuery. Имеет следующий синтаксис.

`<Новый массив> = $.unique(<Исходный массив DOM-элементов>);`

Обратите внимание

Функция `$.unique()` работает только с массивами DOM-элементов. Если элементы массива будут состоять из строк или чисел, то результата не будет. Функция для этого не предназначена.

Рассмотрим пример использования функции `$.unique()` (листинг 9.2).

Листинг 9.2. Функция \$.unique()

```
<html>
<head>
<title>Функция $.unique()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
$(document).ready(function() {
```

```

var arr1 = $("a[href$='.php']").get();
var arr2 = $(".cls1").get();
$("#div1").append("Элементов в arr1: " + arr1.length + "<br>");
$("#div1").append("Элементов в arr2: " + arr2.length + "<br>");
// Объединяем два массива
var arr3 = arr1.concat(arr2);
$("#div1").append("Элементов в arr3: " + arr3.length + "<br>");
// Оставляем только уникальные значения
arr3 = $.unique(arr3);
$("#div1").append("Уникальных элементов: " + arr3.length + "<br>");

});
</script>
<style>
.cls1 { color:red; }
</style>
</head>
<body>
<div id="div1"></div>
<a href="link1.html" class="cls1">Ссылка 1</a><br>
<a href="link2.php" class="cls1">Ссылка 2</a><br>
<a href="link3.html">Ссылка 3</a><br>
<a href="link4.php">Ссылка 4</a><br>
<p class="cls1">Абзац</p>
</body>
</html>

```

Результат:

Элементов в arr1: 2

Элементов в arr2: 3

Элементов в arr3: 5

Уникальных элементов: 4

Итак, вначале находим все ссылки, которые имеют расширение .php. С помощью метода `get()` получаем массив DOM-элементов. Наш массив будет состоять из следующих элементов:

```

<a href="link2.php" class="cls1">Ссылка 2</a><br>
<a href="link4.php">Ссылка 4</a><br>

```

Далее мы получаем все элементы с классом `cls1` и с помощью метода `get()` преобразуем их в массив DOM-элементов. Массив будет состоять из следующих элементов:

```

<a href="link1.html" class="cls1">Ссылка 1</a><br>
<a href="link2.php" class="cls1">Ссылка 2</a><br>
<p class="cls1">Абзац</p>

```

После объединения массивов получаем повторяющийся элемент.

```

<a href="link2.php" class="cls1">Ссылка 2</a><br>

```

Функция `$.unique()` оставит этот элемент в единственном числе, и в результате мы получим четыре элемента вместо пяти.

```

<a href="link2.php" class="cls1">Ссылка 2</a><br>
<a href="link4.php">Ссылка 4</a><br>
<a href="link1.html" class="cls1">Ссылка 1</a><br>
<p class="cls1">Абзац</p>

```

9.8. Функция `$.trim()` — удаление пробельных символов

Функция `$.trim()` позволяет удалить пробельные символы в начале и конце строки. Возвращает новую строку. Применяется как обычная функция, объявленная в пространстве имен библиотеки jQuery. Имеет следующий синтаксис.

```
<Новая строка> = $.trim(<Исходная строка>);
```

Рассмотрим пример.

```
var str = "      Стока с пробелами      ";
alert("До: '" + str + "'");
alert("После: '" + $.trim(str) + "'");
```

В результате получим следующее.

```
До: ' Стока с пробелами '
После: 'Стока с пробелами'
```

9.9. Функции `$.data()` и `$.removeData()` — работа с данными

Функция `$.data()`, предназначенная для сохранения и получения данных, имеет следующий формат.

```
$.data(<DOM-элемент> [, <Название> [, <Значение>]])
```

Если указан только первый параметр, то функция возвращает уникальный идентификатор элемента. Выведем значение идентификатора.

```
alert($.data($("#div1").get(0)));
```

В параметре `<Название>` можно указать название данных, а в параметре `<Значение>` — данные, которые должны быть сохранены. Формат сохраняемых данных может быть любым. Сохраним строку под названием `myData` в элементе с идентификатором `div1`.

```
var elem = $("#div1").get(0);
$.data(elem, "myData", "Строка с данными");
```

Получить данные можно следующим образом.

```
var elem = $("#div1").get(0);
alert("Данные: " + $.data(elem, "myData"));
```

Теперь сохраним объект.

```
var elem = $("#div1").get(0);
$.data(elem, "myData", { el1:"Строка1", el2:"Строка2" });
```

Получить данные можно так.

```
var obj = $.data($("#div1").get(0), "myData");
alert("el1: " + obj.el1 + "\n" + "el2: " + obj.el2);
```

Сохраним массив.

```
var elem = $("#div1").get(0);
$.data(elem, "myData", ["Строка1", "Строка2"]);
```

Получим такие данные.

```
var arr = $.data($("#div1").get(0), "myData");
alert(arr[0] + "\n" + arr[1]);
```

Функция `$.removeData()`, предназначенная для удаления данных, имеет следующий формат.

```
$.removeData(<DOM-элемент>[, <Название>])
```

Если второй параметр не указан, то будут удалены все данные. Пример сохранения, получения и удаления данных приведен в листинге 9.3.

Листинг 9.3. Сохранение, получение и удаление данных

```
<html>
<head>
<title>Сохранение, получение и удаление данных</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#btn1").click(function() {
        var elem = $("#div1").get(0);
        $.data(elem, "myData", $("#txt1").val());
        $("#div2").text("Данные сохранены");
    });
    $("#btn2").click(function() {
        var elem = $("#div1").get(0);
        $("#div2").text("Данные: " + $.data(elem, "myData"));
    });
    $("#btn3").click(function() {
        var elem = $("#div1").get(0);
        $.removeData(elem, "myData");
        $("#div2").text("Данные удалены");
    });
});
//--
</script>
</head>
<body>
<div id="div1"></div>
<div id="div2"></div><br>
<input type="text" id="txt1">
<input type="button" value="Сохранить" id="btn1">
<input type="button" value="Получить" id="btn2">
<input type="button" value="Удалить" id="btn3">
</body>
</html>
```

Вместо функций `$.data()` и `$.removeData()`, можно использовать методы `data()` и `removeData()`. Формат метода `data()`:

```
data(<Название>[, <Значение>])
```

Если параметр <Значение> не указан, то функция возвращает текущие данные первого элемента коллекции. Если же параметр указан, то данные будут сохранены во всех элементах коллекции.

Для удаления данных предназначен метод `removeData()`. Формат метода:

```
removeData([<Название>])
```

Если параметр не указан, то будут удалены все данные.

Переделаем наш предыдущий пример и используем методы `data()` и `removeData()` вместо функций `$.data()` и `$.removeData()` (листинг 9.4).

Листинг 9.4. Методы `data()` и `removeData()`

```
<html>
<head>
<title>Методы data() и removeData()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#btn1").click(function() {
        $("#div1").data("myData", $("#txt1").val());
        $("#div2").text("Данные сохранены");
    });
    $("#btn2").click(function() {
        $("#div2").text("Данные: " + $("#div1").data("myData"));
    });
    $("#btn3").click(function() {
        $("#div1").removeData("myData");
        $("#div2").text("Данные удалены");
    });
});
//-->
</script>
</head>
<body>
<div id="div1"></div>
<div id="div2"></div><br>
<input type="text" id="txt1">
<input type="button" value="Сохранить" id="btn1">
<input type="button" value="Получить" id="btn2">
<input type="button" value="Удалить" id="btn3">
</body>
</html>
```

9.10. Свойство `$.browser` — определение типа и версии браузера

Определить тип и версию браузера позволяет свойство `$.browser`. Оно возвращает несколько свойств: `safari`, `opera`, `msie` и `mozilla`. В зависимости от типа браузера, соответствующее свойство будет содержать значение `true`, а все остальные свойства — значение `false`. Это позволяет определить тип браузера следующим образом.

```
if ($.browser.msie) {  
    alert("Это Internet Explorer");  
}  
}
```

Версия браузера доступна через свойство `version`.

```
if ($.browser.msie) {  
    alert("Это Internet Explorer " + $.browser.version);  
}  
}
```

Вывести значения всех свойств можно с помощью функции `$.each()`.

```
$.each($.browser, function(key, val) {  
    $("#div1").append(key + " => " + val + "<br>");  
});  
}
```

Бот вывод для браузера Internet Explorer 7.

```
version => 7.0  
safari => false  
opera => false  
msie => true  
mozilla => false
```

Обратите внимание

Начиная с версии jQuery 1.3 свойство `$.browser` признано устаревшим и не рекомендуется к использованию. Вместо `$.browser` следует использовать свойство `$.support`.

9.11. Свойство `$.boxModel` — определение блочной модели

Свойство `$.boxModel` позволяет определить, какой тип блочной модели применяет браузер. Возвращает значение `true`, если блочная модель соответствует стандартам консорциума W3C, и `false` — если браузер использует простой режим (так называемый режим Quirks). В основном, это касается браузера Internet Explorer, который использует простой режим в случае, если отсутствует заголовок `DOCTYPE`, и строгий режим — при его наличии. Из этого правила могут быть исключения. Например, в Internet Explorer 6, если перед заголовком `DOCTYPE` указан XML-пролог

```
<?xml version="1.0" encoding="utf-8"?>,
```

браузер перейдет в простой режим. В браузере Internet Explorer 7 это поведение было изменено.

Примечание

Более подробную информацию о типах блочной модели можно получить в Интернете на странице консорциума W3C <http://www.w3.org/TR/CSS2/box.html> и на странице <http://www.quirksmode.org/css/quirksmode.html>.

Рассмотрим определение блочной модели на примере (листинг 9.5).

Листинг 9.5. Определение блочной модели

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xml:lang="en" lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Определение блочной модели</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    if ($.boxModel) {
        $("#div1").html("Строгий режим");
    }
    else {
        $("#div1").html("Простой режим");
    }
});
//-->
</script>
</head>
<body>
<div id="div1"></div>
</body>
</html>
```

В этом примере мы указали заголовок:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Это обязывает браузер соответствовать стандартам консорциума W3C. Браузер Internet Explorer 7 в этом случае выведет надпись Строгий режим. Если убрать заголовок DOCTYPE и обновить страницу, то надпись изменится на Простой режим.

Обратите внимание

Начиная с версии jQuery 1.3 свойство `$.boxModel` признано устаревшим и не рекомендуется к использованию. Вместо `$.boxModel` следует использовать свойство `$.support.boxModel`.

Для версии jQuery 1.3 наш код следует переписать следующим образом.

```
$(document).ready(function() {
    if ($.support.boxModel) {
        $("#div1").html("Строгий режим");
    }
    else {
        $("#div1").html("Простой режим");
    }
});
```

Свойство `$.support` содержит еще несколько свойств, позволяющих учитывать нюансы различных браузеров. Более подробную информацию о свойстве `$.support` можно получить на странице <http://docs.jquery.com/Utilities/jQuery.support>.

9.12. Создание собственных модулей

Библиотека jQuery предоставляет возможность расширения функциональности за счет добавления новых методов и функций. Вы можете создать собственный модуль и добавить его в хранилище на сайте разработчика библиотеки. Модуль принято размещать в отдельном файле с названием.

```
jquery.<Название модуля>.js
```

Код в модуле обычно располагают внутри следующей конструкции:

```
(function($) {
    // Код модуля
})(jQuery);
```

Символ \$, указанный в качестве параметра, означает, что мы можем использовать этот символ внутри конструкции как псевдоним функции `jQuery()`. Вместо этого символа можно указать любой другой допустимый идентификатор и использовать его внутри конструкции.

```
(function(j$) {
    // Код модуля
})(jQuery);
```

В этом случае мы можем использовать идентификатор `j$` вместо символа `$`.

Подключать модуль необходимо после подключения библиотеки `jQuery`.

```
<script src="jquery.js" type="text/javascript"></script>
<!-- Подключение модуля -->
<script src="jquery.myPlugin.js" type="text/javascript"></script>
```

Создать собственный метод можно, добавив его в объект `fn`. Для примера добавим метод `newcolor()`, позволяющий изменить цвет текста.

```
(function($) {
    $.fn.newcolor = function() {
        return this.css("color", "red");
    }
})(jQuery);

$("div").click(function() {
    $(this).newcolor();
});
```

```
<div>Нажмите для изменения цвета</div>
```

Внутри нового метода доступен указатель `(this)` на коллекцию элементов `jQuery`. По этой причине мы можем использовать метод `css()` для изменения цвета текста. Для того чтобы не прерывать цепочку вызовов, метод должен возвращать указатель. В нашем примере этот указатель возвращает метод `css()`, а мы в свою очередь возвращаем его из метода `newcolor()`.

Добавить собственный метод можно с помощью функции `$.fn.extend()`. В качестве параметра указывается объект. Переделаем наш предыдущий пример и используем функцию `$.fn.extend()`.

```
(function($) {
    $.fn.extend({
```

```

    newcolor: function() {
        return this.css("color", "red");
    }
});
}) (jQuery);

```

Добавить функцию, которая будет доступна без создания коллекции элементов, можно с помощью функции `$.extend()`. В качестве параметра указывается объект. Для примера добавим функцию суммирования двух чисел.

```

(function($) {
    $.extend({
        sum: function(x, y) {
            x = parseInt(x);
            if (isNaN(x)) x = 0;
            y = parseInt(y);
            if (isNaN(y)) y = 0;
            return (x + y);
        }
    });
}) (jQuery);

$("#btn1").click(function() {
    alert($.sum($("#txt1").val(), $("#txt2").val()));
});

число 1: <input type="text" id="txt1"><br>
число 2: <input type="text" id="txt2"><br>
<input type="button" id="btn1" value="Получить сумму чисел">

```

После щелчка на кнопке **Получить сумму чисел** будет вызвана функция `$.sum()`. В качестве параметров функции мы указываем значения текстовых полей.

Функция `$.extend()` позволяет также произвести слияние нескольких объектов. Формат функции:

```
<Объект> = $.extend(<Начальный объект>, <Объект1>[, ..., <ОбъектN>])
```

Если в начальном объекте существует такое же свойство, как в других объектах, то его значение будет перезаписано, а если не существует, то свойство будет добавлено. Это позволяет определять значения опций по умолчанию. В качестве примера добавим метод `newmsg()`, который выводит сообщение.

```

(function($) {
    $.fn.extend({
        newmsg: function(obj) {
            var m = { message: "Сообщение по умолчанию" };
            m = $.extend(m, obj);
            alert(m.message);
            return this;
        }
    });
}) (jQuery);

$("p").click(function() {
    $(this).newmsg().newmsg({ message:"Новое сообщение" });
});

<p>Нажмите для вывода сообщений</p>

```

В этом примере при щелчке на абзаце вначале будет выведено сообщение `Сообщение по умолчанию`, так как мы не передали параметр. Затем получим сообщение `Новое сообщение` за счет перезаписи значения свойства `message` с помощью функции `$.extend()`. Обратите внимание на цепочку вызовов.

```
$(this).newmsg().newmsg({ message:"Новое сообщение" }) ;
```

Это достигается за счет возвращения указателя из метода с помощью строки:

```
return this;
```

В конец этой цепочки можно добавлять другие методы. Например, выведем сообщение, а затем изменим цвет текста абзаца.

```
$(this).newmsg().newmsg({ message:"Новое сообщение" })
.css("color", "red") ;
```

Глава 10

Основы технологии AJAX

A JAX (Asynchronous Javascript And XML, асинхронный JavaScript и XML) — это технология взаимодействия веб-браузера с сервером без перезагрузки страницы. Веб-браузер в фоновом режиме отправляет запрос серверу. На сервере скрипт обрабатывает переданный запрос и отправляет ответ в виде XML-документа, фрагмента HTML-документа, скрипта JavaScript, объекта JSON или просто текста. Полученный ответ обрабатывается в веб-браузере с помощью языка JavaScript, и на основе ответа обновляется структура текущего HTML-документа. Таким образом не происходит перезагрузка всей веб-страницы, а только лишь изменяется какой-либо ее фрагмент. Классический пример технологии AJAX демонстрирует поисковая система Google, позволяющая при наборе поисковой фразы в строке запроса предлагать различные варианты, начинающиеся с набранных букв (рис. 10.1).

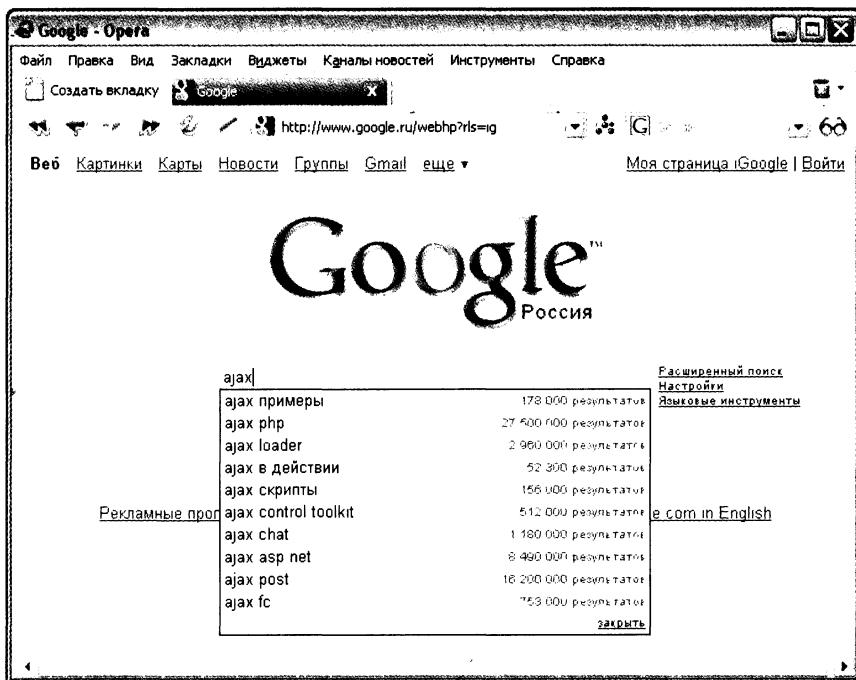


Рис. 10.1. Пример использования технологии AJAX

Но прежде чем изучать основы этой замечательной технологии, рассмотрим альтернативный способ общения веб-браузера с сервером без перезагрузки веб-страницы.

10.1. Обмен данными с помощью тега <iframe>

После заполнения формы и щелчка на кнопке **Submit** данные будут отправлены серверу, и ответ сервера полностью заменит текущую веб-страницу. Однако если в параметре `target` тега `<form>` указать название фрейма, данные формы будут загружены в этот фрейм, а текущая веб-страница останется без изменений. На этом и основан альтернативный способ обмена данными без перезагрузки веб-страницы.

Для создания фрейма используется тег `<iframe>` с нулевой высотой и шириной. Более того, обычно фрейм полностью скрывают от глаз пользователей, присваивая значение `none` атрибуту `display`. Название этого фрейма указывается в параметре `target` тега `<form>`. После отправки формы скрипт на сервере должен вернуть фрагмент кода на языке JavaScript, который произведет изменения в текущей веб-странице.

Рассмотрим все на примере. Создадим документ с формой (листинг 10.1), который будет обмениваться данными с файлом `ajax.php` (листинг 10.2), расположенным на сервере.

Листинг 10.1. Использование тега <iframe>

```
<html>
<head>
<title>Использование тега &lt;iframe&gt;</title>
</head>
<body>
<form action="ajax.php" method="GET" target="frame_ajax">
<input type="text" name="text1" id="text1"><br>
<input type="submit" value="Отправить"><br>
</form>
<div id="otvet"></div>
<iframe name="frame_ajax" src="ajax.php" width="0"
height="0" style="display:none">
</body>
</html>
```

Листинг 10.2. Исходный код файла ajax.php

```
<?php
header('Content-Type: text/html; charset=utf-8');
if (isset($_GET['text1'])) {
    $msg = 'Данные - ' . $_GET['text1'];
    echo '<script type="text/javascript">', "\n";
    echo "<!--\n";
    echo 'parent.document.getElementById("text1").value = "";', "\n";
    echo 'parent.document.getElementById("otvet").innerHTML = "' ;
    echo htmlspecialchars($msg, ENT_COMPAT, 'UTF-8'), '"';
    echo "\n//-->\n";
    echo '</script>\n';
}
else {
    echo 'Данные не получены';
}
?>
```

После заполнения формы и щелчка на кнопке Отправить во фрейм с названием frame_ajax будет загружен фрагмент кода JavaScript (листинг 10.3), сформированный скриптом ajax.php.

Листинг 10.3. Исходный код, возвращаемый сервером

```
<script type="text/javascript">
<!--
parent.document.getElementById("text1").value = "";
parent.document.getElementById("otvet").innerHTML = "Данные - Стока";
//-->
</script>
```

Данный скрипт очищает поле в форме и выводит полученные данные внутри тега <div> с идентификатором otvet. Для получения ссылки на родительское окно используется свойство parent объекта window. Вместо свойства parent можно воспользоваться свойством top, которое возвращает ссылку на самое верхнее родительское окно.

```
top.document.getElementById("text1").value = "";
top.document.getElementById("otvet").innerHTML = "Данные - Стока";
```

Получить ссылку на элемент HTML-документа по его идентификатору позволяет метод getElementById() объекта document. После получения ссылки мы можем изменить значение элемента. Для изменения данных в текстовом поле достаточно изменить значение свойства value. Изменить содержимое тега <div> можно с помощью свойства innerHTML, которое не только выводит текстовую информацию, но и обновляет объектную модель HTML-документа.

На самом деле использование тега <iframe> не является единственной альтернативой технологии AJAX. Вместо этого тега можно динамически изменять параметр src тега <script>, передавая параметры в URL-адресе. В ответ скрипт на сервере также должен сформировать фрагмент кода JavaScript, который изменит текущую веб-страницу без перезагрузки.

Все рассмотренные нами способы позволяют возвращать результат только в виде фрагмента кода JavaScript. Возможности технологии AJAX гораздо шире. Мы можем получить ответ в виде XML-документа, фрагмента HTML-документа, скрипта JavaScript, объекта JSON или просто текста. Передать запрос на сервер можно как методом GET, так и методом POST. Рассмотрим технологию AJAX более подробно.

10.2. Объект XMLHttpRequest

Для реализации обмена данными между веб-браузером и сервером используется объект XMLHttpRequest. Этот объект в фоновом режиме может отправить запрос и получить результат, который не отображается в окне веб-браузера. Получить доступ к результату запроса можно с помощью языка JavaScript.

Прежде чем использовать объект XMLHttpRequest, необходимо его создать с помощью оператора new. Проблема заключается в том, что в разных веб-браузерах объект создается по-разному. В одних используется встроенный объект window.XMLHttpRequest, а в других — ActiveX-объект MSXML2.XMLHTTP или Microsoft.XMLHTTP. Кросбраузерный код создания объекта приведен в листинге 10.4.

Листинг 10.4. Создание объекта XMLHttpRequest

```
if (window.XMLHttpRequest) {
    link = new XMLHttpRequest();
}
else {
    if (window.ActiveXObject) {
        var XMLHTTP = ["MSXML2.XMLHTTP.6.0", "MSXML2.XMLHTTP.3.0",
                      "Microsoft.XMLHTTP"];
        for (var i=0; i<4; i++) {
            try { // Обработка исключений
                link = new ActiveXObject(XMLHTTP[i]);
                break; // Если объект создан, то выход из цикла
            }
            catch (e) {}
        }
    }
}
```

Объект XMLHttpRequest имеет следующие методы.

- `open(<Метод передачи данных>, <URL-адрес>, <Тип запроса>)` — подготавливает запрос к отправке на сервер. В первом параметре можно указать метод GET или POST. Параметр `<URL-адрес>` предназначен для указания URL-адреса скрипта, которому будет отправлен запрос. Можно указать как абсолютный путь, так и относительный. При указании абсолютного пути следует иметь в виду, что запрос может быть отправлен только к домену, с которого был загружен текущий документ. Кроме того, некоторые веб-браузеры не позволяют отправить запрос к домену `http://www.site.ru/`, если документ был загружен с `http://site.ru/`. По этой причине лучше указывать относительный путь от корня сервера или от местонахождения текущего документа. Если для отправки данных используется метод GET, то после названия скрипта указываются передаваемые параметры. В параметре `<Тип запроса>` могут быть заданы следующие значения:
 - ◊ `true` — асинхронный запрос (выполнение текущего скрипта не приостанавливается до получения ответа);
 - ◊ `false` — синхронный запрос (выполнение текущего скрипта будет продолжено только после получения ответа сервера).

В большинстве случаев используются именно асинхронные запросы, так как ответ от сервера может занять некоторое время. Метод `open()` имеет еще два необязательных параметра: имя пользователя и пароль. В практике эти параметры не применяются, и мы их рассматривать не будем.

- `setRequestHeader(<Заголовок>, <Значение>)` — позволяет добавить заголовок запроса. В большинстве случаев используется для указания типа передаваемых данных методом POST.

```
link.setRequestHeader('Content-Type',
                     'application/x-www-form-urlencoded');
```

- `send(<Данные для отправки методом POST>)` — отправляет запрос на сервер. В качестве параметра при запросе методом GET указывается значение `null`.

```
link.send(null);
```

Для метода POST указывается строка передаваемых данных. Передаваемые данные должны быть перекодированы с помощью функции encodeURIComponent().

```
param = "text1=" + encodeURIComponent("Текст");  
link.send(param);
```

Рассмотрим свойства объекта XMLHttpRequest.

- onreadystatechange — в этом свойстве указывается ссылка на функцию-обработчик при асинхронных запросах.

```
link.onreadystatechange = f_razborRequest;
```

Обратите внимание на тот факт, что название функции указывается без круглых скобок и параметров. Если указать круглые скобки, то смысл происходящего будет другим. Функция будет выполнена, и результат ее работы будет присвоен свойству onreadystatechange. Так как в функцию-обработчик невозможно передать параметр, то внутри функции мы можем обратиться к объекту XMLHttpRequest, только если он будет объявлен в глобальной области видимости. Это не всегда удобно. Однако передать параметр можно, указав в качестве обработчика анонимную функцию, внутри которой происходит вызов функции с параметром.

```
link.onreadystatechange = function() {  
    f_razborRequest(link);  
};
```

Этим способом мы и будем пользоваться в дальнейших примерах. Функция f_razborRequest() будет вызываться несколько раз при каждом изменении статуса обработки запроса.

- readyState — содержит текущий статус обработки запроса. Может принимать следующие значения:

- ◊ 0 — объект запроса создан оператором new;
- ◊ 1 — запрос подготовлен с помощью метода open();
- ◊ 2 — запрос отправлен методом send();
- ◊ 3 — ответ доступен, но загружен не полностью;
- ◊ 4 — ответ полностью загружен с сервера и доступен для обработки.

В дальнейшем нас будет интересовать именно статус 4.

- status — содержит код возврата при значении свойства readyState, равном 4. Успешными считаются коды в пределах от 200 до 299, а также код 304, указывающий, что документ не был изменен с последнего запроса.

Получить доступ к ответу сервера можно с помощью двух свойств.

- responseText — ответ в виде текста, фрагмента HTML-документа или объекта JSON;
- responseXML — ответ в виде XML-документа. Свойство будет доступно, только если в заголовках ответа сервера указан заголовок Content-Type: application/xml

или

Content-Type: text/xml

Кроме того, XML-документ должен быть иметь правильную структуру и соответствовать спецификации DOM.

Итак, как вы уже знаете, ответ сервера может быть предоставлен в виде:

- фрагмента HTML-документа, скрипта JavaScript или просто текста;
- XML-документа;
- объекта JSON.

Рассмотрим каждый вариант обмена данными более подробно.

10.3. Обмен данными в текстовом формате

Обмен данными с помощью фрагментов HTML-страницы, фрагментов кода JavaScript или текста является самым простым способом получения результата. Ответ сервера будет сохранен в свойстве `responseText` объекта `XMLHttpRequest`. Для отображения результата запроса на веб-странице достаточно присвоить значение свойства `responseText` свойству `innerHTML` какого-либо существующего HTML-элемента. На основании вставленного фрагмента веб-браузер обновит объектную модель HTML-документа и изменит внешний вид веб-страницы без перегрузки страницы.

Рассмотрим обмен данными в текстовом формате на примере. Создадим документ с формой (листинг 10.5), который будет обмениваться данными с файлом `ajax.php` (листинг 10.6), расположенным на сервере, методами GET и POST.

Листинг 10.5. Обмен данными в текстовом формате

```
<html>
<head>
<title>Обмен данными в текстовом формате</title>
<script type="text/javascript">
<!--
function f_GET() {
    // Отправка запроса методом GET
    var link;
    var url;
    var txt = document.getElementById("text1").value;
    if (txt == "") {
        window.alert("Не заполнено поле");
        return false;
    }
    if (window.XMLHttpRequest) {
        link = new XMLHttpRequest();
    }
    else {
        if (window.ActiveXObject) {
            var XMLHTTP = ["MSXML2.XMLHTTP.6.0", "MSXML2.XMLHTTP.3.0",
                           "MSXML2.XMLHTTP", "Microsoft.XMLHTTP"];
            for (var i=0; i<4; i++) {
                try { // Обработка исключений
                    link = new ActiveXObject(XMLHTTP[i]);
                }
                catch (e) {}
            }
        }
    }
    link.open("GET", url);
    link.onreadystatechange = function() {
        if (link.readyState == 4) {
            document.getElementById("text2").value = link.responseText;
        }
    }
    link.send(null);
}
--></script>
<body>
<form>
    <input type="text" id="text1" value="" />
    <input type="button" value="Получить данные" onclick="f_GET()" />
    <input type="text" id="text2" value="" />
</form>
</body>

```

```

        break; // Если объект создан, то выход из цикла
    }
    catch (e) {}
}
}
if (!link) {
    window.alert("Ваш веб-браузер не поддерживает технологию Ajax");
    return false;
}
url = "/ajax.php?text1=" + encodeURIComponent(txt);
// Подготовка асинхронного запроса методом GET
link.open('GET', url, true);
link.onreadystatechange = function() {
    f_razborRequest(link);
};
link.send(null); // Отправляем запрос
document.getElementById("otvet").innerHTML = "Загрузка...";
}

function f_POST() {
// Отправка запроса методом POST
var link;
var param;
var txt = document.getElementById("text1").value;
if (txt == "") {
    window.alert("Не заполнено поле");
    return false;
}
if (window.XMLHttpRequest) {
    link = new XMLHttpRequest();
}
else {
    if (window.ActiveXObject) {
        var XMLHTTP = ["MSXML2.XMLHTTP.6.0", "MSXML2.XMLHTTP.3.0",
                      "MSXML2.XMLHTTP", "Microsoft.XMLHTTP"];
        for (var i=0; i<4; i++) {
            try { // Обработка исключений
                link = new ActiveXObject(XMLHTTP[i]);
                break; // Если объект создан, то выход из цикла
            }
            catch (e) {}
        }
    }
}
if (!link) {
    window.alert("Ваш веб-браузер не поддерживает технологию Ajax");
    return false;
}
// Передаваемые параметры
param = "text1=" + encodeURIComponent(txt);
// Подготовка асинхронного запроса методом POST
link.open('POST', '/ajax.php', true);
// Добавляем HTTP-заголовок
}

```

```

link.setRequestHeader('Content-Type',
    'application/x-www-form-urlencoded');
link.onreadystatechange = function() {
    f_razborRequest(link);
};
link.send(param); // Отправляем запрос
document.getElementById("otvet").innerHTML = "Загрузка...";
}

function f_razborRequest(link) {
    // Обрабатываем асинхронный запрос
    if (link.readyState == 4) {
        if (link.status == 200) { // Запрос успешно обработан
            var otvet = link.responseText;
            document.getElementById("text1").value = "";
            document.getElementById("otvet").innerHTML = otvet;
        }
        else {
            document.getElementById("otvet").innerHTML = "Ошибка";
        }
    }
}
//-->
</script>
</head>
<body>
<input type="text" id="text1"><br>
<input type="button" value="Отправить методом GET" onclick="f_GET();">
<br>
<input type="button" value="Отправить методом POST"
onclick="f_POST();">
<div id="otvet"></div>
</body>
</html>

```

Листинг 10.6. Исходный код файла ajax.php

```

<?php
header('Content-Type: text/html; charset=utf-8');
if (isset($_GET['text1'])) {
    echo 'Метод GET - ', $_GET['text1'];
}
else {
    if (isset($_POST['text1'])) {
        echo 'Метод POST - ', $_POST['text1'];
    }
    else {
        echo 'Данные не получены';
    }
}
?>

```

После загрузки кода из листинга 10.5 на веб-странице будет отображено поле для ввода и две кнопки. При щелчке на кнопке **Отправить методом GET** будет вызвана

функция `f_GET()`, а при щелчке на кнопке Отправить методом POST — функция `f_POST()`. Результат выполнения запроса будет выведен в теге `<div>`, имеющем идентификатор `otvet` (`id="otvet"`).

После заполнения формы и щелчка на кнопке Отправить методом GET внутри функции `f_GET()` получаем значение текстового поля и сохраняем его в переменной `txt`.

```
var txt = document.getElementById("text1").value;
```

Далее проверяем, заполнено ли поле ввода, и, если оно не заполнено, выводим сообщение и завершаем выполнение функции.

```
if (txt == "") {  
    window.alert("Не заполнено поле");  
    return false;  
}
```

На следующем этапе пытаемся создать объект `XMLHttpRequest`. Если объект создать не удалось, выводим сообщение и завершаем выполнение функции.

```
if (!link) {  
    window.alert("Ваш веб-браузер не поддерживает технологию Ajax");  
    return false;  
}
```

Затем формируем строку запроса, которую будем передавать методом GET. Все данные, введенные пользователем в поле ввода, перекодируем с помощью метода `encodeURIComponent()`. Сформированную строку сохраняем в переменной `url`.

```
url = "/ajax.php?text1=" + encodeURIComponent(txt);
```

Далее с помощью метода `open()` подготавливаем объект к отправке асинхронного запроса методом GET.

```
link.open('GET', url, true);
```

В свойстве `onreadystatechange` указываем анонимную функцию, в которой вызывается функция `f_razborRequest()`. Благодаря этому можем передать объект в качестве параметра функции. Функция `f_razborRequest()` будет вызываться при каждом изменении статуса запроса.

```
link.onreadystatechange = function() {  
    f_razborRequest(link);  
};
```

С помощью метода `send()` отправляем запрос на сервер. Так как мы передаем данные методом POST, в качестве параметра метода `send()` указываем значение `null`.

```
link.send(null);
```

Поскольку поступление ответа от сервера может произойти через некоторое (иногда продолжительное) время, выводим сообщение, подтверждающее отправку данных.

```
document.getElementById("otvet").innerHTML = "Загрузка...";
```

При каждом изменении статуса запроса будет вызываться функция `f_razborRequest()`. Внутри этой функции проверяем значение свойства `readyState`. Если статус запроса равен 4, то ответ полностью получен с сервера и доступен для обработки.

```
if (link.readyState == 4) {  
    ...  
}
```

Наличие статуса завершения еще не гарантирует успешность выполнения запроса. Для этого мы проверяем значение свойства `status`. Если значение равно 200, то запрос выполнен успешно. В этом случае получаем ответ сервера с помощью свойства `resonseText`, очищаем поле ввода и присваиваем значение элементу с идентификатором `otvet` с помощью свойства `innerHTML`.

```
if (link.status == 200) {  
    var otvet = link.responseText;  
    document.getElementById("text1").value = "";  
    document.getElementById("otvet").innerHTML = otvet;  
}
```

В случае заполнения формы и последующего щелчка на кнопке Отправить методом `POST` внутри функции `f_POST()` мы получаем значение текстового поля и сохраняем его в переменной `txt`. Далее проверяем, заполнено ли поле ввода, и, если оно не заполнено, выводим сообщение и завершаем выполнение функции. На следующем этапе пытаемся создать объект `XMLHttpRequest`. Если объект создать не удалось, выводим сообщение и завершаем выполнение функции. Затем формируем строку запроса, которую будем передавать методом `POST`. Все данные, введенные пользователем в поле ввода, перекодируем с помощью метода `encodeURIComponent()`. Сформированную строку сохраняем в переменной `param`.

```
param = "text1=" + encodeURIComponent(txt);
```

Далее с помощью метода `open()` подготавливаем объект к отправке асинхронного запроса методом `POST`.

```
link.open('POST', '/ajax.php', true);
```

В первом параметре указываем метод `POST`, а во втором — передаем путь к скрипту относительно корня сервера. Значение третьего параметра (`true`) указывает, что запрос будет асинхронным.

При передаче данных методом `POST` необходимо дополнительно указать, что данные переданы из формы. Для этого в методе `setRequestHeader` указываем значение `application/x-www-form-urlencoded`.

```
link.setRequestHeader('Content-Type',  
    'application/x-www-form-urlencoded');
```

Далее, как обычно, в свойстве `onreadystatechange` указываем функцию-обработчик. Затем с помощью метода `send()` отправляем запрос на сервер. В качестве параметра указываем сформированную строку запроса, которую мы сохранили в переменной `param`.

```
link.send(param);
```

Дальнейшая обработка запроса происходит точно так же, как и при отправке запроса методом `GET`.

Теперь рассмотрим, что происходит при получении запроса сервером. Получив запрос, сервер создаст следующие переменные окружения:

- при передаче методом `GET`:
`$_GET['text1']`
- при передаче методом `POST`:
`$_POST['text1']`

Проверить существование переменных можно с помощью функции `isset()`. После получения переменных можем делать с ними все, что захотим. В этом примере мы просто возвращаем отправленное значение обратно веб-браузеру, дополнительно указывая, каким методом были получены данные.

Для корректного отображения отправленной информации указывается MIME-тип и кодировка с помощью функции `header()`.

```
header('Content-Type: text/html; charset=utf-8');
```

Если кодировка не указана сервером, то веб-браузер по умолчанию предполагает, что ответ пришел в кодировке UTF-8. Так что если данные посылаются в другой кодировке, то ее необходимо обязательно указывать.

10.4. Обмен данными в формате XML

С одной стороны, обмен данными в текстовом формате является простым, но с другой стороны, это влечет за собой жесткую зависимость между клиентским и серверным кодами. Если разработкой проекта занимаются разные люди, а обычно так и бывает, то такой способ обмена данными не является эффективным. При обмене данными в формате XML разработчики могут заранее договориться о формате XML-документа, и в дальнейшем каждый будет выполнять свою часть работы, опираясь на согласованный формат.

Прежде чем рассматривать обработку данных в формате XML с помощью JavaScript, напомним некоторые правила создания простейших XML-документов. XML-документ состоит из двух разделов — пролога и тела документа. В прологе указывается объявление XML.

```
<?xml version="1.0" encoding="utf-8" ?>
```

В параметре `encoding` указывается кодировка XML-документа. Если она не указана, то по умолчанию используется кодировка UTF-8.

Тело XML-документа должно находиться внутри корневого тега. Обычно имя тега совпадает с именем документа.

```
<?xml version="1.0" encoding="utf-8" ?>
<ajax>
<!-- Тело XML-документа -->
</ajax>
```

При назначении имени тега необходимо помнить, что регистр символов имеет значение. Имена должны начинаться с буквы и содержать буквы, цифры или символ подчеркивания. В отличие от HTML-тегов, в XML пространство имен выбирает сам разработчик.

Следует учитывать, что каждому открывающему тегу должен соответствовать закрывающий тег.

```
<item></item>
```

Кроме того, должна соблюдаться вложенность тегов. Если элемент является пустым, то он может быть записан следующим образом.

```
<item/>
```

XML-теги могут иметь параметры. Параметр указывается в формате:

```
<Имя параметра>=<Значение>
```

Значение параметра обязательно должно быть указано в кавычках, а сам параметр указывается в открывающем теге.

```
<item id="5">Данные</item>
```

Все специальные символы внутри тегов и в значениях параметров должны быть заменены на мнемонические имена.

- < — знак меньше (<)
- > — знак больше (>)
- & — амперсанд (&)
- " — кавычка ("")
- ' — апостроф ('')

Внутри тегов можно использовать комментарии, которые имеют синтаксис, похожий на синтаксис HTML-комментариев.

```
<!-- Это комментарий -->
```

Следование этим несложным правилам избавит от множества проблем при работе с XML-документами.

Теперь рассмотрим обмен данными в формате XML на примере. Создадим документ с формой (листинг 10.7), который будет обмениваться данными с файлом ajax.php (листинг 10.8), расположенным на сервере.

Листинг 10.7. Обмен данными в формате XML

```
<html>
<head>
<title>Обмен данными в формате XML</title>
<script type="text/javascript">
<!--
function f_GET() {
    // Отправка запроса методом GET
    var link;
    var url;
    var txt1 = document.getElementById("text1").value;
    var txt2 = document.getElementById("text2").value;
    if (txt1 == "" || txt2 == "") {
        window.alert("Не заполнено поле");
        return false;
    }
    if (window.XMLHttpRequest) {
        link = new XMLHttpRequest();
    }
    else {
        if (window.ActiveXObject) {
            var XMLHTTP = ["MSXML2.XMLHTTP.6.0", "MSXML2.XMLHTTP.3.0",
                           "MSXML2.XMLHTTP", "Microsoft.XMLHTTP"];
            for (var i=0; i<4; i++) {
                try { // Обработка исключений
                    link = new ActiveXObject(XMLHTTP[i]);
                    break; // Если объект создан, то выход из цикла
                }
            }
        }
    }
}
```

```

        catch (e) {}
    }
}
if (!link) {
    window.alert("Ваш веб-браузер не поддерживает технологию Ajax");
    return false;
}
url = "/ajax.php?text1=" + encodeURIComponent(txt1);
url += "&text2=" + encodeURIComponent(txt2);
// Подготовка асинхронного запроса методом GET
link.open('GET', url, true);
link.onreadystatechange = function() {
    f_razborRequest(link);
};
link.send(null); // Отправляем запрос
document.getElementById("otvet").innerHTML = "Загрузка...";
}

function f_razborRequest(link) {
    // Обрабатываем асинхронный запрос
    if (link.readyState == 4) {
        if (link.status == 200) { // Запрос успешно обработан
            var xml = link.responseXML;
            var otvet = xml.getElementsByTagName("otvet");
            var text1 = otvet.item(0)
                .getElementsByTagName("text1").item(0);
            var text2 = otvet.item(0)
                .getElementsByTagName("text2").item(0);
            var msg = text1.firstChild.data + "<br>" +
                text2.firstChild.data;
            document.getElementById("text1").value = "";
            document.getElementById("text2").value = "";
            document.getElementById("otvet").innerHTML = msg;
        }
        else {
            document.getElementById("otvet").innerHTML = "Ошибка";
        }
    }
}
//-->
</script>
</head>
<body>
<input type="text" id="text1"><br>
<input type="text" id="text2"><br>
<input type="button" value="Отправить методом GET" onclick="f_GET();">
<br><div id="otvet"></div>
</body>
</html>

```

Листинг 10.8. Исходный код файла ajax.php

```
<?php
header('Content-Type: text/xml; charset=utf-8');
echo '<?xml version="1.0" encoding="utf-8" ?>', "\n";
echo "<ajax>\n";
echo "    <otvet>\n";
if (isset($_GET['text1']) && isset($_GET['text2'])) {
    echo "        <text1>";
    echo 'Поле 1 - ';
    echo htmlspecialchars($_GET['text1'], ENT_COMPAT, 'UTF-8');
    echo "</text1>\n";
    echo "        <text2>";
    echo 'Поле 2 - ';
    echo htmlspecialchars($_GET['text2'], ENT_COMPAT, 'UTF-8');
    echo "</text2>\n";
}
else {
    echo 'Данные не получены';
}
echo "    </otvet>\n";
echo "</ajax>\n";
?>
```

После заполнения двух полей и щелчка на кнопке Отправить методом GET внутри функции `f_GET()` получаем значение текстовых полей и сохраняем их в переменных `txt1` и `txt2`. Далее проверяем, заполнены ли поля ввода, и, если они не заполнены, выводим сообщение и завершаем выполнение функции. На следующем этапе пытаемся создать объект `XMLHttpRequest`. Если объект создать не удалось, выводим сообщение и завершаем выполнение функции.

Затем формируем строку запроса, которую будем передавать методом GET. Все данные, введенные пользователем в поля ввода, перекодируем с помощью метода `encodeURIComponent()`. Сформированную строку сохраняем в переменной `url`.

```
url = "/ajax.php?text1=" + encodeURIComponent(txt1);
url += "&text2=" + encodeURIComponent(txt2);
```

Далее с помощью метода `open()` подготавливаем объект к отправке асинхронного запроса методом GET.

```
link.open('GET', url, true);
```

В свойстве `onreadystatechange` указываем функцию-обработчик и с помощью метода `send()` отправляем запрос на сервер.

```
link.send(null);
```

Теперь рассмотрим, что происходит при получении запроса сервером. Получив запрос, сервер создаст две переменные окружения: `$_GET['text1']` и `$_GET['text2']`. Для корректного отображения отправленной информации указываем MIME-тип и кодировку с помощью функции `header()`.

```
header('Content-Type: text/xml; charset=utf-8');
```

Далее формируем XML-документ, который отправим в ответ на запрос. В результате веб-браузер получит XML-документ, приведенный в листинге 10.9.

Листинг 10.9. XML-документ, формируемый в качестве ответа сервера

```
<?xml version="1.0" encoding="utf-8" ?>
<ajax>
  <ответ>
    <text1>Поле 1 - Стока 1</text1>
    <text2>Поле 2 - Стока 2</text2>
  </ответ>
</ajax>
```

После получения ответа XML-документ в веб-браузере будет доступен через свойство `responseXML` внутри функции `f_razborRequest()`.

```
var xml = link.responseXML;
```

С помощью метода `getElementsByTagName()` получаем список всех элементов `ответ` и сохраняем его в переменной `ответ`.

```
var ответ = xml.getElementsByTagName("ответ");
```

Получить доступ к отдельному элементу в списке позволяет метод `item()`. Нумерация элементов начинается с нуля. Общее количество элементов можно получить с помощью свойства `length`.

```
var count = ответ.length;
```

Доступ к элементам `text1` и `text2` внутри элемента `ответ` также происходит с помощью метода `getElementsByTagName()`.

```
var text1 = ответ.item(0).getElementsByTagName("text1").item(0);
var text2 = ответ.item(0).getElementsByTagName("text2").item(0);
```

Получить доступ к элементу можно по индексу, указанному в квадратных скобках.

```
var text1 = ответ[0].getElementsByTagName("text1")[0];
var text2 = ответ[0].getElementsByTagName("text2")[0];
```

Чтобы получить данные внутри элементов `text1` и `text2`, следует воспользоваться свойством `firstChild`. Свойство `firstChild` содержит информацию о длине строки (свойство `length`) и текстовых данных элемента (свойство `data`). Именно свойство `data` позволяет нам получить переданные данные и сформировать строку для вывода результата.

```
var msg = text1.firstChild.data + "<br>" + text2.firstChild.data;
document.getElementById("ответ").innerHTML = msg;
```

10.5. Обмен данными в формате JSON

Хорошей альтернативой использования XML-документа для обмена данными с сервером служит применение JSON-объектов. XML-документ из листинга 10.9 можно представить в качестве JSON-объекта следующим образом.

```
var json = '{' +
'"text1":"Поле 1 - Стока 1",' +
'"text2":"Поле 2 - Стока 2"' +
'}';
```

Чтобы преобразовать строку `json` в JSON-объект, достаточно обработать ее с помощью функции `eval()`. Функция `eval()` помещает JSON-объект в память интерпретатора.

ра JavaScript и возвращает ссылку на него. Для корректной работы функции eval() строку, содержащую JSON-объект, необходимо взять в круглые скобки.

```
var jsonObject = eval('(' + json + ')');
```

Получить доступ к данным можно следующим образом.

```
var text1 = jsonObject.text1;
var text2 = jsonObject.text2;
```

В качестве JSON-объекта может быть указан массив с данными.

```
var json = '["Элемент1", "Элемент2", "Элемент3", "Элемент4"]';
var jsonObject = eval('(' + json + ')');
alert(jsonObject[0]); // Выведет: Элемент1
```

JSON-объект может иметь сложную структуру. Например, он может быть массивом объектов.

```
var json = '[ {"text1": "Поле 1", "text2": "Поле 2"}, ';
json += '{ "text3": "Поле 3", "text4": "Поле 4" } ]';
var jsonObject = eval('(' + json + ')');
alert(jsonObject[1].text3); // Выведет: Поле 3
```

Или объектом, значения свойств которого являются массивами.

```
var json = '{ "elem1": [1, 2, 3], "elem2": [4, 5, 6] }';
var jsonObject = eval('(' + json + ')');
alert(jsonObject.elem2[0]); // Выведет: 4
```

Можно также комбинировать различные варианты.

```
var json = '{ "elem1": [{ "txt1": "Текст1", "txt2": "Текст2" }, 2] }';
var jsonObject = eval('(' + json + ')');
alert(jsonObject.elem1[0].txt2); // Выведет: Текст2
```

Приведем пример обмена данными в формате JSON. Создадим документ с формой (листинг 10.10), который будет обмениваться данными с файлом ajax.php (листинг 10.11), расположенным на сервере.

Листинг 10.10. Обмен данными в формате JSON

```
<html>
<head>
<title>Обмен данными в формате JSON</title>
<script type="text/javascript">
<!--
function f_GET() {
    // Отправка запроса методом GET
    var link;
    var url;
    var txt1 = document.getElementById("text1").value;
    var txt2 = document.getElementById("text2").value;
    if (txt1 == "" || txt2 == "") {
        window.alert("Не заполнено поле");
        return false;
    }
    if (window.XMLHttpRequest) {
        link = new XMLHttpRequest();
    }
}
```

```

else {
    if (window.ActiveXObject) {
        var XMLHTTP = ["MSXML2.XMLHTTP.6.0", "MSXML2.XMLHTTP.3.0",
                      "MSXML2.XMLHTTP", "Microsoft.XMLHTTP"];
        for (var i=0; i<4; i++) {
            try { // Обработка исключений
                link = new ActiveXObject(XMLHTTP[i]);
                break; // Если объект создан, то выход из цикла
            }
            catch (e) {}
        }
    }
}
if (!link) {
    window.alert("Ваш веб-браузер не поддерживает технологию Ajax");
    return false;
}
url = "/ajax.php?text1=" + encodeURIComponent(txt1);
url += "&text2=" + encodeURIComponent(txt2);
// Подготовка асинхронного запроса методом GET
link.open('GET', url, true);
link.onreadystatechange = function() {
    f_razborRequest(link);
};
link.send(null); // Отправляем запрос
document.getElementById("otvet").innerHTML = "Загрузка...";
}

function f_razborRequest(link) {
    // Обрабатываем асинхронный запрос
    if (link.readyState == 4) {
        if (link.status == 200) { // Запрос успешно обработан
            var json = link.responseText;
            var otvet = eval('(' + json + ')');
            var msg = otvet.text1 + "<br>" + otvet.text2;
            document.getElementById("text1").value = "";
            document.getElementById("text2").value = "";
            document.getElementById("otvet").innerHTML = msg;
        }
        else {
            document.getElementById("otvet").innerHTML = "Ошибка";
        }
    }
}
//-->
</script>
</head>
<body>
<input type="text" id="text1"><br>
<input type="text" id="text2"><br>
<input type="button" value="Отправить методом GET"
       onclick="f_GET();">
<br>
<div id="otvet"></div>

```

```
</body>
</html>
```

Листинг 10.11. Исходный код файла ajax.php

```
<?php
header('Content-Type: text/plain; charset=utf-8');
if (isset($_GET['text1']) && isset($_GET['text2'])) {
    $text1 = htmlspecialchars($_GET['text1'], ENT_COMPAT, 'UTF-8');
    $text2 = htmlspecialchars($_GET['text2'], ENT_COMPAT, 'UTF-8');
    $arr = array('text1' => $text1, 'text2' => $text2);
    echo json_encode($arr);
}
else {
    $arr = array('text1' => 'Данные не получены', 'text2' => '');
    echo json_encode($arr);
}
?>
```

Для работы с JSON-объектами в PHP существует две функции.

- `json_encode(<Ассоциативный массив>)` — преобразует ассоциативный массив в JSON-объект.
- `json_decode(<JSON-объект>[, <Тип результата>])` — производит разбор JSON-объекта. Необязательный параметр `<Тип результата>` может принимать следующие значения:
 - ◊ `true` — результат в виде ассоциативного массива;
 - ◊ `false` — результат в виде объекта (значение по умолчанию).

Рассмотрим пример.

```
<?php
$arr['text1'] = 'Строка 1';
$arr['text2'] = "Строка 2";
$json = json_encode($arr);
$arr2 = json_decode($json);
echo $arr2->text1, '<br>'; // Страна 1
echo $arr2->text2, '<br>'; // Страна 2
$arr3 = json_decode($json, true);
echo $arr3['text1'], '<br>'; // Страна 1
echo $arr3['text2']; // Страна 2
?>
```

Глава 11

Поддержка AJAX в jQuery

В библиотеке jQuery технология AJAX также не оставлена без внимания — здесь предполагается, пожалуй, самый удобный интерфейс доступа к ней. Реализована поддержка всех подвидов технологии AJAX, а также альтернативный вариант общения браузера с сервером через динамически создаваемый тег `<script>`. Этот вариант позволяет получать данные не только с того же домена, но и с любого другого при соблюдении некоторых условий. Получить с другого домена можно код JavaScript, а также объект в формате JSONP.

При отправке запроса на сервер добавляется заголовок `x-requested-with` со значением `XMLHttpRequest`.

`x-requested-with: XMLHttpRequest`

Благодаря этому заголовку на сервере можно определить, откуда поступил запрос. Узнать, что запрос сделан с помощью AJAX, можно так.

```
if (isset($_SERVER['HTTP_X_REQUESTED_WITH']))  
    && $_SERVER['HTTP_X_REQUESTED_WITH'] == 'XMLHttpRequest') {  
    // Запрос выполнен с помощью AJAX  
}
```

11.1. Метод `load()`

Метод `load()` позволяет загрузить данные с сервера в определенный элемент коллекции jQuery. После вставки данных объектная модель документа будет обновлена. Формат метода:

`load(<URL> [<Селектор>] [, <Данные>] [, <Функция обратного вызова>])`

Загрузить данные можно только с того же домена. Если указан только первый параметр, то запрос будет сделан методом GET.

```
$("#div1").load("/ajax.php?id=1");
```

Получить отправленные данные на сервере можно с помощью глобального массива `$_GET`.

```
if (isset($_GET['id'])) $id = $_GET['id'];
```

В необязательном параметре `<Селектор>` можно указать селектор, который ограничит набор вставляемых данных. Например, при таком содержимом файла `ajax.php`:

```
<?php  
header('Content-Type: text/html; charset=utf-8');  
?  
<span id="span1">Данные 1</span>  
<span id="span2">Данные 2</span>  
  
выражение  
$("#div1").load("/ajax.php");
```

загрузит в элемент с идентификатором div1 следующие данные:

```
<span id="span1">Данные 1</span>
<span id="span2">Данные 2</span>
```

Если указать выражение

```
$("#div1").load("/ajax.php #span2");
```

то вставляемые данные будут ограничены элементом с идентификатором span2, и мы получим следующие данные:

```
<span id="span2">Данные 2</span>
```

Параметр <Данные> позволяет передать несколько параметров. Для этого параметры и значения должны быть указаны следующим образом.

```
{
Параметр1: "Значение1",
Параметр2: "Значение2",
...
ПараметрN: "ЗначениеN"
}
```

Если параметр <Данные> указан, то данные передаются методом POST. В случае если параметры или значения содержат запрещенные символы, то они будут автоматически перекодированы с помощью метода encodeURIComponent().

```
$("#div1").load("/ajax.php", { txt: "Текст" });
```

Получить отправленные данные на сервере можно с помощью глобального массива \$_POST.

```
if (isset($_POST['txt'])) $txt = $_POST['txt'];
```

В качестве параметра <Функция обратного вызова> указывается ссылка на функцию следующего формата.

```
function <Название функции>(<Загруженные данные>[, <Статус>[,  
    <Объект XMLHttpRequest>]]) {  
    // ...  
}
```

Функция будет вызвана после окончания загрузки, независимо от ее результата. Внутри функции доступна ссылка (*this*) на текущий DOM-элемент. Если в первом параметре указать переменную, то через нее будут доступны данные, которые загружены с сервера. Через параметр <Статус> доступен статус запроса. Возвращаются следующие значения:

- error — при ошибке;
- success — при успешном выполнении запроса.

Обработать ошибку загрузки можно, например, так.

```
$("#div1").load("/ajax.php", { txt: "Текст" },
    function(data, status) {
        if (status == 'error') {
            $(this).html("Ошибка при загрузке");
        }
    });
});
```

Через параметр <Объект XMLHttpRequest> доступна ссылка на объект XMLHttpRequest. Обработать ошибку загрузки с помощью этого параметра можно, например, так.

```
$("#div1").load("/ajax2.php", { txt: "Текст" },
  function(data, status, XMLHttpRequest) {
    if (XMLHttpRequest.status != 200) {
      $(this).html("Ошибка при загрузке");
    }
  });
});
```

В качестве примера использования метода load() рассмотрим вывод каталога фильмов. Изначально из базы данных MySQL выводятся только названия фильмов. После щелчка на ссылке Показать с помощью технологии AJAX с сервера будет загружено полное описание фильма и отображено в элементе DIV. Изначально элемент является скрытым. После загрузки описаний мы отображаем этот элемент и изменяем текст ссылки на Скрыть. Если щелкнуть мышью на этой ссылке, то описание будет скрыто, а текст ссылки изменится на Показать. Повторный щелчок на ссылке Показать приводит к отображению описания без повторной загрузки. SQL-запрос для создания таблицы представлен в листинге 11.1. Содержимое файла index.php (основной файл) представлено в листинге 11.2, а текст файла ajax.php (для вывода полного описания фильма) содержится в листинге 11.3.

Листинг 11.1. SQL-запрос для создания таблицы

```
CREATE TABLE `films` (
  `id` int(11) auto_increment,
  `name` varchar(255) character set `utf8` collate `utf8_general_ci`,
  `descr` text character set `utf8` collate `utf8_general_ci`,
  PRIMARY KEY (`id`)
) engine=MYISAM character set `utf8` collate `utf8_general_ci`;
INSERT INTO `films` (`id`, `name`, `descr`) VALUES
(1, 'Фильм 1', 'Описание фильма 1'),
(2, 'Фильм 2', 'Описание фильма 2'),
(3, 'Фильм 3', 'Описание фильма 3');
```

Листинг 11.2. Содержимое файла index.php

```
<?php
// Запрещаем кеширование
header('Expires: Sun, 27 May 2007 01:00:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-store, no-cache, must-revalidate');
header('Pragma: no-cache');
// Указываем кодировку
header('Content-Type: text/html; charset=utf-8');
// Подключаемся к базе
$db = mysql_connect('localhost', 'root', '123456')
  or die('Не удалось установить подключение!');
mysql_select_db('test') or die('Такой базы данных нет');
mysql_query('SET NAMES `utf8`'); // Кодировка соединения
?>
<html>
<head>
<title>Метод load()</title>
```

```

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
function f_show(id, elem) {
    var div = $("#div" + id);
    var link = $(elem);
    if (div.is(":hidden")) {
        if (div.text() == "") {
            div.load("/ajax.php", { "id": id },
                    function(responseText, textStatus) {
                        if (textStatus == 'error') {
                            div.html("Ошибка при загрузке");
                        }
                        div.slideDown(300);
                        link.html("Скрыть");
                    });
        }
        else {
            div.slideDown(300);
            link.html("Скрыть");
        }
    }
    else {
        div.slideUp(300);
        link.html("Показать");
    }
}
//--
</script>
</head>
<body>
<?php
$query = 'SELECT `id`, `name` FROM `films` ORDER BY `name`';
if ($res = mysql_query($query)) {
    while ($row = mysql_fetch_row($res)) {
        echo '<a href="#" onclick="f_show(' . $row[0] . ', this); ';
        echo 'return false;">Показать</a> <b>';
        echo htmlspecialchars($row[1], ENT_COMPAT, 'UTF-8');
        echo "</b><br>\n";
        echo '<div id="div' . $row[0] . '" style="display:none;">';
        echo "</div>\n";
    }
}
mysql_close($db);
?>
</body>
</html>

```

Листинг 11.3. Содержимое файла ajax.php

```

<?php
// Указываем кодировку
header('Content-Type: text/html; charset=utf-8');
if (!isset($_SERVER['HTTP_X_REQUESTED_WITH'])) {

```

```

    exit('Данные отправлены не через AJAX');
}
if (isset($_POST['id'])) $id = (int)$_POST['id'];
else $id = 0;
if ($id === 0) exit('Параметр id содержит ошибку');
// Подключаемся к базе
$db = mysql_connect('localhost', 'root', '123456')
    or die('Не удалось установить подключение');
mysql_select_db('test') or die('Такой базы данных нет');
mysql_query('SET NAMES `utf8`'); // Кодировка соединения
$query = 'SELECT `descr` FROM `films` WHERE `id`=' . $id;
if ($res = mysql_query($query)) {
    if (mysql_num_rows($res) != 0) {
        $row = mysql_fetch_row($res);
        echo $row[0]; // Выводим описание
    }
    else {
        echo 'Фильм не найден';
    }
}
else {
    echo 'Ошибка при выполнении запроса';
}
mysql_close($db);
?>

```

11.2. Функция \$.getJSON()

Функция `$.getJSON()` позволяет получить данные в форматах JSON и JSONP методом GET. Формат функции:

```
$.getJSON(<URL>[, <Данные>] [, <Функция обратного вызова>])
```

Следует учитывать, что при использовании формата JSON данные можно загрузить только с того же домена. Необязательный параметр `<Данные>` позволяет передать несколько параметров. Для этого параметры и значения должны быть указаны следующим образом.

```
{
Параметр1: "Значениe1",
Параметр2: "Значениe2",
...
ПараметрN: "ЗначениeN"
}
```

В качестве параметра `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>(<JSON-объект>[, <Статус>]) {
    // ...
}
```

Внутри функции обратного вызова доступна ссылка (`this`) на объект с параметрами запроса. После получения данных они автоматически выполняются с помощью функции `eval()`, и JSON-объект становится доступным через первый параметр. С помощью параметра `<Статус>` можно получить статус запроса (значение `success` — при успешном выполнении).

В качестве примера рассмотрим получение данных в формате JSON. Создадим файл index.php (листинг 11.4), который будет обмениваться данными с файлом ajax.php (листинг 11.5), расположенным на том же домене.

Листинг 11.4. Содержимое файла index.php

```
<html>
<head>
<title>Функция $.getJSON() и JSON</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $(":button").click(function() {
        $.getJSON("/ajax.php", { txt: "test" },
            function(jsonObject) {
                $("#div1").html("param1 = " + jsonObject.param1 +
                    "<br>" + "param2 = " + jsonObject.param2);
            }
        );
    });
//-->
</script>
</head>
<body>
<div id="div1"></div>
<input type="button" value="Получить данные">
</body>
</html>
```

Листинг 11.5. Содержимое файла ajax.php

```
<?php
header('Expires: Sun, 27 May 2007 01:00:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-store, no-cache, must-revalidate');
header('Pragma: no-cache');
header('Content-Type: text/javascript; charset=utf-8');
if (!isset($_SERVER['HTTP_X_REQUESTED_WITH'])) {
    exit('Данные отправлены не через AJAX');
}
if (!isset($_GET['txt'])) {
    exit('Некорректные параметры запроса');
}
?>
{
param1: "Значение 1",
param2: "<?php echo $_GET['txt']; ?>"
```

Функция `$.getJSON()` позволяет также получать данные в формате JSONP с другого домена методом GET. Чтобы отправить данные этим методом, необходимо после параметров в URL-адресе указать конструкцию:

<Параметр>=?

Название параметра может быть произвольным.

`http://site1/ajax.php?txt=test&callback=?`

Если в конце запроса указан вопросительный знак, то данные будут отправлены не с помощью технологии AJAX, а путем создания тега `<script>` с указанием URL-адреса в параметре `src`. Вместо вопросительного знака будет вставлено произвольное значение. Запрос на сервер будет выглядеть примерно следующим образом.

```
GET /ajax.php?txt=test&callback=jsonp1247518622796&_=1247518624156
HTTP/1.1
Host: site1
```

Как видно из примера, вместо знака вопроса мы получили следующую строку.

`jsonp1247518622796&_=1247518624156`

Значение параметра `callback` необходимо указать перед данными в формате JSON в скрипте на сервере. Сами данные указываются внутри круглых скобок.

```
jsonp1247518622796 ({
param1: "Значение 1",
param2: "Значение 2"
});
```

В качестве примера рассмотрим получение данных в формате JSONP с другого домена. Создадим файл `index.php` (листинг 11.6) и разместим его на домене `http://localhost/`. С помощью этого файла будем обмениваться данными со скриптом `ajax.php` (листинг 11.7), расположенным на домене `http://site1/`.

Листинг 11.6. Содержимое файла `http://localhost/index.php`

```
<html>
<head>
<title>Функция $.getJSON() и JSONP</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("button").click(function() {
        $.getJSON("http://site1/ajax.php?txt=test&callback=?",
            function(jsonObject) {
                $("#div1").html("param1 = " + jsonObject.param1 +
                    "<br>" + "param2 = " + jsonObject.param2);
            }
        );
    });
//-->
</script>
</head>
```

```
<body>
<div id="div1"></div>
<input type="button" value="Получить данные">
</body>
</html>
```

Листинг 11.7. Содержимое файла <http://site1/ajax.php>

```
<?php
header('Content-Type: text/javascript; charset=utf-8');
if (!isset($_GET['callback'])) {
    exit('Некорректные параметры запроса');
}
echo $_GET['callback'];
?>({
param1: "Данные с другого домена",
param2: "<?php echo $_GET['txt']; ?>"
});
```

11.3. Функция `$.getScript()`

Функция `$.getScript()` позволяет загрузить сценарий JavaScript с любого домена. После загрузки сценарий выполняется в глобальном контексте. Формат функции:

```
$.getScript(<URL>[, <Функция обратного вызова>])
```

Если во втором параметре указана функция, то она будет выполнена после получения скрипта с сервера. В качестве параметра `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Данные>[, <Статус>]]) {
/   // ...
}
```

Внутри функции обратного вызова доступна ссылка (`this`) на объект с параметрами запроса. Если данные были получены с того же домена, то через первый параметр будут доступны данные, полученные с сервера, а через второй параметр — статус запроса (значение `success` — при успешном выполнении). В случае получения данных с другого домена переменные, указанные в параметрах, будут иметь значение `undefined`. Кроме того, следует учитывать, что при загрузке данных с другого домена функция обратного вызова будет вызвана независимо от результата.

В качестве примера рассмотрим получение и выполнение скрипта с другого домена. Создадим файл `index.php` (листинг 11.8) и разместим его на домене `http://localhost/`. С помощью этого файла будем загружать файл `script.js` (листинг 11.9), расположенный на домене `http://site1/`.

Листинг 11.8. Содержимое файла <http://localhost/index.php>

```
<html>
<head>
<title>Функция $.getScript()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
```

```

<!--
$(document).ready(function() {
    $(":button").click(function() {
        $.getScript("http://site1/script.js",
            function() {
                if (typeof myStatus != "undefined")
                    alert("Данные успешно получены");
                else alert("Проблемы");
            }
        );
    });
//-->
</script>
</head>
<body>
<input type="button" value="Получить данные">
</body>
</html>

```

Листинг 11.9. Содержимое файла http://site1/script.js

```

var myStatus = "OK";
alert("Это данные с другого домена");

```

11.4. Функция \$.get()

Функция `$.get()` позволяет сделать запрос методом GET. Имеет следующий формат:

```

$.get(<URL> [, <Данные>] [, <Функция обратного вызова> [, <Тип данных>]] )

```

Необязательный параметр `<Данные>` позволяет передать несколько параметров. Для этого параметры и значения должны быть указаны следующим образом.

```

{
Параметр1: "Значение1",
Параметр2: "Значение2",
...
ПараметрN: "ЗначениеN"
}

```

Кроме того, можно указать сформированную строку запроса. В этом случае заботиться о допустимости символов необходимо самим.

```
txt1=%D0%A2%D0%B5%D0%BA%D1%81%D1%82&id=5
```

В качестве параметра `<Функция обратного вызова>` указывается ссылка на функцию следующего формата:

```

function <Название функции>(<Данные> [, <Статус>]) {
    // ...
}

```

Внутри функции обратного вызова доступна ссылка (`this`) на объект с параметрами запроса. Следует учитывать, что функция вызывается только в случае успешного завер-

шения запроса. Если необходимо контролировать процесс запроса и обрабатывать ошибки, то следует воспользоваться функцией `$.ajax()`.

Необязательный параметр `<Тип данных>` позволяет указать тип возвращаемых данных. Могут быть указаны значения "json", "xml" и некоторые другие.

В качестве примера использования функции `$.get()` рассмотрим заполнение взаимосвязанных списков. Изначально из базы данных заполняется первый список. При выборе пункта списка значение параметра `value` будет отправлено на сервер с помощью функции `$.get()`. На сервере делаем запрос и выводим связанные разделы. После получения данных в основном файле заполняем второй список. SQL-запрос для создания таблицы представлен в листинге 11.10. Содержимое файла `index.php` (основной файл) представлено в листинге 11.11, а текст файла `ajax.php` (для вывода связанных разделов) содержится в листинге 11.12.

Листинг 11.10. SQL-запрос для создания таблицы

```
CREATE TABLE `categories` (
  `id` int(11) auto_increment,
  `name` varchar(255),
  `parentID` int(11),
  PRIMARY KEY (`id`)
) engine=MYISAM character set `utf8` collate `utf8_general_ci`;
INSERT INTO `categories` VALUES (1, 'Раздел 1', 0);
INSERT INTO `categories` VALUES (2, 'Раздел 2', 0);
INSERT INTO `categories` VALUES (3, 'Раздел 3', 0);
INSERT INTO `categories` VALUES (4, 'Подраздел 1.1', 1);
INSERT INTO `categories` VALUES (5, 'Подраздел 1.2', 1);
INSERT INTO `categories` VALUES (6, 'Подраздел 2.1', 2);
INSERT INTO `categories` VALUES (7, 'Подраздел 2.2', 2);
INSERT INTO `categories` VALUES (8, 'Подраздел 3.1', 3);
INSERT INTO `categories` VALUES (9, 'Подраздел 3.2', 3);
INSERT INTO `categories` VALUES (10, 'Подраздел 3.3', 3);
```

Листинг 11.11. Содержимое файла index.php

```
<?php
// Запрещаем кеширование
header('Expires: Sun, 27 May 2007 01:00:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-store, no-cache, must-revalidate');
header('Pragma: no-cache');
// Указываем кодировку
header('Content-Type: text/html; charset=utf-8');
// Подключаемся к базе
$db = mysql_connect('localhost', 'root', '123456')
      or die('Не удалось установить подключение');
mysql_select_db('test') or die('Такой базы данных нет');
mysql_query('SET NAMES `utf8`'); // Кодировка соединения
?>
<html>
<head>
<title>Функция $.get()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
```

```

<script type="text/javascript">
<!--
function f_result(data) {
    if (data.indexOf("ERROR") == -1) {
        var el;
        el = '<select size="4" id="sel2">' + data + '</select>';
        $("#sel2").replaceWith(el);
    }
    else {
        if (data == "") alert("Ошибка запроса");
        else alert(data);
    }
    $("#div1").stop().hide(300);
}
$(document).ready(function() {
    $("#sel1").change(function() {
        $("#div1").stop().show(300);
        $("#sel2").attr("disabled", true);
        $.get("/ajax.php", { id: $("#sel1").val() }, f_result);
    });
//-->
</script>
<style type="text/css">
#sel1, #sel2 { width:150px; }
</style>
</head>
<body>
<select size="4" id="sel1">
<?php
$query = 'SELECT `id`, `name` FROM `categories` ';
$query .= 'WHERE `parentID`=0 ORDER BY `name`';
if ($res = mysql_query($query)) {
    while ($row = mysql_fetch_row($res)) {
        echo '<option value="' . $row[0] . '">';
        echo htmlspecialchars($row[1], ENT_COMPAT, 'UTF-8');
        echo "</option>\n";
    }
}
mysql_close($db);
?>
</select>&nbsp;&nbsp;&nbsp;
<select size="4" id="sel2" style="display:none;">
</select>
<div id="div1" style="display:none;">Загрузка...</div>
</body>
</html>

```

Листинг 11.12. Содержимое файла ajax.php

```

<?php
// Запрещаем кеширование
header('Expires: Sun, 27 May 2007 01:00:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');

```

```

header('Cache-Control: no-store, no-cache, must-revalidate');
header('Pragma: no-cache');
// Указываем кодировку
header('Content-Type: text/html; charset=utf-8');
if (!isset($_SERVER['HTTP_X_REQUESTED_WITH'])) {
    exit('ERROR: Данные отправлены не через AJAX');
}
if (isset($_GET['id'])) $id = (int)$_GET['id'];
else $id = 0;
if ($id === 0) exit('ERROR: Параметр id содержит ошибку');
// Подключаемся к базе
$db = mysql_connect('localhost', 'root', '123456')
    or die('ERROR: Не удалось установить подключение');
mysql_select_db('test') or die('ERROR: Такой базы данных нет');
mysql_query('SET NAMES `utf8`'); // Кодировка соединения
$query = 'SELECT `id`, `name` FROM `categories`';
$query .= ' WHERE `parentID`=' . $id . ' ORDER BY `name`';
if ($res = mysql_query($query)) {
    if (mysql_num_rows($res) != 0) {
        while ($row = mysql_fetch_row($res)) {
            echo '<option value="' . $row[0] . '">' .
                htmlspecialchars($row[1], ENT_COMPAT, 'UTF-8') .
            '</option>\n';
        }
    } else {
        echo '<option value="0">Разделов нет</option>';
    }
} else {
    echo 'ERROR: Ошибка выполнения';
}
mysql_close($db);
?>

```

11.5. Функция \$.post()

Функция `$.post()` позволяет сделать запрос методом POST. Имеет следующий формат:

```
$.post(<URL>[, <Данные>] [, <Функция обратного вызова>[, <Тип данных>]])
```

Следует учитывать, что данные можно загрузить только с того же домена. Необязательный параметр `<Данные>` позволяет передать несколько параметров. Для этого параметры и значения должны быть указаны следующим образом.

```
{
Параметр1: "Значение1",
Параметр2: "Значение2",
...
ПараметрN: "ЗначениеN"
}
```

Кроме того, можно указать сформированную строку запроса. В этом случае заботиться о допустимости символов необходимо самим.

```
txt1=%D0%A2%D0%B5%D0%BA%D1%81%D1%82&id=5
```

В качестве параметра *<Функция обратного вызова>* указывается ссылка на функцию следующего формата.

```
function <Название функции>(<Данные> [, <Статус>]) {  
    // ...  
}
```

Внутри функции обратного вызова доступна ссылка (*this*) на объект с параметрами запроса. Следует учитывать, что функция вызывается только в случае успешного завершения запроса. Если необходимо контролировать процесс запроса и обрабатывать ошибки, то следует воспользоваться функцией *\$.ajax()*.

Необязательный параметр *<Тип данных>* позволяет указать тип возвращаемых данных. Могут быть указаны значения "json", "xml" и некоторые другие. Если указано значение "json", то данные будут выполнены с помощью функции *eval()*, и JSON-объект будет доступен через параметр *<Данные>* в функции обратного вызова.

```
$.post("/ajax.php", { id:1 }, function(data) {  
    alert(data.param1 + "\n" + data.param2);  
}, "json");
```

Скрипт на сервере:

```
<?php  
header('Content-Type: text/javascript; charset=utf-8');  
?  
{  
param1: "Текст",  
param2: 12  
}
```

Результат выполнения:

```
Текст  
12
```

Если указано значение "xml", то через параметр *<Данные>* в функции обратного вызова будет доступен документ в формате XML.

```
$.post("/ajax.php", { id:1 }, function(xml) {  
    var text1 = $("text1", xml);  
    var text2 = $("text2", xml);  
    alert(text1.eq(0).text() + " - " + text2.eq(0).text() +  
        " - " + text1.eq(1).text() + " - " + text2.eq(1).text());  
}, "xml");
```

Скрипт на сервере:

```
<?php  
header('Content-Type: text/xml; charset=utf-8');  
echo '<?xml version="1.0" encoding="utf-8" ?>', "\n";  
?  
<ajax>  
    <ответ>  
        <text1>Поле 1</text1>
```

```
<text2>Поле 2</text2>
</otvet>
<otvet>
    <text1>Поле 3</text1>
    <text2>Поле 4</text2>
</otvet>
</ajax>
```

Результат выполнения:

Поле 1 - Поле 2 - Поле 3 - Поле 4

На самом деле указывать значение "xml" необязательно. Если сервер возвращает XML-документ и соответствующие заголовки, то через параметр `<Данные>` в функции обратного вызова будет доступен документ в формате XML и без указания значения "xml".

Обратите внимание на следующую строку.

```
var text1 = $("text1", xml);
```

Библиотека jQuery позволяет работать с XML-документом так же, как и с HTML-документом. Для этого необходимо во втором параметре функции `$()` указать XML-документ в качестве контекста.

Для примера рассмотрим форму отправки сообщения. Отправлять сообщение на почтовый адрес электронной почты мы не будем, так как очень часто письма не доходят до адресата. Вместо этого будем сохранять сообщение в базе данных MySQL. В этом случае администрация сайта точно получит сообщение. SQL-запрос для создания таблицы представлен в листинге 11.13. Содержимое файла `index.php` (файл с формой) представлено в листинге 11.14, а текст файла `ajax.php` (для сохранения сообщения) содержится в листинге 11.15.

Листинг 11.13. SQL-запрос для создания таблицы

```
CREATE TABLE `mail` (
    `id` int(11) auto_increment,
    `name` varchar(255) default '',
    `email` varchar(255) default '',
    `tema` varchar(255) default '',
    `msg` text,
    `date_msg` datetime,
    `status` enum('new', 'old'),
    PRIMARY KEY (`id`)
) engine=MYISAM character set `utf8` collate `utf8_general_ci`
```

Листинг 11.14. Содержимое файла index.php

```
<?php
// Запрещаем кэширование
header('Expires: Sun, 27 May 2007 01:00:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-store, no-cache, must-revalidate');
header('Pragma: no-cache');
// Указываем кодировку
header('Content-Type: text/html; charset=utf-8');
?>
<html>
```

```

<head>
<title>Функция $.post()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
function f_result(xml) {
    var status = $("status", xml).eq(0).text();
    var msg = $("msg", xml).eq(0).text();
    if (status == "ok") {
        $("#author").val("");
        $("#emails").val("");
        $("#tema").val("");
        $("#msg").val("");
    }
    $("#div1").html(msg);
    $("#btn").removeAttr("disabled");
}
$(document).ready(function() {
    $("#frm").submit(function() {
        $("#btn").attr("disabled", true);
        $("#div1").html("Загрузка...");
        $.post("/ajax.php", $("#frm").serialize(), f_result, "xml");
        return false;
    });
    $("#author").keypress(function(e) {
        if (e.keyCode==13) {
            $("#emails").focus();
            return false;
        }
    });
    $("#emails").keypress(function(e) {
        if (e.keyCode==13) {
            $("#tema").focus();
            return false;
        }
    });
    $("#tema").keypress(function(e) {
        if (e.keyCode==13) {
            $("#msg").focus();
            return false;
        }
    });
});
//--
</script>
</head>
<body>
<div id="div1"></div>
<form method="POST" action="ajax.php" id="frm">
<b>Имя:</b><br>
<input type="text" name="author" id="author" size="40"><br>
<b>E-mail:</b><br>
<input type="text" name="emails" id="emails" size="40"><br>

```

```

<b>Тема:</b><br>
<input type="text" name="tema" id="tema" size="40"><br>
<b>Сообщение:</b><br>
<textarea name="msg" id="msg" style="width:300px; height:150px">
</textarea><br><br>
<input type="submit" id="btn" value="Отправить">
</form>
</body>
</html>

```

Листинг 11.15. Содержимое файла ajax.php

```

<?php
// Запрещаем кеширование
header('Expires: Sun, 27 May 2007 01:00:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-store, no-cache, must-revalidate');
header('Pragma: no-cache');

if (!isset($_SERVER['HTTP_X_REQUESTED_WITH'])) {
    header('Content-Type: text/html; charset=utf-8');
    // Здесь обрабатываем данные, если отключен JavaScript
    exit('Данные отправлены не через AJAX');
}

else { // Если отправлено через AJAX
    // Указываем тип документа и кодировку
    header('Content-Type: text/xml; charset=utf-8');
    // Получаем данные
    if (isset($_POST['emails'])) $emails = $_POST['emails'];
    else $emails = '';
    if (isset($_POST['author'])) $author = $_POST['author'];
    else $author = '';
    if (isset($_POST['tema'])) $tema = $_POST['tema'];
    else $tema = '';
    if (isset($_POST['msg'])) $msg = $_POST['msg'];
    else $msg = '';
    $err_msg = '';
    // Удаляем пробельные символы
    $author = trim($author);
    $tema = trim($tema);
    $msg = trim($msg);
    // Если директива magic_quotes_gpc включена,
    // то удаляем защитные слеши перед спецсимволами
    if (get_magic_quotes_gpc()) {
        $author = stripslashes($author);
        $tema = stripslashes($tema);
        $msg = stripslashes($msg);
    }
    $pattern = '/^([a-z0-9_.-]+@[a-z0-9-]+\.)+[a-z]{2,6}$/';
    if (!preg_match($pattern, $emails)) {
        $err_msg .= 'Недопустимый адрес E-mail !<br>';
    }
    if (mb_strlen($emails, 'UTF-8') > 60) {
        $err_msg .= 'Длина E-mail больше 60 символов!<br>';
    }
}

```

```

if (mb_strlen($author, 'UTF-8') == 0) {
    $err_msg .= 'Поле Имя не заполнено!<br>';
}
if (mb_strlen($author, 'UTF-8') > 30) {
    $err_msg .= 'Длина поля Имя более 30 символов!<br>';
}
if (mb_strlen($tema, 'UTF-8') == 0) {
    $err_msg .= 'Поле Тема не заполнено!<br>';
}
if (mb_strlen($tema, 'UTF-8') > 50) {
    $err_msg .= 'Длина поля Тема более 50 символов!<br>';
}
if (mb_strlen($msg, 'UTF-8') == 0) {
    $err_msg .= 'Поле Сообщение не заполнено!<br>';
}
if (mb_strlen($msg, 'UTF-8') > 65000) {
    $err_msg .= 'Длина поля Сообщение более 65000 символов!<br>';
}
if ($err_msg == '') {
    // Подключаемся к базе
    if (!@$db = mysql_connect('localhost', 'root', '123456')) {
        $err_msg .= 'Не удалось установить подключение!<br>';
    }
    else {
        if (mysql_select_db('test')) {
            mysql_query('SET NAMES `utf8`');
        }
        else {
            $err_msg .= 'Такой базы данных нет!<br>';
        }
    }
}
if ($err_msg == '') {
    // Если ошибок нет, то сохраняем сообщение
    // При желании можно отправить на E-mail с помощью функции
    mail()
    $author = mysql_real_escape_string($author);
    $tema = mysql_real_escape_string($tema);
    $msg = mysql_real_escape_string($msg);
    $q = "INSERT INTO `mail` VALUES (NULL, '$author', '$emails', ";
    $q .= "'$tema', '$msg', NOW(), 'new')";
    if (!mysql_query($q)) {
        $err_msg .= 'Не удалось сохранить сообщение<br>';
    }
    mysql_close($db);
}
if ($err_msg != '') {
    $status = 'error';
    $otvet = '<span style="color:red;">' . $err_msg . '</span>';
}
else {
    $status = 'ok';
    $otvet = '<span style="color:green; font-weight:bold;">';
    $otvet .= 'Сообщение успешно сохранено</span>';
}

```

```

        }
        $otvet = htmlspecialchars($otvet, ENT_COMPAT, 'UTF-8');

        echo "<ajax>\n";
        echo '<status>' . $status . "</status>\n";
        echo '<msg>' . $otvet . "</msg>\n";
        echo "</ajax>\n";
    }
?>

```

11.6. Функция \$.ajax()

Практически все функции, которые мы рассматривали в предыдущих разделах, вызывают функцию `$.ajax()`. Она является универсальной и позволяет полностью контролировать весь процесс обмена данными с сервером. Формат функции:

`$.ajax(<Объект с опциями>)`

Функция возвращает ссылку на объект `XMLHttpRequest`. Единственный параметр представляет собой объект, состоящий из пар “опция/значение”. Могут быть указаны следующие опции.

- `url` — URL-адрес обработчика на сервере.
- `type` — метод передачи данных (GET (значение по умолчанию) или POST).
- `data` — данные, которые будут переданы на сервер. Параметры и значения должны быть указаны следующим образом.

```

{
    Параметр1: "Значение1",
    Параметр2: "Значение2",
    ...
    ПараметрN: "ЗначениеН"
}

```

В случае если параметры или значения содержат запрещенные символы, то они будут автоматически перекодированы с помощью метода `encodeURIComponent()`. В качестве значения опции `data` можно также указать сформированную строку запроса. В этом случае заботиться о допустимости символов необходимо самим.

`txt=%D0%A2%D0%B5%D0%BA%D1%81%D1%82&id=80`

- `dataType` — ожидаемый тип данных в ответе сервера. Если опция не указана, то тип будет определяться автоматически по MIME-типу в заголовках ответа сервера. Можно указать следующие типы данных.
 - ◊ `text` — простой текст; ответ будет получен через свойство `responseText`.
 - ◊ `html` — HTML-документ; ответ будет получен через свойство `responseText`.
 - ◊ `xml` — XML-документ; ответ будет получен через свойство `responseXML` объекта `XMLHttpRequest`. Свойство будет доступно, только если в заголовках ответа сервера указан MIME-тип `application/xml` или `text/xml`. Кроме того, XML-документ должен иметь правильную структуру и соответствовать спецификации DOM.

- ◊ `script` — JavaScript-код; после получения код выполняется в глобальном контексте. Если данные были получены с того же домена, то внутри функции, указанной в опции `success`, будут доступны данные в виде обычного текста. Данные также могут быть получены с другого домена. В случае получения данных с другого домена, внутри функции, указанной в опции `success`, данные будут иметь значение `undefined`. Кроме того, следует учитывать, что, при загрузке данных с другого домена, функция, указанная в опции `success`, будет вызвана независимо от результата.
 - ◊ `json` — ответ в формате JSON; после получения данные автоматически выполняются с помощью функции `eval()`, и JSON-объект становится доступным внутри функции, указанной в опции `success`.
 - ◊ `jsonp` — ответ в формате JSONP; данные могут быть получены с другого домена. Чтобы отправить данные в этом формате, необходимо после параметров в URL-адресе указать конструкцию: `<Параметр>=?`. Название параметра может быть произвольным. Если в конце запроса указан вопросительный знак, то данные будут отправлены не с помощью технологии AJAX, а путем создания тега `<script>` с указанием URL-адреса в параметре `src`. Вместо вопросительного знака будет вставлено произвольное значение. Значение из конструкции `<Параметр>` необходимо указать перед данными в формате JSON в скрипте на сервере. Сами данные указываются внутри круглых скобок.
 - `async` — тип запроса. Могут быть заданы следующие значения.
 - ◊ `true` — асинхронный запрос (выполнение текущего скрипта не приостанавливается до получения ответа); значение по умолчанию.
 - ◊ `false` — синхронный запрос (выполнение текущего скрипта будет продолжено только после получения ответа сервера).
 - `cache` — управление кешированием. Опция имеет смысл только при запросах методом GET. Могут быть заданы следующие значения.
 - ◊ `true` — ответ может быть помещен в кеш веб-браузера. Если в заголовках ответа сервера указано запрещение кеширования, то данные не будут помещены в кеш.
 - ◊ `false` — запрещение кеширования. Если указано это значение, то к строке запроса будет добавлен параметр `_=<Случайное число>`.
- `GET /ajax.php?_=1247862154453&t=5 HTTP/1.1`
- `timeout` — максимальное время выполнения запроса в миллисекундах.
 - `contentType` — позволяет указать значение заголовка `Content-Type`. По умолчанию при запросах методом POST имеет значение `application/x-www-form-urlencoded`.
 - `processData` — если указано значение `false`, то данные, указанные в опции `data`, не будут преобразовываться в строку запроса. Например, можно передать массив.
- ```
$.ajax({
 url: "/ajax.php",
 data: [1,2,3],
```

```
 dataType: "text",
 processData: false,
 success: function(data) {
 alert(data);
 }
);
}
```

В результате строка запроса будет выглядеть так.

```
GET /ajax.php?1,2,3 HTTP/1.1
```

Если опция имеет значение `true` или не указана, то результат будет другим.

```
GET /ajax.php?undefined=undefined&undefined=undefined
&undefined=undefined HTTP/1.1
```

- `ifModified` — если указать значение `true`, то запрос будет успешным только в том случае, если данные изменились. Для этого в первом запросе будет добавлен заголовок.

```
If-Modified-Since: Thu, 01 Jan 1970 00:00:00 GMT
```

Во втором запросе в этом заголовке будет уже указана дата последнего изменения данных, полученная с сервера в первом запросе.

```
If-Modified-Since: Sun, 25 Jan 2009 23:48:56 GMT
```

Если сервер вернет код статуса 304 (документ без изменений), то функция, указанная в опции `success`, вызвана не будет.

- `jsonp` — позволяет указать название параметра для JSONP-запроса. При указании этой опции нет необходимости в конструкции `<Параметр>=?`.

```
$.ajax({
 url: "http://site1/ajax.php",
 data: "txt=test",
 dataType: "jsonp",
 jsonp: "callback",
 success: function(data) {
 alert(data.param1);
 }
});
```

Строка запроса будет выглядеть следующим образом.

```
GET /ajax.php?callback=jsonp1247869542750&_=1247869544031&txt=
test HTTP/1.1
```

- `scriptCharset` — позволяет указать кодировку данных при значениях опции `dataType`, равных `"jsonp"` или `"script"`.
- `username` — логин для HTTP-авторизации.
- `password` — пароль для HTTP-авторизации.
- `beforeSend` — ссылка на функцию, которая будет вызвана до передачи запроса на сервер. С помощью этой функции можно, например, добавить или изменить заголовки запроса. Формат функции:

```
function <Название функции>(<Объект XMLHttpRequest>) {
 // ...
}
```

В качестве примера изменим значение заголовка User-Agent.

```
beforeSend: function(obj) {
 obj.setRequestHeader("User-Agent", "MyRobot");
}
```

- `dataFilter` — ссылка на функцию, которая будет вызвана сразу после получения ответа с сервера, но до обработки данных с помощью jQuery. Формат функции:

```
function <Название функции>(<Данные>[, <Тип данных>]) {
 // Обрабатываем данные
 return <Данные>; // Возвращаем обработанные данные
}
```

Внутри функции необходимо возвращать обработанные данные.

- `complete` — ссылка на функцию, которая будет вызвана по окончании запроса, независимо от успешности его выполнения. Формат функции:

```
function <Название функции>(<Объект XMLHttpRequest>[, <Статус>]) {
 // ...
}
```

- `success` — ссылка на функцию, которая будет вызвана в случае успешного выполнения запроса. Формат функции:

```
function <Название функции>(<Данные>[, <Статус>]) {
 // ...
}
```

Через параметр `<Данные>` доступны данные, обработанные с учетом значения опции `dataType`.

- `error` — ссылка на функцию, которая будет вызвана при ошибке. Формат функции:

```
function <Название функции>([<Объект XMLHttpRequest>[, <Текст ошибки>[, <Исключение>]]]) {
 // ...
}
```

Параметр `<Текст ошибки>` может возвращать следующие значения: `null`, `timeout`, `error`, `notmodified` или `parsererror`. Параметр `<Исключение>` позволяет получить информацию об ошибке при значении `null` в параметре `<Текст ошибки>`. Например, в веб-браузере Opera при обращении к другому домену в параметре `<Исключение>` будет следующее значение:

```
[Error:
name: Error
message: Security violation
]
```

Все указанные параметры можно указать в функции `$.ajaxSetup()`. В этом случае это будут значения по умолчанию для всех запросов. Формат функции:

```
$.ajaxSetup(<Объект с параметрами>)
```

Рассмотрим пример.

```
$.ajaxSetup({ // Запрос методом GET
 url: "/ajax.php", // URL-адрес
```

```

dataType: "html", // Тип возвращаемых данных
cache: false, // Запрет кеширования
success: function(data) { // При удачном запросе
 alert(data);
},
error: function(obj, err) { // При ошибке
 alert("Ошибка: " + err);
}
});
$.ajax({ // Все опции запроса указаны в $.ajaxSetup()
 data: { id:1, txt:"Данные" } // Передаваемые данные
});

```

## 11.7. Глобальные обработчики событий AJAX

Помимо локальных обработчиков событий, библиотека jQuery предоставляет несколько глобальных. Все глобальные обработчики являются методами объекта jQuery и требуют привязки к элементам коллекции. Могут быть назначены следующие обработчики.

- `ajaxStart()` — вызывается в самом начале запроса. В качестве параметра указывается ссылка на функцию следующего формата.

```

function <Название функции>() {
 // Обрабатываем событие
}

```

- `ajaxSend()` — срабатывает до передачи запроса на сервер. В качестве параметра указывается ссылка на функцию следующего формата.

```

function <Название функции>([<Объект event>[, <Объект XMLHttpRequest>[, <Опции запроса>]]]) {
 // Обрабатываем событие
}

```

Выведем значение опции `dataType`.

```

$("#div1").ajaxSend(function(e, xhr, options) {
 alert(options.dataType);
});

```

- `ajaxSuccess()` — вызывается в случае успешного выполнения запроса. В качестве параметра указывается ссылка на функцию следующего формата.

```

function <Название функции>([<Объект event>[, <Объект XMLHttpRequest>[, <Опции запроса>]]]) {
 // Обрабатываем событие
}

```

Выведем текст, полученный с сервера.

```

$("#div1").ajaxSuccess(function(e, xhr, options) {
 alert(xhr.responseText);
});

```

- `ajaxError()` — срабатывает в случае ошибки. В качестве параметра указывается ссылка на функцию следующего формата.

```

function <Название функции>([<Объект event>[, <Объект XMLHttpRequest>[, <Опции запроса>[,

```

```
<Исключение>]]])) {
 // Обрабатываем событие
}
```

- `ajaxComplete()` — вызывается по окончании запроса, независимо от успешности его выполнения. В качестве параметра указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>[, <Объект XMLHttpRequest>[, <Опции запроса>]]) {
 // Обрабатываем событие
}
```

Выведем текст, полученный с сервера, в случае успешности запроса или сообщение об ошибке — при неудачном запросе.

```
$("#div1").ajaxComplete(function(e, xhr) {
 if (xhr.readyState == 4 && xhr.status == 200) {
 alert(xhr.responseText);
 }
 else {
 alert("Ошибка");
 }
});
```

- `ajaxStop()` — срабатывает в самом конце запроса. В качестве параметра указывается ссылка на функцию следующего формата.

```
function <Название функции>() {
 // Обрабатываем событие
}
```

Текущий элемент доступен внутри функций в обработчиках через указатель `this`. Обратите внимание на то, что указатель `this` ссылается на текущий элемент *объектной модели документа*, а не на элемент коллекции `jQuery`.

Назначить обработчик можно также с помощью метода `bind()`. В первом параметре указывается глобальное событие AJAX, а во втором — ссылка на функцию, которая будет вызвана при наступлении события.

```
$("#div1").bind("ajaxSuccess", function() {
 $(this).append("Ответ успешно получен
");
}).bind("ajaxError", function() {
 $(this).append("Произошла ошибка
");
});
```

Если в функции `$.ajax()` или `$.ajaxSetup()` опция `global` имеет значение `false`, то глобальные обработчики вызываться не будут. По умолчанию опция имеет значение `true`.

```
$.ajax({
 url: "/ajax.php",
 dataType: "html",
 data: { id:1, txt:"Данные" },
 global: false // Глобальные обработчики отключены
});
```

Продемонстрируем последовательность прохождения событий (листинг 11.16).

### Листинг 11.16. Последовательность событий AJAX

```
<html>
<head>
<title>Последовательность событий AJAX</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#div1").ajaxStart(function() {
 $(this).append("Глобальное событие ajaxStart
");
 }).ajaxSend(function() {
 $(this).append("Глобальное событие ajaxSend
");
 }).ajaxSuccess(function() {
 $(this).append("Глобальное событие ajaxSuccess
");
 }).ajaxError(function() {
 $(this).append("Глобальное событие ajaxError
");
 }).ajaxComplete(function() {
 $(this).append("Глобальное событие ajaxComplete
");
 }).ajaxStop(function() {
 $(this).append("Глобальное событие ajaxStop
");
 });
 $("button").click(function() {
 $.ajax({
 url: "/ajax.php",
 dataType: "html",
 cache: false,
 beforeSend: function() {
 $("#div1").append("Локальное событие beforeSend
");
 },
 dataFilter: function(data) {
 $("#div1").append("Локальное событие dataFilter
");
 return data;
 },
 success: function() {
 $("#div1").append("Локальное событие success
");
 },
 error: function() {
 $("#div1").append("Локальное событие error
");
 },
 complete: function() {
 $("#div1").append("Локальное событие complete
");
 },
 data: { id:1, txt:"Данные" }
 });
 });
});
//-->
</script>
</head>
<body>
<div id="div1"></div>
```

```
<input type="button" value="Получить данные">
</body>
</html>
```

---

При успешном запросе результат будет следующим.

Глобальное событие ajaxStart  
Локальное событие beforeSend  
Глобальное событие ajaxSend  
Локальное событие dataFilter  
Локальное событие success  
Глобальное событие ajaxSuccess  
Локальное событие complete  
Глобальное событие ajaxComplete  
Глобальное событие ajaxStop

При ошибке получим такой результат.

Глобальное событие ajaxStart  
Локальное событие beforeSend  
Глобальное событие ajaxSend  
Локальное событие error  
Глобальное событие ajaxError  
Локальное событие complete  
Глобальное событие ajaxComplete  
Глобальное событие ajaxStop



## Глава 12

# Библиотека jQuery UI

jQuery UI (*User Interface*) — это библиотека визуальных компонентов пользовательского интерфейса, обеспечивающая кроссбраузерную поддержку (работает в Internet Explorer 6.0+, Mozilla Firefox 2+, Safari 3.1+, Opera 9.0+ и Chrome 1.0). Библиотека имеет модульную структуру. На странице загрузки <http://jqueryui.com/download> можно выбрать компоненты, которые планируется использовать. Прежде чем загружать готовые компоненты, необходимо на странице <http://jqueryui.com/themeroller/> выбрать тему оформления. На вкладке *Gallery* (рис. 12.1) представлены примеры оформления компонента *Datepicker* (календарь). Щелкнув мышью на понравившемся календаре, можно сразу увидеть, как будут выглядеть другие компоненты. Если ни одна из тем не подходит, то на вкладке *Roll Your Own* можно изменить значения различных атрибутов стиля и создать свою собственную тему.

Прежде чем использовать возможности jQuery UI, необходимо подключить базовую библиотеку jQuery. При подключении следует учитывать версии библиотек. Для нормальной работы jQuery UI 1.7.2 необходима библиотека jQuery 1.3.2, а для jQuery UI 1.6 требуется библиотека jQuery 1.2.6. В этой главе мы рассмотрим возможности jQuery UI версии 1.7.2.

### **Примечание**

В приведенном описании указываются точные версии подключаемых библиотек. Скорее всего, со временем подготовки книги будут выпущены новые версии. В этом случае рекомендуется использовать их, особенно если номера версий отличаются только последними цифрами.

## **12.1. Модуль UI Draggable — перемещение элементов**

Модуль UI Draggable позволяет свободно перемещать любой DOM-элемент с помощью мыши. Для загрузки модуля переходим на страницу <http://jqueryui.com/download>. Оставляем установленными флагги *UI Core*, *Draggable*, 1.7.2, а затем щелкаем на кнопке *Download*. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы следующие файлы:

- `jquery-1.3.2.min.js` (папка `js`);
- `jquery-ui-1.7.2.custom.min.js` (папка `js`).

Все указанные файлы необходимо подключить к скрипту в том порядке, в котором они перечислены. В качестве примера рассмотрим использование модуля с параметрами по умолчанию (листинг 12.1).

jQuery UI · ThemeRoller · Opera

Файл Правка Вид Закладки Виджеты Инструменты Справка  
Создать вкладку jQuery UI · ThemeRoller

http://jqueryui.com/themeroller/#ffDefault=Lucida+Grande,+Lucida+Sans,+Arial,+sans-serif&fwDefault=bold&fsDef=14px&fsSel=16px

UI Plugins Themes Examples

# jQuery

UI · Documentation Themes Plugins Examples

\* New! Bring ThemeRoller into any page. Get the ThemeRoller Firefox Bookmarklet.

**Accordion**

Section 1

Mauris mauns ante, blandit et, ultrices a, suscipit eget, quam Integer ut neque Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc Nam a nibh Donec suscipit eros Nam mi Proin viverra leo ut odio Curabitur malesuada Vestibulum a velit eu ante scelerisque vulputate

Section 2

Section 3

**Slider**

**Datepicker**

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

**Progressbar**

**Highlight / Error**

Hey! Sample ui-state-highlight style.

ThemeRoller · Polyfill · Gallery · Help

Download Edit · Themes · Examples

December 2009

First Second Third

jQuery UI · Documentation Themes Plugins Examples

## Листинг 12.1. UI Draggable. Перемещение элементов

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Draggable. Перемещение элементов</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<style type="text/css">
* { margin:0; padding:0; }
body { font-size:8pt; font-family:Verdana, sans-serif; }
#draggable {
 position:absolute;
 top:50px; left:50px;
 width:100px; height:100px;
 padding:2px;
 border:2px solid #000000;
}
.ui-draggable {
 /* Класс, добавляемый после применения метода draggable() */
 background-color:#E8E8E8;
}
.ui-draggable-dragging {
 /* Класс, добавляемый при перемещении элемента */
 background-color:#008800;
}
</style>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#draggable").draggable();
});
//-->
</script>
</head>
<body>
<div id="draggable"><p>Передвинь меня с помощью мыши</p></div>
</body>
</html>
```

Сделать перемещаемым можно практически любой элемент, хотя чаще всего используется элемент DIV. В нашем примере для элемента с идентификатором draggable мы указали атрибут position со значением absolute. Если атрибут не указан, то после применения метода draggable() автоматически будет добавлен атрибут position со значением relative. Кроме того, по умолчанию добавляется класс ui-draggable, а при перемещении элемента — класс ui-draggable-dragging, который удаляется после окончания перемещения элемента.

Для того чтобы элемент можно было перемещать, необходимо указать это с помощью метода draggable(). Формат метода:

```
draggable([<Объект с опциями>])
```

Параметр *<Объект с опциями>* представляет собой объект, состоящий из пар “опция/значение”. Могут быть указаны следующие опции.

- `addClasses` — если указано значение `false`, то класс `ui-draggable` не будет добавляться при инициализации. Значение по умолчанию — `true`.
- `axis` — позволяет ограничить перемещение элемента. Могут быть указаны следующие значения:
  - ◊ `x` — перемещение только по горизонтали;
  - ◊ `y` — перемещение только по вертикали;
  - ◊ `false` — без ограничения (значение по умолчанию).
- `revert` — определяет, куда можно переместить элемент. Могут быть заданы следующие значения:
  - ◊ `false` — можно переместить куда угодно (значение по умолчанию);
  - ◊ `true` — после окончания перемещения элемент автоматически вернется на прежнюю позицию;
  - ◊ `invalid` — после окончания перемещения элемент вернется на прежнюю позицию, если он не был отпущен над элементом, к которому применен метод `droppable()` (применяется при использовании модуля UI Draggable);
  - ◊ `valid` — можно переместить куда угодно, но если элемент был отпущен над элементом, к которому применен метод `droppable()`, то он автоматически вернется на прежнюю позицию (применяется при использовании модуля UI Draggable).
- `revertDuration` — продолжительность анимации при возвращении элемента на прежнее место (в миллисекундах). Используется, если опция `revert` не равна значению `false`. Значение по умолчанию — `500`.
- `containment` — позволяет ограничить перемещение в пределах указанных границ. В качестве значения можно указать DOM-элемент, селектор jQuery, массив с координатами области (`[x1, y1, x2, y2]`) или одно из значений:
  - ◊ `parent` — в пределах родительского элемента;
  - ◊ `window` — в пределах окна;
  - ◊ `document` — в пределах всего документа.Значение по умолчанию — `false`.
- `opacity` — задает степень прозрачности элемента во время перемещения. Может быть указано вещественное число от 0 до 1. Значение по умолчанию — `false`.
- `cursor` — вид курсора при перемещении элемента. Указывается одно из значений атрибута `cursor` в CSS. Значение по умолчанию — `auto`.
- `cursorAt` — задает местоположение курсора при перемещении элемента. В качестве значения указывается объект с комбинацией следующих свойств:
  - ◊ `top` — положение относительно верхней границы;
  - ◊ `left` — положение относительно левой границы;

- ◊ `right` — положение относительно правой границы. При положительных значениях курсор сдвигается внутрь элемента, а при отрицательных значениях отодвигается от элемента справа;
- ◊ `bottom` — положение относительно нижней границы. При положительных значениях курсор сдвигается внутрь элемента, а при отрицательных значениях отодвигается от элемента снизу.

Значение по умолчанию — `false` (при перемещении элемента курсор располагается в том месте, где был произведен щелчок мышью).

Рассмотрим несколько примеров. Во время перемещения курсор будет находиться над верхней границей элемента.

```
cursorAt: { top: 0 }
```

Курсор будет расположен в левом верхнем углу.

```
cursorAt: { top: 0, left: 0 }
```

Курсор будет расположен на одном уровне с верхней границей, но сдвинут на 5 px от элемента относительно правой границы.

```
cursorAt: { top: 0, right: -5 }
```

Курсор будет смешен на 5 px ниже нижней границы и на 5 px от левой границы элемента.

```
cursorAt: { bottom: -5, left: -5 }
```

### **Обратите внимание**

Нельзя одновременно указывать свойства `top` и `bottom`, а также `left` и `right`.

- `delay` — позволяет задать задержку в миллисекундах перед началом перемещения элемента. Используется для предотвращения нежелательного перемещения при случайном щелчке на элементе. Значение по умолчанию — 0.
- `distance` — расстояние в пикселях, после прохождения которого начинается перемещение элемента. Используется для предотвращения нежелательного перемещения при случайном щелчке на элементе. Значение по умолчанию — 1.
- `grid` — устанавливает шаг перемещения по горизонтали и вертикали (в пикселях). В качестве значения указывается массив с двумя элементами.

[*<Шаг по горизонтали>, <Шаг по вертикали>*]

Рассмотрим пример.

```
grid: [40, 60]
```

Значение по умолчанию — `false`.

- `snap` — если указано значение `true`, то элемент будет “прилипать” к границе любого перемещаемого элемента в случае приближения к нему на расстояние, заданное в опции `snapTolerance`. В качестве значения можно также указать селектор jQuery. Значение по умолчанию — `false`.

- `snapTolerance` — минимальное расстояние (в пикселях), при котором перемещаемый элемент начнет “прилипать” к границе другого элемента. Применяется, если значение опции `snap` не равно `false`. Значение по умолчанию — `20`.
- `snapMode` — позволяет указать, к каким сторонам границы будет “прилипать” перемещаемый элемент. Применяется, если значение опции `snap` не равно `false`. Могут быть указаны следующие значения:
  - ◊ `both` — “прилипание” к внутренним и внешним сторонам границы (значение по умолчанию);
  - ◊ `inner` — только к внутренней стороне границы;
  - ◊ `outer` — только к внешней стороне границы.
- `scroll` — если указано значение `false`, то перемещение элемента за границы окна не будет приводить к автоматической прокрутке документа. Значение по умолчанию — `true`.
- `scrollSensitivity` — расстояние (в пикселях) от края окна, при котором будет происходить прокрутка документа. Обратите внимание на то, что расстояние отсчитывается от указателя мыши, а не от границы перемещаемого элемента. Значение по умолчанию — `20`.
- `scrollSpeed` — скорость прокрутки документа. Значение по умолчанию — `20`.
- `helper` — позволяет указать элемент, который будет перемещаться. Могут быть указаны следующие значения:
  - ◊ `original` — перемещается сам элемент (значение по умолчанию);
  - ◊ `clone` — перемещается копия элемента.

В качестве значения может быть указана ссылка на функцию. В этом случае функция должна возвращать DOM-элемент. Пример возврата текущего элемента.

```
helper: function() { return this; }
```

А теперь пример возврата копии элемента:

```
helper: function() { return $(this).clone()[0]; }
```

Перемещение созданного элемента:

```
helper: function() { return $("<div>Произвольный
элемент</div>"); }
```

- `appendTo` — ссылка на элемент-контейнер, в конец которого будет вставляться элемент, указанный в опции `helper` или если значение опции `helper` равно `clone`. В качестве значения указывается селектор jQuery или ссылка на элемент. Значение по умолчанию — `parent` (для вставки используется родительский элемент).
- `zIndex` — значение CSS-атрибута `z-index` при перемещении элемента. Значение по умолчанию — `false`.
- `stack` — устанавливает автоматический контроль значения CSS-атрибута `z-index` для группы элементов. При использовании этой опции перемещаемый элемент всегда будет расположен над другими элементами в группе. В качестве значения указывается объект с двумя свойствами:

◊ `group` — ссылка на группу элементов (указывается селектор jQuery);

◊ `min` — минимальное значение CSS-атрибута `z-index`.

Рассмотрим пример.

```
stack: { group: "#div1 div", min: 1 }
```

Значение по умолчанию — `false`.

- `handle` — позволяет указать один или несколько элементов внутри блока, захватив которые можно переместить весь блок. В качестве значения указывается селектор jQuery или ссылка на элемент. Значение по умолчанию — `false` (переместить блок можно, захватив любой элемент, кроме указанных в опции `cancel`).
- `cancel` — позволяет указать один или несколько элементов внутри блока, захватив которые нельзя будет переместить весь блок. В качестве значения указывается селектор jQuery. Значение по умолчанию — ":input, option";
- `scope` — ограничивает область видимости при “сбрасывании” элементов. Если в элементе-цели указана другая область видимости, то событие `drop` не сработает. В качестве значения указывается произвольная строка. Применяется при использовании модуля UI Draggable. Значение по умолчанию — `"default"`.
- `refreshPositions` — если указано значение `true`, то позиции будут вычисляться постоянно в процессе передвижения элемента с помощью мыши. Обратите внимание на то, что эта опция может резко снизить эффективность. Применяется при использовании модуля UI Draggable. Значение по умолчанию — `false`.
- `iframeFix` — если указано значение `true`, то это позволит решить проблему, возникающую в некоторых браузерах при перемещении элемента над содержимым элементов `IFRAME`. В качестве значения можно передать селектор jQuery. Значение по умолчанию — `false`.
- `connectToSortable` — позволяет вставить копию перемещаемого элемента одного списка в другой список, являющийся сортируемым. В качестве значения указывается селектор jQuery. Для нормальной работы необходимо, чтобы опция `helper` имела значение `"clone"`. Применяется при использовании модуля UI Sortable. Значение по умолчанию — `false`.

Пример создания перемещаемых элементов с различными вариантами настроек приведен в листинге 12.2.

### Листинг 12.2. Модуль UI Draggable, различные варианты настроек

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Draggable. Развличные варианты настроек</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<style type="text/css">
```

```

* { margin:0; padding:0; }
body { font-size:8pt; font-family:Verdana, sans-serif; }
.drag {
 position:absolute;
 width:100px; height:100px;
 padding:2px;
 border:2px solid #000000;
 text-align:center;
}
.ui-draggable {
 /* Класс, добавляемый после применения метода draggable() */
 background-color:#fff4dd;
}
.ui-draggable-dragging {
 /* Класс, добавляемый при перемещении элемента */
 background-color:#008800;
}

```

</style>

```

<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#draggable1").draggable({
 // Отключаем добавление класса ui-draggable
 addClasses: false,
 axis: "x", // Перемещение только по горизонтали
 // Возврат на прежнее место по окончании перемещения
 revert: true,
 cursor: "e-resize", // Вид курсора при перемещении
 delay: 200 // Задержка перед перемещением
 });
 $("#draggable2").draggable({
 // Перемещение в пределах родительского элемента
 containment: "parent",
 opacity: 0.3 // Прозрачность при перемещении
 });
 $("#draggable3").draggable({
 grid: [40, 60] // Привязка к сетке
 });
 $("#draggable4").draggable({ snap: true, snapTolerance: 30 });
 $("#draggable5").draggable({ snap: "#div1" });
 $("#draggable6").draggable({ snap: "#div1", snapMode: "inner" });
 // Отключение прокрутки
 $("#draggable7").draggable({ scroll: false });
 $("#draggable8").draggable({ scrollSensitivity: 100,
 scrollSpeed: 20 });
 $("#draggable9").draggable({
 helper: "clone", // Перемещение копии элемента
 // Копию вставляем в конец содержимого элемента BODY
 appendTo: "body",
 // После окончания перемещения возврат на прежнее место
 revert: true
 });
 $("#draggable10").draggable({
 cursor: "crosshair", // Вид курсора при перемещении

```

```

 cursorAt: { top: 0, left: 0 } // Местоположение курсора
 });
 $("#draggable11").draggable({ zIndex: 9000 });
 $("#draggable12, #draggable13").draggable({
 stack: { group: "#div2 div", min: 5 }
 });
 $("#draggable14").draggable({ cursor: "move", handle: "#div3" });
 $("#draggable15").draggable({ cancel: "#div3" });
});
//-->
</script>
</head>
<body>
<div style="height:1500px; width:5px;"></div>
<div id="draggable11" class="drag" style="top:200px; left:700px;">
Блок будет расположен над другими элементами при перемещении</div>
<div id="draggable1" class="drag" style="top:5px; left:50px;">
Блок можно передвинуть только по горизонтали</div>
<div id="div1" style="position:absolute; top:160px; left:50px;
width:550px; height:150px; border:2px solid #000000;">
Родительский элемент id=div1
<div id="draggable2" class="drag" style="top:15px; left:250px;">
Блок можно передвинуть только в пределах родительского элемента</div>
</div>
<div id="draggable3" class="drag"
style="top:5px; left:250px; height:130px;">
Блок можно передвинуть не менее чем на 40px по горизонтали
или не менее чем на 60px по вертикали</div>
<div id="draggable4" class="drag" style="top:350px; left:50px;">
Блок прилипнет к границе любого перемещаемого элемента</div>
<div id="draggable5" class="drag" style="top:350px; left:200px;">
Блок прилипнет к границе элемента с id=div1</div>
<div id="draggable6" class="drag" style="top:350px; left:350px;">
Блок прилипнет к границе элемента с id=div1, но только с
внутренней стороны</div>
<div id="draggable7" class="drag" style="top:5px; left:450px;">
При перемещении за рамки окна прокрутки не будет</div>
<div id="draggable8" class="drag" style="top:5px; left:600px;">
Прокрутка начнется при расстоянии 100px от края окна до
указателя мыши</div>
<div id="draggable9" class="drag" style="top:5px; left:750px;">
Перемещаться будет копия блока</div>
<div id="draggable10" class="drag" style="top:350px; left:500px;">
Блок можно перемещать только за верхний левый угол</div>
<div id="div2" style="position:absolute; top:500px; left:50px;
width:550px; height:150px; border:2px solid #000000;">Группа
<div id="draggable12" class="drag" style="top:15px; left:150px;">
Блок будет расположен над другим элементом в этой группе</div>
<div id="draggable13" class="drag" style="top:15px; left:350px;">
Блок будет расположен над другим элементом в этой группе</div>
</div>
<div id="draggable14" class="drag" style="top:350px; left:700px;">
<div id="div3" style="background-color:#ff9600;">Заголовок</div>

```

```

Блок можно переместить только с помощью заголовка</div>
<div id="draggable15" class="drag" style="top:500px; left:700px;">
<div id="div3" style="background-color:#ff9600;">Заголовок</div>

Блок с помощью заголовка переместить нельзя</div>
</body>
</html>
```

Первый формат метода `draggable()` позволяет задать значения при создании компонента. Чтобы изменить или получить значения опций уже после создания, необходимо использовать второй формат метода `draggable()`. Для этого в первом параметре указывается значение "option", во втором — название опции, а в третьем — новое значение.

```
$("#draggable").draggable("option", "axis", "y");
```

Если необходимо получить значение опции, то третий параметр указывать не нужно. Получим значение опции `axis`:

```
alert($("#draggable").draggable("option", "axis"));
```

Второй формат метода `draggable()` позволяет также управлять перемещаемым элементом. Для этого в первом параметре указываются следующие значения:

- `disable` — временно запрещает перемещение элемента;
- `enable` — разрешает перемещение элемента, если ранее оно было запрещено с помощью метода `disable`;
- `destroy` — удаляет всю функциональность и возвращает все элементы в первоначальное состояние.

Обработать события, происходящие с перемещаемым элементом, можно двумя способами.

```
draggable({
 <Событие>: <Функция обратного вызова>
});
```

или

```
bind(<Событие>, <Функция обратного вызова>)
```

В параметре `<Событие>` могут быть указаны следующие события (в скобках указано значение для метода `bind()`):

- `start (dragstart)` — наступает в начале перемещения элемента;
- `drag (drag)` — возникает постоянно в процессе перемещения;
- `stop (dragstop)` — наступает в конце перемещения элемента.

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции> ([<Объект event>[, <Объект UI>]]) {
 // ...
}
```

Через параметр `<Объект UI>` доступны следующие свойства.

- `helper` — ссылка (объект jQuery) на перемещаемый элемент.
- `position` — текущая позиция относительно родительского элемента. Доступны два следующих свойства:

- ◊ `top` — положение сверху;
- ◊ `left` — положение слева.
- `offset` — текущая позиция относительно окна. Доступны два свойства:
  - ◊ `top` — положение сверху;
  - ◊ `left` — положение слева.

Пример обработки различных событий приведен в листинге 12.3.

### Листинг 12.3. Модуль UI Draggable, обработка событий

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Draggable. Обработка событий</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<style type="text/css">
body { font-size:8pt; font-family:Verdana, sans-serif; }
.drag {
 position:absolute;
 top:10px; left:50px;
 width:100px; height:100px;
 padding:2px;
 border:2px solid #000000;
 text-align:center;
}
.ui-draggable {
 /* Класс, добавляемый после применения метода draggable() */
 background-color:#fff4dd;
}
.ui-draggable-dragging {
 /* Класс, добавляемый при перемещении элемента */
 background-color:#008800;
}
</style>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#draggable").draggable({
 start: function() {
 $("#div2").append("Событие start
");
 },
 drag: function(e, ui) {
 $("#div2").append("Событие drag
");
 var msg = "Положение:
 position - сверху: " +
 ui.position.top;
 msg += " px, слева: " + ui.position.left + " px
";
 msg += "offset - сверху: " + ui.offset.top + " px, слева: " +
 ui.offset.left + " px
";
 $("#div1").html(msg);
 }
 });
}
-->
```

```

},
stop: function(e, ui) {
 $("#div2").append("Событие stop
");
 ui.helper.html("<p>Объект остановился</p>");
}
).bind("dragstart", function() {
 $("#div2").html("Событие dragstart
");
}).bind("drag", function() {
 $("#div2").append("Событие drag (bind)
");
}).bind("dragstop", function() {
 $("#div2").append("Событие dragstop
");
});
$("#btn1").click(function() {
 $("#draggable").draggable("disable");
});
$("#btn2").click(function() {
 $("#draggable").draggable("enable");
});
});
//-->
</script>
</head>
<body>
<input type="button" value="Запретить" id="btn1">
<input type="button" value="Разрешить" id="btn2">

<div id="div1"></div><div id="div2"></div>
<div style="position:absolute; top:160px; left:250px;
width:550px; height:150px; border:2px solid #000000;">
Родительский элемент
<div id="draggable" class="drag"><p>Передвинь меня с помощью
мыши</p></div>
</div>
</body>
</html>

```

## 12.2. Модуль UI Draggable — “сбрасывание” элементов

Модуль UI Draggable позволяет обработать момент “сбрасывания” свободно перемещаемого объекта над каким-либо элементом. Для загрузки модуля переходим на страницу <http://jqueryui.com/download>. Оставляем установленными флаги UI Core, Draggable, Droppable, 1.7.2, а затем щелкаем на кнопке Download. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы следующие файлы:

- jquery-1.3.2.min.js (папка js);
- jquery-ui-1.7.2.custom.min.js (папка js).

Все указанные файлы необходимо подключить к скрипту в том порядке, в котором они перечислены. В качестве примера рассмотрим “сбрасывание” элементов и применение опции scope для ограничения области видимости (листинг 12.4).

#### Листинг 12.4. Модуль UI Droppable, “сбрасывание” элементов

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Droppable. "Сбрасывание" элементов</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#draggable1").draggable({
 helper: "clone",
 scope: "test", // Ограничение области видимости
 zIndex: 9000
 });
 $("#droppable1").droppable({
 scope: "test", // Ограничение области видимости
 drop: function() {
 alert("Попали...");
 }
 });
 $("#draggable2").draggable({ helper: "clone", zIndex: 9000 });
 $("#droppable2").droppable({
 drop: function() {
 alert("Попали...");
 }
 });
});
//--
</script>
<style type="text/css">
body { font-size:8pt; font-family:Verdana, sans-serif; }
div {
 position:absolute;
 top:10px; left:50px;
 width:100px; height:100px;
 padding:2px;
 border:2px solid #000000;
 text-align:center;
 background-color:#ffff4dd;
}
</style>
</head>
<body>
<div id="draggable1" style="top:50px;left:50px;">
Блок можно сбросить только на красный квадрат</div>
<div id="droppable1" style="top:50px;left:350px;
 background-color:red;">

<div id="draggable2" style="top:250px;left:50px;">
Блок можно сбросить только на зеленый квадрат</div>
<div id="droppable2"
```

```
</body>
</html>
```

Чтобы элемент обрабатывал “сбрасывание” объекта над ним, необходимо указать это с помощью метода `droppable()`. Формат метода:

```
droppable([<Объект с опциями>])
```

Параметр `<Объект с опциями>` представляет собой объект, состоящий из пар “опция/значение”. Могут быть указаны следующие опции.

- `scope` — ограничивает область видимости при “сбрасывании” элементов. Если в перемещаемом элементе указана другая область видимости, то событие `drop` не сработает. В качестве значения указывается произвольная строка. Значение по умолчанию — `"default"`.
- `accept` — позволяет указать элементы, которые могут быть “сброшены”. В качестве значения может быть указан селектор jQuery или ссылка на функцию. Если указана ссылка на функцию, то внутри функции необходимо вернуть значение `true`, если элемент можно “бросить”. Если в качестве параметра функции указать переменную, то через нее будет доступна ссылка (объект jQuery) на “сбрасываемый” элемент. Пример использования функции:

```
accept: function(elem) {
 if (elem.attr("id") == "draggable1") return true;
 else return false;
}
```

Пример указания селектора:

```
accept: "#draggable1"
```

- `activeClass` — если указано значение `false`, то класс `ui-droppable` не будет автоматически добавляться при инициализации. Значение по умолчанию — `true`.
- `addClasses` — стилевой класс, добавляемый к элементу при перемещении объекта, который может быть успешно “сброшен”. Значение по умолчанию — `false`.
- `hoverClass` — стилевой класс, добавляемый к элементу, если объект находится над зоной “броса” и “сбрасывание” объекта будет успешным. Значение по умолчанию — `false`.
- `tolerance` — определяет, в каком случае “сбрасывание” будет успешным. Могут быть указаны следующие значения:
  - ◊ `intersect` — если перемещаемый объект на 50% расположен над элементом (значение по умолчанию);
  - ◊ `fit` — если перемещаемый объект полностью расположен над элементом;
  - ◊ `pointer` — если указатель мыши находится над элементом;
  - ◊ `touch` — если любая точка перемещаемого объекта расположена над элементом.
- `greedy` — если указано значение `true`, то событие “сбрасывания” над вложенным элементом не будет обрабатываться родительским элементом. Значение по умолчанию — `false`.

В листинге 12.5 приведен пример, демонстрирующий возможности опции `greedy`, а также других опций.

### Листинг 12.5. Модуль UI `Droppable`, использование опции `greedy`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Droppable. Использование опции greedy</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#draggable1").draggable({
 helper: "clone", // Перемещаем копию элемента
 zIndex: 9000
 });
 $("#droppable1").droppable({
 // Сбросить можно только элемент с id=draggable1
 accept: "#draggable1",
 addClasses: false, // Отключаем добавление класса ui-droppable
 activeClass: "cls1", // Класс, добавляемый при перемещении
 // Класс, добавляемый, если сбрасывание объекта будет успешным
 hoverClass: "cls2",
 // Объект должен целиком располагаться над элементом
 tolerance: "fit",
 drop: function() {
 alert("Попали в родительский блок");
 }
 });
 $("#droppable2").droppable({
 greedy: true,
 activeClass: "cls1",
 hoverClass: "cls2",
 drop: function() {
 alert("Попали во вложенный блок");
 }
 });
 $("#btn1").click(function() {
 if ($("#droppable2").droppable("option", "greedy")) {
 $("#droppable2").droppable("option", "greedy", false)
 .html("Вложенный блок
greedy = false");
 }
 else {
 $("#droppable2").droppable("option", "greedy", true)
 .html("Вложенный блок
greedy = true");
 }
 });
//-->
```

```

</script>
<style type="text/css">
body { font-size:8pt; font-family:Verdana, sans-serif; }
div {
 position:absolute;
 width:100px; height:100px;
 padding:2px;
 border:2px solid #000000;
 text-align:center;
 background-color:#ffff4dd;
}
.cls1 { background-color:#ff0000; }
.cls2 { background-color:#008800; }
.ui-draggable-dragging {
 /* Класс, добавляемый при перемещении элемента */
 background-color:#ff9600;
}
</style>
</head>
<body>

<input type="button" value="Поменять значение опции greedy"
 id="btn1">
<div id="draggable1" style="top:150px;left:50px;">
Сбрось меня на вложенный блок</div>
<div id="droppable1"
style="top:50px;left:350px;width:250px;height:250px;">
Блок-цель
<div id="droppable2"
style="top:110px;left:20px;width:200px;height:120px;">
Вложенный блок
greedy = true</div>
</div>
</body>
</html>

```

Чтобы изменить или получить значения опций уже после создания, необходимо использовать второй формат метода `droppable()`. Для этого в первом параметре указывается значение "option", во втором — название опции, а в третьем — новое значение.

```
$("#droppable").droppable("option", "greedy", true);
```

Если необходимо получить значение опции, то третий параметр указывать не нужно. Получим значение опции `tolerance`:

```
alert($("#droppable").droppable("option", "tolerance"));
```

Второй формат метода `droppable()` позволяет также управлять элементом. Для этого в первом параметре указываются следующие значения:

- `disable` — временно запрещает обработку “сбрасывания”;
- `enable` — разрешает обработку “сбрасывания”, если ранее она была запрещена с помощью метода `disable`;
- `destroy` — удаляет всю функциональность и возвращает все элементы в первоначальное состояние.

Обработать события, происходящие с элементами, можно двумя способами.

```

droppable({
 <Событие>: <Функция обратного вызова>
});
или
bind(<Событие>, <Функция обратного вызова>)

```

В параметре *<Событие>* могут быть указаны следующие события (в скобках указано значение для метода bind()).

- *activate* (*dropactivate*) — наступает в начале перемещения элемента, который может быть “сброшен”. Элемент можно “бросить”, если он находится в одной области видимости (опция *scope*) и разрешен опцией *accept*.
- *over* (*dropover*) — возникает, когда перемещаемый объект находится в зоне “сброса”, заданной опцией *tolerance*.
- *out* (*dropout*) — возникает, когда перемещаемый объект покидает зону “сброса”, заданную опцией *tolerance*.
- *drop* (*drop*) — наступает, когда “сброс” элемента выполнен успешно.
- *deactivate* (*dropdeactivate*) — возникает, когда перемещаемый элемент был “сброшен” (вне зависимости от успешности). При успешном “сбросе” событие наступает после события *drop*.

В параметре *<Функция обратного вызова>* указывается ссылка на функцию следующего формата.

```

function <Название функции>([<Объект event>[, <Объект UI>]]) {
 // ...
}

```

Внутри функции обратного вызова доступна ссылка (*this*) на текущий элемент, который ожидает “сброса” перемещаемого объекта. Через параметр *<Объект UI>* доступны следующие свойства.

- *draggable* — ссылка (объект jQuery) на оригинал перемещаемого элемента.
- *helper* — ссылка (объект jQuery) на копию элемента (если опция *helper* равна "clone") или на сам элемент (если опция *helper* равна "original").
- *position* — текущая позиция перемещаемого объекта относительно родительского элемента. Доступны два свойства:
  - ◊ *top* — положение сверху;
  - ◊ *left* — положение слева.
- *offset* — текущая позиция перемещаемого объекта относительно окна. Доступны два свойства:
  - ◊ *top* — положение сверху;
  - ◊ *left* — положение слева.

Пример обработки различных событий приведен в листинге 12.6.

## Листинг 12.6. Модуль UI Draggable, обработка событий

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Draggable. Обработка событий</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#draggable1").draggable({ revert: "invalid", zIndex: 9000 });
 $("#draggable2").draggable({ revert: "valid", zIndex: 9000 });
 $("#draggable3").draggable({ helper: "clone", zIndex: 9000 });
 $("#droppable").droppable({
 activate: function(e, ui) {
 $("#span1").append("Событие activate
");
 ui.helper.css("background-color", "#ff9600");
 },
 deactivate: function(e, ui) {
 $("#span1").append("Событие deactivate
");
 ui.helper.css("background-color", "#fff4dd");
 $(this).css("background-color", "#fff4dd");
 },
 over: function() {
 $("#span1").append("Событие over
");
 $(this).css("background-color", "#008800");
 },
 out: function() {
 $("#span1").append("Событие out
");
 $(this).css("background-color", "#fff4dd");
 },
 drop: function(e, ui) {
 var msg = "Координаты сброса:
 position - сверху: ";
 msg += ui.position.top + " px, слева: " + ui.position.left;
 msg += " px
offset - сверху: " + ui.offset.top +
 " px, слева: ";
 msg += ui.offset.left + " px
";
 $("#span1").append("Событие drop
" + msg);
 }
 }).bind("dropactivate", function() {
 $("#span1").html("Событие dropactivate
");
 }).bind("dropdeactivate", function() {
 $("#span1").append("Событие dropdeactivate
");
 }).bind("dropover", function() {
 $("#span1").append("Событие dropover
");
 }).bind("dropout", function() {
 $("#span1").append("Событие dropout
");
 }).bind("drop", function() {
 $("#span1").append("Событие drop (bind)
");
 });
});
```

```

 $("#btn1").click(function() {
 $("#droppable").droppable("disable");
 });
 $("#btn2").click(function() {
 $("#droppable").droppable("enable");
 });
});
//-->
</script>
<style type="text/css">
body { font-size:8pt; font-family:Verdana, sans-serif; }
div {
 position:absolute; width:100px; height:100px; padding:2px;
 border:2px solid #000000; text-align:center;
 background-color:#fff4dd;
}
</style>
</head>
<body>
<input type="button" value="Запретить" id="btn1">
<input type="button" value="Разрешить" id="btn2">

<div id="draggable1" style="top:50px;left:600px;z-index:5">
Перемещаемый блок 1
revert = invalid</div>
<div id="draggable2" style="top:200px;left:600px;;z-index:5">
Перемещаемый блок 2
revert = valid</div>
<div id="draggable3" style="top:350px;left:600px;;z-index:5">
Перемещаемый блок 3
helper = clone</div>
<div id="droppable"
style="top:50px;left:250px;width:250px;height:250px;">
Блок-цель</div>
</body>
</html>

```

## 12.3. Модуль UI Sortable — сортировка элементов

Модуль UI Sortable позволяет сортировать элементы с помощью мыши. Для загрузки модуля переходим на страницу <http://jqueryui.com/download>. Оставляем установленными флагки UI Core, Draggable, Sortable, 1.7.2, а затем щелкаем на кнопке Download. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы следующие файлы:

- jquery-1.3.2.min.js (папка js);
- jquery-ui-1.7.2.custom.min.js (папка js).

Все указанные файлы необходимо подключить к скрипту в том порядке, в котором они перечислены. В качестве примера рассмотрим функциональные возможности модуля с параметрами, принятыми по умолчанию, а также применение опции connectToSortable (листинг 12.7).

### Листинг 12.7. Модуль UI Sortable, сортировка элементов списка

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Sortable. Сортировка элементов списка</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js"
type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
type="text/javascript"></script>
<style type="text/css">
body { font-size:10pt; font-family:Verdana, sans-serif; }
ul { margin:0; padding:0; list-style-type:none; }
li { margin: 0 3px 3px 20px; padding: 5px 5px 5px 20px;
width:200px;
 font-size:16px; font-family:Verdana, sans-serif; height:18px;
 border: 1px solid #000000; color:#000000; font-weight:bold; }
.itemColor1 { background-color:#fff4dd; }
.itemColor2 { background-color:#d5dee7; }
</style>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#sortable").sortable().disableSelection();
 $("#draggable li").draggable({
 helper: "clone", // Перемещение копии элемента
 // Пункты можно добавить в список с id=sortable
 connectToSortable: "#sortable"
 }).disableSelection(); // Запрещаем выделение текста пунктов
});
//-->
</script>
</head>
<body>
Сортируемый список

<ul id="sortable">
<li class="itemColor1">Пункт 1
<li class="itemColor1">Пункт 2
<li class="itemColor1">Пункт 3
<li class="itemColor1">Пункт 4

Из этого списка можно добавить пункты
в сортируемый список

<ul id="draggable">
<li class="itemColor2">Пункт 1
<li class="itemColor2">Пункт 2
<li class="itemColor2">Пункт 3
<li class="itemColor2">Пункт 4

</body>
</html>
```

Для того чтобы элементы можно было сортировать с помощью мыши, необходимо применить метод `sortable()`. Формат метода:

```
sortable([<Объект с опциями>])
```

Параметр `<Объект с опциями>` представляет собой объект, состоящий из пар “опция/значение”. Могут быть указаны следующие опции.

- `items` — определяет, какие элементы должны быть сортируемыми. В качестве значения указывается селектор jQuery. Значение по умолчанию — "`> *`".
- `axis` — позволяет ограничить перемещение элемента. Могут быть указаны следующие значения:
  - ◊ `x` — перемещение только по горизонтали;
  - ◊ `y` — перемещение только по вертикали;
  - ◊ `false` — без ограничения (значение по умолчанию).
- `revert` — если указано значение `true`, то после “сброса” элемента перемещение на новое место будет происходить с анимацией. Значение по умолчанию — `false`.
- `containment` — позволяет ограничить перемещение в пределах указанных границ. В качестве значения можно указать DOM-элемент, селектор jQuery или одно из значений:
  - ◊ `parent` — в пределах родительского элемента;
  - ◊ `window` — в пределах окна;
  - ◊ `document` — в пределах всего документа.

Значение по умолчанию — `false`.

- `opacity` — задает степень прозрачности элемента во время перемещения. Может быть указано вещественное число от 0.01 до 1. Значение по умолчанию — `false`.
- `cursor` — вид курсора при перемещении элемента. Указывается одно из значений атрибута `cursor` в CSS. Значение по умолчанию — `"auto"`.
- `cursorAt` — задает местоположение курсора при перемещении элемента. В качестве значения указывается объект с комбинацией следующих свойств:
  - ◊ `top` — положение относительно верхней границы;
  - ◊ `left` — положение относительно левой границы;
  - ◊ `right` — положение относительно правой границы. При положительных значениях курсор смещается внутрь элемента, а при отрицательных значениях — наружу от элемента справа;
  - ◊ `bottom` — положение относительно нижней границы. При положительных значениях курсор смещается внутрь элемента, а при отрицательных значениях смещается от элемента вниз.

Значение по умолчанию — `false` (при перемещении элемента курсор располагается в том месте, где был произведен щелчок мышью).

Рассмотрим несколько примеров. Во время перемещения курсор будет находиться над верхней границей элемента.

```
cursorAt: { top: 0 }
```

Курсор будет расположен в левом верхнем углу.

```
cursorAt: { top: 0, left: 0 }
```

Курсор будет расположен на одном уровне с верхней границей, но сдвинут на 5 px от элемента относительно правой границы.

```
cursorAt: { top: 0, right: -5 }
```

Курсор будет смещен на 5 px ниже нижней границы и на 5 px от левой границы элемента.

```
cursorAt: { bottom: -5, left: -5 }
```

### Обратите внимание

Нельзя одновременно указывать свойства `top` и `bottom`, а также `left` и `right`.

- `delay` — позволяет задать задержку в миллисекундах перед началом перемещения элемента. Используется для предотвращения нежелательного перемещения при случайном щелчке на элементе. Значение по умолчанию — 0.
- `distance` — расстояние в пикселях, после прохождения которого начинается перемещение элемента. Используется для предотвращения нежелательного перемещения при случайном щелчке на элементе. Значение по умолчанию — 1.
- `grid` — устанавливает шаг перемещения по горизонтали и вертикали (в пикселях). В качестве значения указывается массив с двумя элементами.

[`<Шаг по горизонтали>`, `<Шаг по вертикали>`]

Рассмотрим пример.

```
grid: [40, 20]
```

Значение по умолчанию — `false`.

- `scroll` — если указано значение `false`, то перемещение элемента за границы окна не будет приводить к автоматической прокрутке документа. Значение по умолчанию — `true`.
- `scrollSensitivity` — расстояние (в пикселях) от края окна, при котором будет происходить прокрутка документа. Обратите внимание на то, что расстояние отсчитывается от указателя мыши, а не от границы перемещаемого элемента. Значение по умолчанию — 20.
- `scrollSpeed` — скорость прокрутки документа. Значение по умолчанию — 20.
- `helper` — позволяет указать элемент, который будет перемещаться. Могут быть указаны следующие значения:
  - ◊ `original` — перемещается сам элемент (значение по умолчанию);
  - ◊ `clone` — перемещается копия элемента.

В качестве значения может быть указана ссылка на функцию. В этом случае функция должна возвращать DOM-элемент.

- `appendTo` — ссылка на элемент-контейнер, в конец которого будет вставляться элемент, указанный в опции `helper` или если значение опции `helper` равно `"clone"`. Значение по умолчанию — `"parent"` (для вставки используется родительский элемент).
- `zIndex` — значение CSS-атрибута `z-index` при перемещении элемента. Значение по умолчанию — `1000`.
- `tolerance` — определяет, в каком случае “сбрасывание” будет успешным. Могут быть указаны следующие значения:
  - ◊ `intersect` — если перемещаемый элемент на `50%` расположен над новой позицией (значение по умолчанию);
  - ◊ `pointer` — если указатель мыши находится над новой позицией.
- `handle` — позволяет указать один или несколько элементов внутри блока, захватив которые можно переместить весь блок. В качестве значения указывается селектор jQuery или ссылка на элемент. Значение по умолчанию — `false` (т.е. переместить можно, захватив любой элемент, кроме указанных в опции `cancel`).
- `cancel` — позволяет указать один или несколько элементов внутри блока, захватив которые нельзя будет переместить весь блок. В качестве значения указывается селектор jQuery. Значение по умолчанию — ":input, option".
- `placeholder` — позволяет указать стилевой класс для элемента, служащего “заглушкой” при перемещении пункта списка. Значение по умолчанию — `false`. Чтобы понять смысл этой опции, рассмотрим все на примере. Итак, например, есть список с двумя элементами.

```
<ul id="sortable">
<li class="itemColor">Пункт 1
<li class="itemColor">Пункт 2

```

В начале перемещения второго пункта исходный HTML-код списка будет выглядеть следующим образом.

```
<ul id="sortable" class="ui-sortable" unselectable="on">
<li class="itemColor">Пункт 1
<li class="itemColor" style="width:200px;height:18px;
position:absolute;z-index:1000">Пункт 2
<li class="itemColor ui-sortable-placeholder"
style="visibility:hidden">

```

Как видно из примера, перемещаемый элемент получает абсолютное позиционирование, а на его место вставляется скрытая “заглушка”.

```
<li class="itemColor ui-sortable-placeholder"
style="visibility:hidden">
```

Если в опции `placeholder` указан стилевой класс (например, `cls1`), то исходный код “заглушки” будет выглядеть так.

```
<li class="cls1">
```

- `forcePlaceholderSize` — если указано значение `true`, то для “заглушки” будет указана высота. Например, при параметрах

```
placeholder: "cls1",
forcePlaceholderSize: true
```

исходный код “заглушки” будет выглядеть так,

```
<li class="cls1" style="height:18px">
```

Значение по умолчанию — `false`.

- `connectWith` — позволяет указать ссылку на сортируемый список, в который можно переместить элементы текущего списка. В качестве значения указывается селектор jQuery. Значение по умолчанию — `false` (элементы между списками переместить нельзя).
- `dropOnEmpty` — если указано значение `false`, то элементы из списка в другой пустой список добавить нельзя. Значение по умолчанию — `true`.

Пример создания сортируемых списков с различными вариантами параметров настройки приведен в листинге 12.8.

### Листинг 12.8. Модуль UI Sortable, различные варианты параметров настройки

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Sortable. Различные варианты настроек</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<style>
body { font-size:10pt; font-family:Verdana, sans-serif; }
ul { margin:0; padding:0; list-style-type:none; }
li { margin: 0 3px 3px 20px; padding: 5px 5px 5px 20px;
width:200px;
 font-size:16px; font-family:Verdana, sans-serif; height:18px;
 border: 1px solid #000000; color:#000000; font-weight:bold; }
.itemColor1 { background-color:#fff4dd; }
.itemColor2 { background-color:#d5dee7; }
.cls1 { border: dotted 1px black; background-color:#ffe9b3; }
</style>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#sortable1").sortable({
 items: "li", // Перемещаем элементы LI
 axis: "y", // Перемещение только по вертикали
 cursor: "s-resize", // Вид курсора при перемещении
 cancel: "span", // За элемент SPAN переместить нельзя
 // Куда можно переместить
 connectWith: "#sortable2,#sortable3",
 delay: 200 // Задержка перед перемещением
 }).disableSelection();
-->
```

184

Глава 12

```
$("#sortable2").sortable({
 revert: true, // Перемещение с анимацией
 containment: "window", // Перемещение в пределах окна
 opacity: 0.5, // Степень прозрачности
 helper: "clone", // Перемещение копии пункта списка
 placeholder: "cls1", // Класс для "заглушки"
 // Куда можно переместить
 connectWith: "#sortable1,#sortable3",
 // Добавить элементы в пустой список нельзя
 dropOnEmpty: false,
 cursorAt: { top: 0 } // Перемещение за верхнюю границу
}).disableSelection();
$("#sortable3").sortable({
 // Куда можно переместить
 connectWith: "#sortable1,#sortable2"
}).disableSelection();
});
//-->
</script>
</head>
<body>
Сортируемый список 1

За красный прямоугольник переместить элемент нельзя

<ul id="sortable1">
<li class="itemColor1">Пункт 1
&nbsp&nbsp&nbsp&nbsp

<li class="itemColor1">Пункт 2
<li class="itemColor1">Пункт 3
<li class="itemColor1">Пункт 4

Сортируемый список 2

<ul id="sortable2">
<li class="itemColor2">Пункт 1
<li class="itemColor2">Пункт 2
<li class="itemColor2">Пункт 3
<li class="itemColor2">Пункт 4

Сортируемый список 3

Добавить элементы в этот пустой список можно только из списка
1

<ul style="background-color:#fff4dd;margin:5px;padding:5px;
width:250px;">
<li id="sortable3">

</body>
</html>
```

Чтобы изменить или получить значения опций уже после создания, необходимо использовать второй формат метода `sortable()`. Для этого в первом параметре указывается значение "option", во втором — название опции, а в третьем — новое значение.

```
$("#sortable").sortable("option", "delay", 200);
```

Если необходимо получить значение опции, то третий параметр указывать не нужно. В качестве примера получим значение опции `delay`:

```
alert($("#sortable").sortable("option", "delay"));
```

Второй формат метода `sortable()` позволяет также управлять элементом. Для этого в первом параметре указываются следующие значения.

- `serialize` — получает позиции всех сортируемых элементов и составляет строку, которую можно отправить на сервер с помощью технологии AJAX. Формат метода:

```
sortable("serialize", <Объект с опциями>)
```

Если параметр `<Объект с опциями>` не указан, то по умолчанию строка составляется из значений параметров `id`. Для этого идентификатор элемента должен быть в следующем формате.

`<Название>_<Число>`

Например:

```
id="item_1"
```

В качестве разделителя, вместо символа подчеркивания, можно использовать знак равенства (=) или дефис (-). Если идентификаторы равны `item_1`, `item_2`, `item_3`, то в результате получим следующую строку.

```
item[] =1&item[] =2&item[] =3
```

В параметре `<Объект с опциями>` можно дополнительно указать объект со следующими свойствами.

- ◊ `attribute` — параметр, из значений которого будет составляться строка. Значение по умолчанию — "id".
- ◊ `key` — название, которое будет использоваться, например, вместо `item[]` при значении параметра "item\_1".
- ◊ `expression` — позволяет указать регулярное выражение (объект `RegExp`) для разбора значения параметра, указанного в свойстве `attribute`. Значение по умолчанию — `/(.+)[-=_](.+)/`. Следует учитывать, что при одновременном использовании свойств `expression` и `key` в регулярном выражении должна быть только одна подмаска, в противном случае — две.

Рассмотрим пример использования параметра `<Объект с опциями>`.

```
$("#sortable").sortable("serialize", {
 attribute: "id",
 key: "myParam[]",
 expression: /(_.+)/
});
```

Если идентификаторы равны `item_1`, `item_2`, `item_3`, то в результате получим следующую строку.

```
myParam[] =1&myParam[] =2&myParam[] =3
```

- `toArray` — возвращает массив значений всех идентификаторов сортируемых элементов. Нумерация начинается с 0. Формат метода:

```
sortable("toArray" [, <Объект с опциями>])
```

Если идентификаторы равны item\_1, item\_2, item\_3, то в результате выполнения кода

```
var arr = $("#sortable").sortable("toArray");
var msg = "";
for (var i=0, j=arr.length; i<j; i++)
 msg += "[" + i + "] => " + arr[i] + "\n";
alert(msg);
```

получим следующий массив.

```
[0] => item_1
[1] => item_2
[2] => item_3
```

В параметре <Объект с опциями> с помощью свойства attribute можно указать атрибут тега, значения которого необходимо вернуть.

```
var arr = $("#sortable").sortable("toArray", {attribute: "id"});
```

- cancel — позволяет отменить последнюю сортировку.
- disable — временно запрещает сортировку элементов.
- enable — разрешает сортировку элементов, если ранее она была запрещена с помощью метода disable.
- destroy — удаляет всю функциональность и возвращает все элементы в первоначальное состояние.

Обработать события, происходящие с сортируемым списком, можно двумя способами.

```
sortable({
 <Событие>: <Функция обратного вызова>
});
```

или

```
bind(<Событие>, <Функция обратного вызова>)
```

В параметре <Событие> могут быть указаны следующие события (в скобках указано значение для метода bind()).

- start (sortstart) — наступает в начале сортировки.
- activate (sortactivate) — возникает в начале перемещения элемента любого связанного списка.
- sort (sort) — срабатывает постоянно в процессе сортировки.
- out (sortout) — наступает, когда элемент покидает пределы списка или в конце перемещения.
- over (sortover) — возникает, когда элемент возвращается в пределы списка или только начал движение внутри списка.
- change (sortchange) — наступает в процессе сортировки, когда позиция элемента внутри списка изменилась.
- beforeStop (sortbeforeStop) — возникает в конце сортировки, но при данном событии еще доступны элемент “заглушка” и перемещаемый элемент (копия элемента, если опция helper имеет значение "clone").

- `receive (sortreceive)` — наступает при добавлении элемента из другого списка.
- `update (sortupdate)` — возникает после события `beforeStop`, если позиция элемента изменилась (если позиция осталась прежней, то событие не возникает).
- `remove (sortremove)` — наступает, когда пункт списка был перемещен в другой список.
- `deactivate (sortdeactivate)` — возникает в конце перемещения элемента любого связанного списка.
- `stop (sortstop)` — наступает в конце сортировки.

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>[, <Объект UI>]]) {
 // ...
}
```

Через параметр `<Объект UI>` доступны следующие свойства.

- `item` — ссылка (объект `jQuery`) на оригинал перемещаемого элемента.
- `helper` — ссылка (объект `jQuery`) на копию элемента (если опция `helper` равна "clone") или на сам элемент (если опция `helper` равна "original").
- `placeholder` — ссылка (объект `jQuery`) на элемент "заглушка".
- `sender` — ссылка (объект `jQuery`) на связанный список, из которого перемещается элемент. Если элемент перемещается внутри одного списка, то объект не доступен.
- `position` — текущая позиция перемещаемого объекта относительно родительского элемента. Доступны два свойства:
  - ◊ `top` — положение сверху;
  - ◊ `left` — положение слева.
- `offset` — текущая позиция перемещаемого объекта относительно окна. Доступны два свойства:
  - ◊ `top` — положение сверху;
  - ◊ `left` — положение слева.

Пример обработки различных событий приведен в листинге 12.9.

### **Листинг 12.9. Модуль UI Sortable, обработка событий**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Sortable. Обработка событий</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<style type="text/css">
div { margin: 0; padding: 0; }
```

```

body { font-size:10pt; font-family:Verdana, sans-serif; }
h2 { background-color: #ffe9b3; font-size: 12pt;
 padding: 3px 3px 3px 10px; margin: 0 0 5px 0;
}
.section {
 margin: 0 0 15px 5px; width: 300px; height: 100px;
 background-color: #ffffbf; border: 1px solid #000000;
 color: #000000; font-weight: bold;
}
#sections, #sections2 { width: 400px; }
</style>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#sections").sortable({
 items: "div", // Сортируемые элементы
 handle: "h2", // Перемещение только за заголовок
 revert: true, // Перемещение с анимацией
 connectWith: "#sections2", // Куда можно переместить
 start: function(e, ui) {
 $("#div1").html("Событие start
");
 ui.helper.find("h2").css("background-color", "#ff0000");
 },
 sort: function() { $("#div1").append("Событие sort
"); },
 change: function() { $("#div1").append("Событие change
"); },
 beforeStop: function(e, ui) {
 $("#div1").append("Событие beforeStop
");
 ui.helper.find("h2").css("background-color", "#ffe9b3");
 },
 stop: function() { $("#div1").append("Событие stop
"); },
 update: function() { $("#div1").append("Событие update
"); },
 receive: function(e, ui) {
 $("#div1").append("Событие receive
");
 ui.item.find("p").html("Элемент перемещен
 из другого списка");
 },
 remove: function() { $("#div1").append("Событие remove
"); },
 over: function() { $("#div1").append("Событие over
"); },
 out: function() { $("#div1").append("Событие out
"); },
 activate: function() {
 $("#div1").append("Событие activate
"); },
 deactivate: function() {
 $("#div1").append("Событие deactivate
"); }
 });
 $("#sections2").sortable({
 items: "div", // Сортируемые элементы
 handle: "h2", // Перемещение только за заголовок
 connectWith: "#sections" // Куда можно переместить
 });
 $("#btn1").click(function() {
 $("#sections").sortable("disable");
 });
 $("#btn2").click(function()

```

```

 { $("#sections").sortable("enable"); });
$("#btn3").click(function() { $("#sections").sortable
("cancel"); });
$("#btn4").click(function() {
 alert($("#sections").sortable("serialize", { attribute: "id",
 key: "myParam[]", expression: /(.+)/ }));
 var arr = $("#sections").sortable("toArray",
 { attribute: "id" });
 var msg = "";
 for (var i=0, count=arr.length; i<count; i++)
 msg += "[" + i + "] => " + arr[i] + "\n";
 alert(msg);
});
});
```

//-->

```

</script>
</head>
<body>
<input type="button" value="Запретить" id="btn1">
<input type="button" value="Разрешить" id="btn2">
<input type="button" value="Отменить сортировку списка 1" id="btn3">
<input type="button" value="Положение элементов списка 1" id="btn4">

Переместить элементы можно только за заголовок

<table border="0"><tr><td valign="top">
<div id="sections">
<div class="section" id="item_1">
<h2>Секция 1</h2><p> Содержимое секции 1</p></div>
<div class="section" id="item_2">
<h2>Секция 2</h2><p> Содержимое секции 2</p></div>
<div class="section" id="item_3">
<h2>Секция 3</h2><p> Содержимое секции 3</p></div>
</div>
</td><td valign="top">
<div id="sections2">
<div class="section" id="item_4">
<h2>Секция 4</h2><p> Содержимое секции 4</p></div>
<div class="section" id="item_5">
<h2>Секция 5</h2><p> Содержимое секции 5</p></div>
<div class="section" id="item_6">
<h2>Секция 6</h2><p> Содержимое секции 6</p></div>
</div></td><td valign="top">
<div id="div1"></div>
</td></tr></table>
</body>
</html>

```

## 12.4. Модуль UI Selectable — выделение элементов

Модуль UI Selectable позволяет выделять элементы с помощью мыши. Если при выделении удерживать нажатой клавишу **<Ctrl>**, то можно добавить новые элементы к уже выделенным. Для загрузки модуля переходим на страницу <http://jqueryui.com/>

download. Оставляем установленными флагки **UI Core**, **Selectable**, 1.7.2, а затем щелкаем на кнопке **Download**. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы следующие файлы:

- jquery-1.3.2.min.js (папка js);
- jquery-ui-1.7.2.custom.min.js (папка js).

Все указанные файлы необходимо подключить к скрипту в том порядке, в котором они перечислены. В качестве примера рассмотрим функциональные возможности модуля с параметрами, принятыми по умолчанию (листинг 12.10).

#### Листинг 12.10. Модуль UI Selectable, выделение элементов

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Selectable. Выделение элементов</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<style type="text/css">
body { font-size:10pt; font-family:Verdana, sans-serif; }
ul { margin:0; padding:15px; list-style-type:none; }
li { margin: 0 3px 3px 20px; padding: 5px 5px 5px 20px;
width:300px;
 font-size:16px; font-family:Verdana, sans-serif; height:18px;
 border: 1px solid #000000; color:#000000; font-weight:bold; }
.ui-selecting { background-color: #ffe9b3; }
.ui-selected { background-color: #ff9600; color: #ffffff; }
</style>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#selectable").selectable();
});
//-->
</script>
</head>
<body>
Нажмите левую кнопку мыши над элементом и, не отпуская ее, выделяйте
элементы.

Удерживайте нажатой клавишу Ctrl, чтобы добавить новые
элементы.

<ul id="selectable">
Пункт 1
Пункт 2
Пункт 3
Пункт 4
Пункт 5
Пункт 6
Пункт 7
Пункт 8
```

```

</body>
</html>
```

Для того чтобы элементы можно было выделять с помощью мыши, необходимо применить метод `selectable()`. Формат метода:

```
selectable([<Объект с опциями>])
```

Параметр `<Объект с опциями>` представляет собой объект, состоящий из пар “опция/значение”. Могут быть указаны следующие опции.

- `filter` — позволяет указать элементы, которые могут быть выделены. В качестве значения указывается селектор jQuery. Значение по умолчанию — `"*"`.
- `cancel` — позволяет указать элементы, над которыми нельзя начать выделение. В качестве значения указывается селектор jQuery. Значение по умолчанию — ":input, option".
- `tolerance` — определяет, в каком случае выделение будет успешным. Могут быть указаны следующие значения:
  - ◊ `touch` — если любая точка выделяемого элемента попадает в область выделения (значение по умолчанию);
  - ◊ `fit` — если выделяемый элемент полностью попадает в область выделения.
- `delay` — позволяет задать задержку в миллисекундах перед началом выделения. Используется для предотвращения нежелательного выделения при случайном щелчке на элементе. Значение по умолчанию — 0.
- `distance` — расстояние в пикселях, после прохождения которого начинается выделение. Используется для предотвращения нежелательного выделения при случайном щелчке на элементе. Значение по умолчанию — 0.
- `autoRefresh` — если указано значение `false`, то положение и размер каждого выделяемого элемента автоматически пересчитываться не будет. Значение по умолчанию — `true`.

Первый формат метода `selectable()` позволяет задать значения при инициализации. Чтобы изменить или получить значения опций уже после инициализации, необходимо использовать второй формат метода `selectable()`. Для этого в первом параметре указывается значение `"option"`, во втором — название опции, а в третьем — новое значение.

```
$("#selectable").selectable("option", "tolerance", "fit");
```

Если необходимо получить значение опции, то третий параметр указывать не нужно. В качестве примера получим значение опции `tolerance`:

```
alert($("#selectable").selectable("option", "tolerance"));
```

Второй формат метода `selectable()` позволяет также управлять выделением. Для этого в первом параметре указываются следующие значения.

- `refresh` — позволяет обновить положение и размер каждого выделяемого элемента; применяется, если опция `autoRefresh` имеет значение `false`.
- `disable` — временно запрещает выделение.

- `enable` — разрешает выделение, если ранее оно было запрещено с помощью метода `disable`.
- `destroy` — удаляет всю функциональность и возвращает все элементы в первоначальное состояние.

Обработать события, происходящие с перемещаемым элементом, можно двумя способами.

```
selectable({
 <Событие>: <Функция обратного вызова>
});
```

или

```
bind(<Событие>, <Функция обратного вызова>)
```

В параметре `<Событие>` могут быть указаны следующие события (в скобках указано значение для метода `bind()`).

- `start (selectablestart)` — наступает в начале выделения.
- `selecting (selectableselecting)` — возникает, когда элемент попадает в область выделения.
- `unselecting (selectableunselecting)` — наступает, когда элемент покидает область выделения.
- `selected (selectableselected)` — возникает в конце операции выделения для каждого элемента, попавшего в область выделения.
- `unselected (selectableunselected)` — наступает при удалении элемента из набора выделенных элементов.
- `stop (selectablestop)` — возникает в конце выделения.

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>[, <Объект UI>]]) {
 // ...
}
```

Через параметр `<Объект UI>` доступны следующие свойства.

- `selecting` — ссылка на DOM-элемент, который попал в область выделения. Свойство доступно только при событии `selecting`.
- `selected` — ссылка на DOM-элемент, который был выделен. Свойство доступно только при событии `selected`.
- `unselecting` — ссылка на DOM-элемент, который покинул область выделения. Свойство доступно только при событии `unselecting`.
- `unselected` — ссылка на DOM-элемент, который был удален из набора выделенных элементов. Свойство доступно только при событии `unselected`.

При выделении элементов к ним добавляется стилевой класс `ui-selected`. Благодаря этому классу можно получить все выделенные элементы. В качестве примера получим индексы всех выделенных пунктов внутри списка.

```
$("#selectable li.ui-selected").each(function() {
 alert($("#selectable li").index(this));
});
```

Пример обработки различных событий приведен в листинге 12.11.

### Листинг 12.11. Модуль UI Selectable, обработка событий

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Selectable. Обработка событий</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<style type="text/css">
body { font-size:10pt; font-family:Verdana, sans-serif; }
ul { margin:0; padding:15px; list-style-type:none; }
.ui-selectee {
 /* Класс, добавляемый при инициализации */
 margin: 0 3px 3px 20px; padding: 5px 5px 5px 20px; width:300px;
 font-size:16px; font-family:Verdana, sans-serif; height:18px;
 border: 1px solid #000000; color:#000000; font-weight:bold;
}
.ui-selecting {
 /* Класс, добавляемый, если элемент попал в область выделения */
 background-color: #ffe9b3;
}
.ui-selected {
 /* Класс, добавляемый, если элемент был выделен */
 background-color: #ff9600; color: #ffffff;
}
</style>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#selectable").selectable({
 filter: "li", // Выделяемые элементы
 cancel: "#id3", // Над элементом с id=id3 нельзя начать
 // выделение
 tolerance: "touch",
 start: function() { $("#div1").append("Событие start
"); },
 selected: function() {$("#div1").append("Событие
 selected
"); },
 selecting: function() {
 $("#div1").append("Событие selecting
"); },
 unselected: function() {
 $("#div1").append("Событие unselected
"); },
 });
},
```

```

unselecting: function() {
 $("#div1").append("Событие unselecting
");
},
stop: function() {
 $("#div1").append("Событие stop
");
 // Выводим идентификаторы всех выделенных пунктов
 var elem = $("#selectedItems").empty();
 var sel = $("#selectable li.ui-selected");
 if (sel.size() > 0)
 sel.each(function() { elem.append(" " + this.id); });
 else elem.append("Нет");
}
}).bind("selectablestart", function() {
 $("#div1").html("Событие selectablestart
");
}),
$("#btn1").click(function() {
 $("#selectable").selectable("disable");
});
$("#btn2").click(function() {
 $("#selectable").selectable("enable");
});
});
//-->
</script>
</head>
<body>
<input type="button" value="Запретить" id="btn1">
<input type="button" value="Разрешить" id="btn2">

ID выделенных элементов: Нет

<ul id="selectable">
<li id="id1">Пункт 1
<li id="id2">Пункт 2
<li id="id3">Пункт 3
<li id="id4">Пункт 4
<li id="id5">Пункт 5
<li id="id6">Пункт 6

<div id="div1"></div>
</body>
</html>

```

## 12.5. Модуль UI Resizable — изменение размеров

Модуль UI Resizable позволяет изменять размеры элементов с помощью мыши. Для загрузки модуля переходим на страницу <http://jqueryui.com/download>. Оставляем установленными флагки UI Core, Resizable, 1.7.2, выбираем тему из списка (например, UI lightness), а затем щелкаем на кнопке Download. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы следующие файлы:

- jquery-ui-1.7.2.custom.css и каталог images (расположены в папке css\ui-lightness);

- `jquery-1.3.2.min.js` (расположен в папке `js`);
- `jquery-ui-1.7.2.custom.min.js` (расположен в папке `js`).

Все указанные файлы необходимо подключить к скрипту в том порядке, в котором они перечислены. В качестве примера рассмотрим функциональные возможности модуля с параметрами, принимаемыми по умолчанию (листинг 12.12).

### **Листинг 12.12. Модуль UI Resizable, изменение размеров**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Resizable. Изменение размеров</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/ui-lightness/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<style type="text/css">
body { font-size:10pt; font-family:Verdana, sans-serif; }
#resizable { width:300px; height:150px; padding:15px; }
</style>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#resizable").resizable();
});
//-->
</script>
</head>
<body>
<div id="resizable" class="ui-widget-content">
 Размеры этого блока можно изменять с помощью мыши.</div>
</body>
</html>
```

Для того чтобы можно было изменять размеры элемента, необходимо указать это с помощью метода `resizable()`. После применения метода исходный HTML-код элемента будет выглядеть так.

```
<div id="resizable" class="ui-widget-content ui-resizable">
 Размеры этого блока можно изменять с помощью мыши.
 <div class="ui-resizable-handle ui-resizable-e" unselectable="on">
 </div>
 <div class="ui-resizable-handle ui-resizable-s" unselectable="on">
 </div>
 <div class="ui-resizable-handle ui-resizable-se ui-icon
ui-icon-gripsmall-diagonal-se" style="z-index: 1001"
unselectable="on"></div>
</div>
```

Метод `resizable()` имеет следующий формат:

```
resizable([<Объект с опциями>])
```

Параметр `<Объект с опциями>` представляет собой объект, состоящий из пар “опция/значение”. Могут быть указаны следующие опции.

- `maxHeight` — максимально возможная высота элемента при изменении размера; значение по умолчанию — `null`.
- `maxWidth` — максимально возможная ширина элемента при изменении размера; значение по умолчанию — `null`.
- `minHeight` — минимально возможная высота элемента при изменении размера; значение по умолчанию — `10`.
- `minWidth` — минимально возможная ширина элемента при изменении размера; значение по умолчанию — `10`.
- `delay` — позволяет задать задержку в миллисекундах перед началом изменения размера. Используется для предотвращения нежелательного изменения размера при случайном щелчке на элементе. Значение по умолчанию — `0`.
- `distance` — расстояние в пикселях, после прохождения которого начинается изменение размера. Используется для предотвращения нежелательного изменения размера при случайном щелчке на элементе. Значение по умолчанию — `1`.
- `aspectRatio` — позволяет задать пропорциональное изменение размеров элемента. Если указано значение `true`, то размер будет изменяться пропорционально первоначальным размерам. В качестве значения можно также указать вещественное число (например, `2.0`) или пропорцию (например, `2/1`). Значение по умолчанию — `false`.
- `grid` — устанавливает шаг изменения размера по горизонтали и вертикали (в пикселях). В качестве значения указывается массив с двумя элементами.  
[`<Шаг по горизонтали>`, `<Шаг по вертикали>`]

Рассмотрим пример.

```
grid: [40, 60]
```

Значение по умолчанию — `false`.

- `containment` — позволяет ограничить изменение размера в пределах указанных границ. В качестве значения можно указать DOM-элемент, селектор jQuery или одно из строковых значений:
  - ◊ `parent` — в пределах родительского элемента;
  - ◊ `document` — в пределах всего документа.
- `helper` — позволяет указать стилевой класс для вспомогательного элемента, который будет изменять размеры вместо оригинала. По окончании изменения размера оригинал примет размеры вспомогательного элемента; значение по умолчанию — `false`.
- `ghost` — если указано значение `true`, то в качестве вспомогательного элемента будет использоваться полупрозрачная копия оригинала. По окончании изменения размера оригинал примет размеры вспомогательного элемента; значение по умолчанию — `false`.

- `animate` — если указано значение `true`, то изменение размеров будет произвольиться с анимацией; значение по умолчанию — `false`.
- `animateDuration` — позволяет задать продолжительность анимации. Можно указать количество миллисекунд или одно из значений:
  - ◊ `fast` — 200 миллисекунд;
  - ◊ `normal` — 400 миллисекунд;
  - ◊ `slow` — 600 миллисекунд (значение по умолчанию).
- `animateEasing` — позволяет задать эффект замедления из дополнительных модулей или два следующих значения:
  - ◊ `swing` — в начале идет ускорение, а в конце замедление (значение по умолчанию);
  - ◊ `linear` — равномерная скорость движения.
- `autoHide` — если указано значение `true`, то маркер изменения размера (в правом нижнем углу) будет показываться только в случае, когда курсор мыши находится над элементом. Значение по умолчанию — `false`.
- `alsoResize` — позволяет указать еще один элемент, который будет изменять размеры одновременно с основным. В качестве значения можно указать DOM-элемент, селектор или объект jQuery; значение по умолчанию — `false`.
- `handles` — позволяет указать, захватывая какие элементы блока можно будет изменять его размеры. Указываются следующие значения (или их комбинация через запятую):
  - ◊ `n` — верхняя граница;
  - ◊ `e` — правая граница;
  - ◊ `s` — нижняя граница;
  - ◊ `w` — левая граница;
  - ◊ `nw` — верхний левый угол;
  - ◊ `ne` — верхний правый угол;
  - ◊ `se` — нижний правый угол;
  - ◊ `sw` — нижний левый угол;
  - ◊ `all` — все значения сразу.

Можно также передать объект со свойствами (`n, e, s, w, ne, se, sw, nw`). В качестве значения свойств указывается ссылка (селектор jQuery) на элемент. Значение по умолчанию — "`e, s, se`".

- `cancel` — позволяет указать элемент или несколько элементов внутри блока, захватив которые нельзя будет изменить размеры блока. В качестве значения указывается селектор jQuery. Значение по умолчанию — ":input, option".

Чтобы изменить или получить значения опций после инициализации, необходимо использовать второй формат метода `resizable()`. Для этого в первом параметре указывается значение `"option"`, во втором — название опции, а в третьем — новое значение.  
`$("#resizable").resizable("option", "containment", "document");`

Если необходимо получить значение опции, то третий параметр указывать не нужно. В качестве примера получим значение опции `maxHeight`.

```
alert($("#resizable").resizable("option", "maxHeight"));
```

Второй формат метода `resizable()` позволяет также управлять элементом. Для этого в первом параметре указываются следующие значения:

- `disable` — временно запрещает изменение размеров;
- `enable` — разрешает изменение размеров, если ранее оно было запрещено с помощью метода `disable`;
- `destroy` — удаляет всю функциональность и возвращает все элементы в первоначальное состояние.

Обработать события, происходящие с элементом, можно двумя способами.

```
resizable({
 <Событие>: <Функция обратного вызова>
});
```

или

```
bind(<Событие>, <Функция обратного вызова>)
```

В параметре `<Событие>` могут быть указаны следующие события (в скобках указано значение для метода `bind()`).

- `start (resizestart)` — наступает в начале изменения размеров.
- `resize (resize)` — возникает постоянно в процессе изменения размеров.
- `stop (resizestop)` — наступает в конце изменения размеров.

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>[, <Объект UI>]]) {
 // ...
}
```

Через параметр `<Объект UI>` доступны следующие свойства.

- `helper` — ссылка (объект jQuery) на вспомогательный элемент.
- `originalPosition` — позиция элемента до изменения размеров. Доступны два свойства:
  - ◊ `top` — положение сверху;
  - ◊ `left` — положение слева.
- `position` — позиция элемента после изменения размеров. Доступны два свойства:
  - ◊ `top` — положение сверху;
  - ◊ `left` — положение слева.
- `originalSize` — размеры элемента до их изменения. Доступны два свойства:
  - ◊ `width` — ширина элемента;
  - ◊ `height` — высота элемента.
- `size` — размеры элемента после их изменения. Доступны два свойства:

- ◊ `width` — ширина элемента;
- ◊ `height` — высота элемента.

Пример использования различных опций и обработки событий приведен в листинге 12.13.

### Листинг 12.13. Модуль UI Resizable, различные параметры настройки и события

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Resizable. РАЗличные настройки</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/ui-lightness/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<style>
body { font-size:10pt; font-family:Verdana, sans-serif; }
.resizable { width:300px; height:150px; padding:15px; }
.cls1 { border:1px dotted #000000; z-index:5000; }
</style>
<script type="text/javascript">
$(document).ready(function() {
 $("#resizable").resizable({
 maxHeight: 600, // Максимальная высота
 maxWidth: 1200, // Максимальная ширина
 minHeight: 100, // Минимальная высота
 minWidth: 100, // Минимальная ширина
 containment: "parent", // Ограничение границами родителя
 // Показывать маркер, только когда курсор расположен
 // над элементом
 autoHide: true,
 aspectRatio: true, // Пропорциональное изменение размеров
 stop: function(e, ui) {
 var msg=Размеры до: ширина - " +
 ui.originalSize.width;
 msg += ", высота - " + ui.originalSize.height + "
";
 msg += "Размеры после: ширина - " + ui.size.width;
 msg += ", высота - " + ui.size.height + "
";
 $("#div1").html(msg);
 }
 });
 $("#resizable2").resizable({
 animate: true, // Изменение размеров с анимацией
 animateDuration: 1000, // Продолжительность
 animateEasing: "swing", // Эффект замедления
 helper: "cls1" // Стилевой класс для помощника
 });
 $("#resizable3").resizable({ alsoResize: "#resizable4" });
 $("#resizable4").resizable({
```

```

 handles: "all", // Иэм. размера захватом за любую границу
 // Захватом за нижнюю границу изменить размер нельзя
 cancel: ".ui-resizable-s"
 });
});
//-->
</script>
</head>
<body>
<div style="width:900px; height:500px; border:1px solid #000000">
<div id="resizable" class="resizable ui-widget-content">
Пропорциональное изменение размеров в пределах родительского элемента
<div id="div1"></div></div>
</div>

<div id="resizable2" class="resizable ui-widget-content">
Использование помощника при изменении размеров

Изменение размеров производится с анимацией</div>

<div id="resizable3" class="resizable ui-widget-content">
Синхронное изменение размеров с нижним элементом</div>

<div id="resizable4" class="resizable ui-widget-content">
Изменить размер можно захватом за любую границу,
кроме нижней</div>

</body>
</html>

```

## 12.6. Модуль UI Accordion — компонент “Аккордеон”

Модуль UI Accordion позволяет создать компонент с несколькими вкладками. Изначально отображается содержимое только одной вкладки, а у остальных доступны только заголовки. После щелчка мышью на заголовке вкладки она открывается, а остальные сворачиваются.

Для загрузки модуля переходим на страницу <http://jqueryui.com/download>. Оставляем установленными флагки UI Core, Accordion, 1.7.2, выбираем тему из списка (например, Start), а затем щелкаем на кнопке Download. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы следующие файлы.

- jquery-ui-1.7.2.custom.css и каталог images (расположены в папке css\start).
- jquery-1.3.2.min.js (расположен в папке js).
- jquery-ui-1.7.2.custom.min.js (расположен в папке js).

Все указанные файлы необходимо подключить к скрипту в том порядке, в котором они перечислены. В качестве примера выведем компонент с параметрами по умолчанию (листинг 12.14).

### Листинг 12.14. Компонент Аккордеон с параметрами по умолчанию

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>

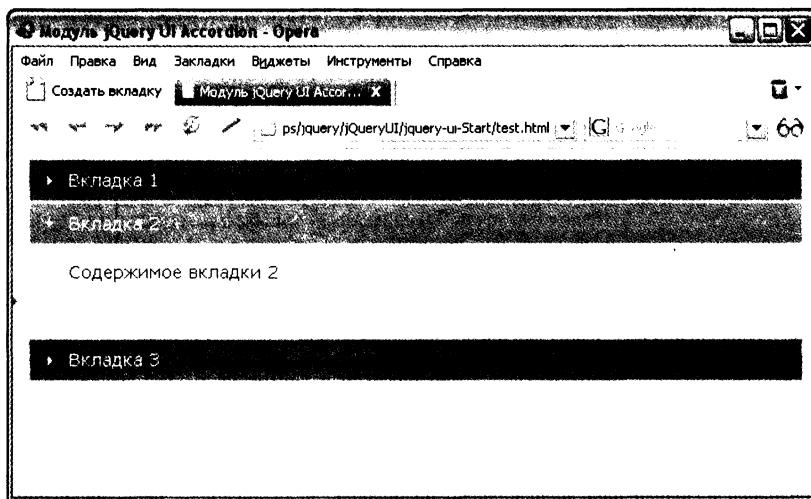
```

```

<head>
<title>Модуль jQuery UI Accordion</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/start/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#accordion").accordion({
 header: "h3"
 });
//-->
</script>
</head>
<body style="font-size:10pt; font-family:Verdana">
<div id="accordion">
<h3>Вкладка 1</h3>
<div>Содержимое вкладки 1</div>
<h3>Вкладка 2</h3>
<div>Содержимое вкладки 2</div>
<h3>Вкладка 3</h3>
<div>Содержимое вкладки 3</div>
</div>
</body>
</html>

```

Результат выполнения листинга 12.14 показан на рис. 12.2.



*Рис. 12.2. Компонент Аккордеон с параметрами по умолчанию*

Шаблоном для компонента является элемент DIV.

```
<div id="accordion">
<h3>Вкладка 1</h3>
<div>Содержимое вкладки 1</div>
...
<h3>Вкладка N</h3>
<div>Содержимое вкладки N</div>
</div>
```

Текст заголовка вкладки должен быть расположен внутри элемента A, а содержимое вкладки — внутри элемента DIV. Чтобы заголовок занимал всю доступную ширину окна, мы разместили заголовок внутри элемента H3. После загрузки по умолчанию отображается содержимое только первой вкладки, а высота равна высоте вкладки с максимальным содержимым. Щелкнув мышью на заголовке, можно отобразить соответствующую вкладку, а остальные будут скрыты.

Для создания компонента и управления им предназначен метод accordion(). Форматы метода:

```
accordion([<Объект с опциями>])
accordion(<Параметр>[, <Название опции>[, <Значение опции>]])
```

Параметр *<Объект с опциями>* представляет собой объект, состоящий из пар “опция/значение”. Могут быть указаны следующие опции.

- active — индекс вкладки, которая будет открыта изначально. Нумерация начинается с нуля. Если при создании компонента указать значение false, то изначально все вкладки будут закрыты. По умолчанию отображается первая вкладка.
- collapsible — если указано значение true, то открытая вкладка будет закрыта при щелчке на заголовке. При использовании этой опции можно закрыть все вкладки. Значение по умолчанию — false.
- header — позволяет задать элемент, который является заголовком вкладки.
- event — событие, по которому происходит переключение вкладок. По умолчанию используется событие click. Если указать событие mouseover, то вкладка будет открываться при наведении курсора.
- icons — позволяет задать значки, которые используются в заголовках. Указывается объект с двумя свойствами:
  - ◊ header — значок в заголовке закрытой вкладки;
  - ◊ headerSelected — значок в заголовке открытой вкладки.

В качестве значения свойств указывается стилевой класс значка в jQuery UI CSS Framework.

```
icons: {
 header: "ui-icon-arrow-1-e",
 headerSelected: "ui-icon-arrow-1-s"
}
```

- autoHeight — если указано значение false, то высота вкладки будет определяться внутренним содержимым. По умолчанию установлено значение true. Высота всех вкладок при этом значении определяется по вкладке с максимальным содержимым.

- `fillSpace` — если установлено значение `true`, то компонент будет занимать всю область, определяемую родительским элементом. Если содержимое вкладки не помещается, то будут выведены полосы прокрутки; значение по умолчанию — `false`.
- `animated` — позволяет указать эффект при открытии вкладки. Если указать значение `false`, то анимация будет отключена; значение по умолчанию — `slide`.
- `clearStyle` — если установлено значение `true`, то высота вкладки будет вычисляться при каждом открытии. Это позволяет работать с динамически изменяемым содержимым; значение по умолчанию — `false`.
- `navigation` — если установлено значение `true`, то все ссылки внутри компонента будут сравниваться с текущим URL-адресом документа. В случае нахождения совпадения, вкладка, которая содержит ссылку с URL-адресом, равным URL-адресу документа, будет открыта. Такое поведение компонента позволяет создать панель навигации. Однако нужно следить за тем, чтобы внутри разных вкладок не было ссылок с одинаковым URL-адресом. В противном случае поведение компонента становится непредсказуемым. Для реализации панели навигации в качестве содержимого вкладки используется разметка в виде списка.

```
<div id="accordion">
<h3>Раздел 1</h3>

Ссылка 1

<h3>Раздел 2</h3>

Ссылка 2

</div>
```

Значение по умолчанию — `false`.

- `navigationFilter` — позволяет задать пользовательскую функцию сравнения при значении опции `navigation`, равном `true`. На каждой итерации внутри функции обратного вызова доступен указатель (`this`) на текущую ссылку. Если внутри функции вернуть значение `true`, то вкладка, содержащая ссылку, будет открыта. По умолчанию все ссылки внутри компонента сравниваются с текущим URL-адресом документа.

```
navigationFilter: function() {
 return
 this.href.toLowerCase() == location.href.toLowerCase();
}
```

В качестве примера создадим панель навигации, которая занимает определенную область. Содержимое раздела будет отображаться при наведении курсора мыши на заголовок (листинг 12.15).

#### **Листинг 12.15. Панель навигации, созданная с помощью модуля UI Accordion**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
```

```

<head>
<title>Панель навигации с помощью модуля UI Accordion</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/start/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#accordion").accordion({
 event: "mouseover",
 header: "h3",
 icons: {
 header: "ui-icon-arrow-1-e",
 headerSelected: "ui-icon-arrow-1-s"
 },
 navigation: true,
 fillSpace: true
 });
 $("#btn1").click(function() {
 $("#div2").text($("#div1").html());
 });
});
//-->
</script>
</head>
<body style="font-size:10pt; font-family:Verdana">
<h2>Панель навигации</h2>
<table border="0"><tr><td valign="top">
<div style="width:150px; height:250px;" id="div1">
<div id="accordion">
<h3>Раздел 1</h3>

Ссылка 1

<h3>Раздел 2</h3>

Ссылка 2

<h3>Раздел 3</h3>

Ссылка 3

</div>
</div>
</td><td valign="top">
Перейдите по этим ссылкам, чтобы увидеть эффект

Ссылка на раздел 1

Ссылка на раздел 2

Ссылка на раздел 3

<input type="button" value="Показать исходный код" id="btn1">
<div id="div2"></div>

```

```
</td></tr></table>
</body>
</html>
```

Результат выполнения листинга 12.15 показан на рис. 12.3.

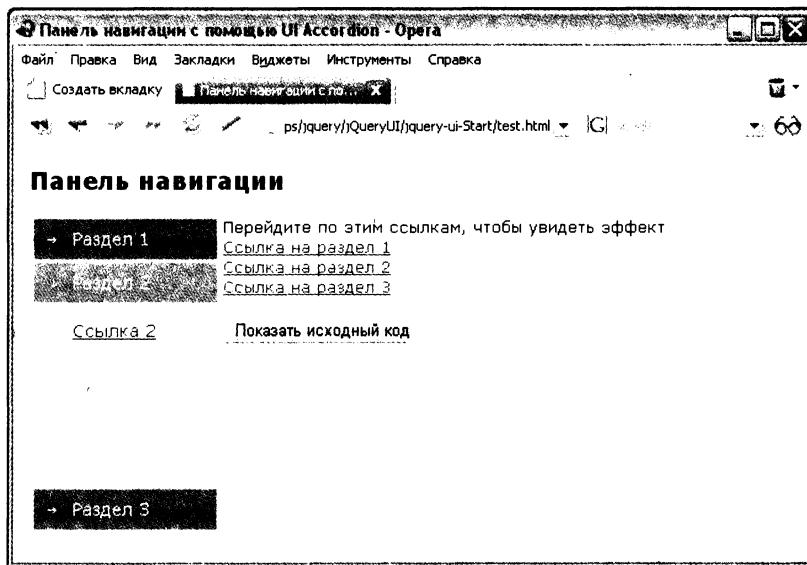


Рис. 12.3. Результат выполнения листинга 12.15

Первый формат метода `accordion()` позволяет задать значения при создании компонента. Чтобы изменить или получить значения опций уже после создания, необходимо использовать второй формат метода `accordion()`. Для этого в первом параметре указывается значение "option", во втором — название опции, а в третьем — новое значение.

```
$("#accordion").accordion("option", "icons", {
 header: "ui-icon-arrow-1-e",
 headerSelected: "ui-icon-arrow-1-s"
});
```

Если необходимо получить значение опции, то третий параметр указывать не нужно. Получим значение опции `animated`.

```
alert($("#accordion").accordion("option", "animated"));
```

Второй формат метода `accordion()` позволяет также управлять созданным компонентом. Для этого в первом параметре указываются следующие значения.

- `activate` — открывает указанную вкладку. Во втором параметре указывается индекс вкладки или селектор. Нумерация вкладок начинается с нуля. Если указать значение `-1`, то все вкладки будут закрыты. В качестве примера откроем вторую вкладку.
  - `destroy` — удаляет компонент и возвращает все элементы в первоначальное состояние.
- ```
$("#accordion").accordion("activate", 1);
```

- `disable` — временно запрещает использование компонента.
- `enable` — разрешает использование компонента, если ранее оно было запрещено с помощью значения `disable`.

Обработать события, происходящие с компонентом, можно двумя способами.

```
accordion({
    <Событие>: <Функция обратного вызова>
});
или
bind(<Событие>, <Функция обратного вызова>)
```

В параметре `<Событие>` могут быть указаны следующие события (в скобках указано значение для метода `bind()`).

- `changestart` (`accordionchangestart`) — наступает в начале смены вкладки.
- `change` (`accordionchange`) — наступает после того, как сменилась вкладка (после окончания анимации).

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата:

```
function <Название функции>([<Объект event>[, <Объект UI>]]) {
    // ...
}
```

Через параметр `<Объект UI>` доступны следующие объекты jQuery:

- `newHeader` — ссылка на заголовок открытой вкладки;
- `oldHeader` — ссылка на заголовок закрытой вкладки;
- `newContent` — ссылка на содержимое открытой вкладки;
- `oldContent` — ссылка на содержимое закрытой вкладки.

Пример обработки различных событий приведен в листинге 12.16.

Листинг 12.16. Модуль UI Accordion, обработка событий

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>События в модуле UI Accordion</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
      href="css/start/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
      type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#accordion").accordion({
        header: "h3",
        changestart: function() {
            $("#div1").append("Событие changestart<br>");
```

```

        },
        change: function(e, ui) {
            $("#div1").append("Событие change<br>");
            ui.newHeader.find("a").html("Открытая вкладка");
            ui.oldHeader.find("a").html("Закрытая вкладка");
        }
    });
    $("#accordion").bind("accordionchangestart", function() {
        $("#div1").append("Событие accordionchangestart<br>");
    }).bind("accordionchange", function() {
        $("#div1").append("Событие accordionchange<br>");
    });
});
//-->
</script>
</head>
<body style="font-size:10pt; font-family:Verdana">
<div id="accordion">
<h3><a href="#">Вкладка 1</a></h3>
<div>Содержимое вкладки 1</div>
<h3><a href="#">Вкладка 2</a></h3>
<div>Содержимое вкладки 2</div>
<h3><a href="#">Вкладка 3</a></h3>
<div>Содержимое вкладки 3</div>
</div>
<div id="div1"></div>
</body>
</html>

```

12.7. Модуль UI Tabs – панель с вкладками

Модуль UI Tabs позволяет создать панель с вкладками. Для загрузки модуля переходим на страницу <http://jqueryui.com/download>. Оставляем установленными флаги UI Core, Tabs, 1.7.2, выбираем тему из списка (например, Redmond), а затем щелкаем на кнопке Download. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы следующие файлы:

- jquery-ui-1.7.2.custom.css и каталог images (расположены в папке css\redmond);
- jquery-1.3.2.min.js (расположен в папке js);
- jquery-ui-1.7.2.custom.min.js (расположен в папке js).

Все указанные файлы необходимо подключить к скрипту в том порядке, в котором они перечислены. В качестве примера выведем панель с параметрами по умолчанию (листинг 12.17).

Листинг 12.17. Панель с вкладками, параметры по умолчанию

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
           "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Панель с вкладками</title>

```

```

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
      href="css/redmond/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
      type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $('#tabs').tabs();
});
//-->
</script>
</head>
<body style="font-size:10pt; font-family:Verdana">
<div id="tabs">
<ul>
<li><a href="#tabs1"><span>Вкладка 1</span></a></li>
<li><a href="#tabs2"><span>Вкладка 2</span></a></li>
<li><a href="#tabs3"><span>Вкладка 3</span></a></li>
</ul>
<div id="tabs1"><p>Содержимое вкладки 1</p></div>
<div id="tabs2"><p>Содержимое вкладки 2</p></div>
<div id="tabs3"><p>Содержимое вкладки 3</p></div>
</div>
</body>
</html>

```

Результат выполнения листинга показан на рис. 12.4.

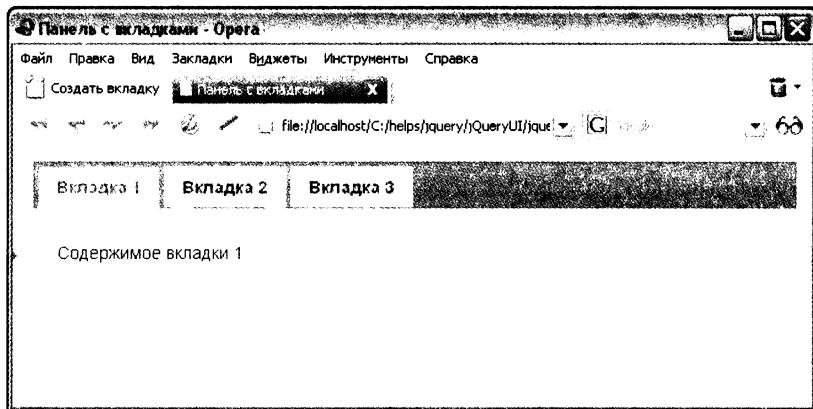


Рис. 12.4. Результат выполнения листинга 12.17

Шаблоном для панели с вкладками является элемент DIV.

```

<div id="tabs">
<ul>
<li><a href="#tabs1"><span>Вкладка 1</span></a></li>
<li><a href="#tabs2"><span>Вкладка 2</span></a></li>
<li><a href="#tabs3"><span>Вкладка 3</span></a></li>

```

```
</ul>
<div id="tabs1"><p>Содержимое вкладки 1</p></div>
<div id="tabs2"><p>Содержимое вкладки 2</p></div>
<div id="tabs3"><p>Содержимое вкладки 3</p></div>
</div>
```

Заголовки вкладок оформляются в виде списка. Внутри пункта списка расположен элемент A, в параметре href которого указывается идентификатор элемента DIV, в котором находится содержимое вкладки. После списка должны быть расположены элементы DIV с содержанием вкладок.

Модуль UI Tabs позволяет также заполнять содержимое вкладок с помощью AJAX-запроса. В этом случае в параметре href тегов <a> указывается URL-адрес документа. Шаблон такой панели будет выглядеть следующим образом.

```
<div id="tabs">
<ul>
<li><a href="doc1.html"><span>Вкладка 1</span></a></li>
<li><a href="doc2.html"><span>Вкладка 2</span></a></li>
</ul>
</div>
```

Для создания панели с вкладками и управления ею предназначен метод tabs(). Формат метода:

```
tabs([<Объект с опциями>])
```

Параметр <Объект с опциями> представляет собой объект, состоящий из пар “опция/значение”. Могут быть указаны следующие опции.

- selected — индекс вкладки, которая будет открыта изначально. Нумерация начинается с нуля. Если при создании компонента указать значение -1, то изначально все вкладки будут закрыты. По умолчанию отображается первая вкладка. В качестве примера отобразим содержимое третьей вкладки.

```
$('#tabs').tabs({ selected: 2 });
```

- event — событие, по которому происходит переключение вкладок. По умолчанию используется событие click. Если указать событие mouseover, то вкладка будет открываться при наведении курсора.
- collapsible — если указано значение true, то открытая вкладка будет закрыта при щелчке на заголовке. При использовании этой опции можно закрыть все вкладки. Значение по умолчанию — false.
- disabled — массив с индексами вкладок, которые должны быть недоступными изначально. Нумерация вкладок начинается с нуля. Если необходимо сделать неактивной первую вкладку, то с помощью опции selected следует указать индекс другой вкладки.
- fx — позволяет задать анимационный эффект при отображении и скрытии содержимого вкладок.

```
$('#tabs').tabs({ fx: { opacity: "toggle" } });
```

- cookie — позволяет сохранить индекс последней выбранной вкладки в cookies. Для использования опции необходимо подключение модуля Cookie (расположен в папке development -bundle\external\cookie). Если индекс вкладки со-

хранен, то она будет открыта при следующей загрузке страницы, при условии что индекс не указан явно в опции `selected`. В качестве значения указывается объект с парами “опция/значение”. Вот пример сохранения cookies на семь дней.

- ```
$('#tabs').tabs({ cookie: { expires: 7 } });
```
- `ajaxOptions` — объект с опциями AJAX-запроса. Значение по умолчанию — `null`.
  - `cache` — управление заполнением вкладок с помощью AJAX-запроса. При значении `true` повторный щелчок на заголовке вкладки не будет приводить к загрузке данных с сервера. Значение по умолчанию — `false`.
  - `spinner` — HTML-код, который отображается в заголовке вкладки во время загрузки данных с помощью AJAX-запроса. Если в качестве значения опции указать пустую строку, то текст заголовка меняться не будет. Значение по умолчанию: `<em>Loading...</em>`.
  - `tabTemplate` — HTML-шаблон для создания заголовка новых вкладок. Значение по умолчанию:  
`<li><a href="#{href}"><span>#{label}</span></a></li>`
  - `panelTemplate` — HTML-шаблон для создания содержимого новых вкладок. Значение по умолчанию: `<div></div>`.

Первый формат метода `tabs()` позволяет задать значения при создании панели. Чтобы изменить или получить значения опций уже после создания, необходимо использовать следующий формат метода `tabs()`.

```
tabs("option", <Название опции>[, <Значение опции>])
```

В первом параметре указывается значение `"option"`, во втором — название опции, а в третьем — новое значение.

```
$("#tabs").tabs("option", "spinner", "Загрузка...");
```

Если необходимо получить значение опции, то третий параметр указывать не нужно.

```
alert($("#tabs").tabs("option", "spinner"));
```

Для управления созданной панелью предназначены следующие методы.

- `add` — добавляет новую вкладку. Формат метода:  
`tabs("add", <URL-адрес или ID>, <Текст заголовка>[, <Индекс>])`  
Если последний параметр не указан, то вкладка будет добавлена после всех вкладок. Нумерация начинается с нуля. Добавим вкладку в самое начало.  
`$("#tabs").tabs("add", "ajax1.php", "Новая вкладка", 0);`
- `remove` — удаляет вкладку. Формат метода:  
`tabs("remove", <Индекс>)`  
Удалим первую вкладку.  
`$("#tabs").tabs("remove", 0);`
- `length` — возвращает количество вкладок.  
`alert($("#tabs").tabs("length"));`
- `select` — позволяет программно выбрать вкладку. Формат метода:  
`tabs("select", <Индекс или ID>)`

Создадим новую вкладку, а затем сделаем ее активной.

```
var i = $("#tabs").tabs("length");
$("#tabs").tabs("add", "ajax1.php", "Новая вкладка", i)
.tabs("select", i);
```

- url — позволяет изменить URL-адрес вкладки. Формат метода:

```
tabs("url", <Индекс вкладки>, <Новый URL-адрес>)
```

Рассмотрим пример.

```
$("#tabs").tabs("url", 1, "ajax2.php");
```

- load — перезагружает содержимое вкладки с помощью AJAX-запроса. Формат метода:

```
tabs("load", <Индекс вкладки>)
```

- rotate — позволяет установить автоматический перебор всех вкладок в цикле. Формат метода:

```
tabs("rotate", <Время>[, <Действие при выборе вкладки>])
```

Во втором параметре указывается количество миллисекунд, в течение которых вкладка будет открыта. Если указать значение 0 или null, то перебор вкладок будет остановлен. Если в третьем параметре указать значение true, то при выборе вкладки пользователем перебор вкладок будет продолжен. По умолчанию параметр имеет значение false.

```
$("#tabs").tabs("rotate", 5000);
```

- abort — прерывает выполнение всех AJAX-запросов и анимаций.
- disable — временно запрещает использование компонента.
- enable — разрешает использование компонента, если ранее оно было запрещено с помощью метода disable.
- disable — делает указанную вкладку недоступной. Обратите внимание на то, что текущая вкладка не может быть недоступной. Формат метода:

```
tabs("disable", <Индекс вкладки>)
```

Если необходимо сделать недоступными сразу несколько вкладок, то можно воспользоваться методом data(). Сделаем вторую и третью вкладки недоступными.

```
$("#tabs").data("disabled.tabs", [1, 2]);
```

- enable — делает недоступную вкладку доступной. Формат метода:

```
tabs("enable", <Индекс вкладки>)
```

Если необходимо сделать все вкладки доступными, то можно воспользоваться методом data().

```
$("#tabs").data("disabled.tabs", []);
```

- destroy — удаляет компонент и возвращает все элементы в первоначальное состояние.

Пример заполнения вкладок с помощью AJAX-запроса приведен в листинге 12.18.

## Листинг 12.18. Заполнение вкладок с помощью AJAX

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Заполнение вкладок с помощью AJAX</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/redmond/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script src="development-bundle/external/cookie/jquery.cookie.min.js"
 type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#tabs").tabs({
 cookie: { expires: 7 }, // Запоминаем номер открытой вкладки
 ajaxOptions: { cache: false }, // Получаем последние данные
 cache: true, // Делаем только один запрос на сервер
 spinner: "Загрузка...",
 disabled: [2] // Делаем третью вкладку недоступной
 });
 $("#btn1").click(function() {
 $("#tabs").tabs("disable", 1);
 });
 $("#btn2").click(function() {
 $("#tabs").data("disabled.tabs", []);
 });
 $("#btn3").click(function() {
 $("#tabs").tabs("add", "ajax3.php", "Новая вкладка");
 });
});
//-->
</script>
</head>
<body style="font-size:10pt; font-family:Verdana">
<div id="tabs">

Вкладка 1
Вкладка 2
Вкладка 3

<div id="tabs1"><p>Содержимое вкладки 1</p></div>
</div>
<input type="button" value="Сделать вторую вкладку недоступной" id="btn1">
<input type="button" value="Сделать все доступными" id="btn2">
<input type="button" value="Добавить вкладку" id="btn3">
</body>
</html>
```

Обработать события, происходящие с компонентом, можно двумя способами.

```
tabs({
 <Событие>: <Функция обратного вызова>
});

или
bind(<Событие>, <Функция обратного вызова>)
```

В параметре *<Событие>* могут быть указаны следующие события (в скобках указано значение для метода bind()):

- `select (tabsselect)` — при выборе неактивной вкладки. Если внутри функции обратного вызова вернуть значение `false`, то вкладка выбрана не будет.
- `load (tabsload)` — после загрузки данных с сервера.
- `show (tabsshow)` — при отображении содержимого вкладки.
- `disable (tabsdisable)` — когда вкладка становится недоступной.
- `enable (tabsenable)` — когда недоступная вкладка становится доступной.
- `add (tabsadd)` — при добавлении новой вкладки.
- `remove (tabsremove)` — при удалении вкладки.

В параметре *<Функция обратного вызова>* указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>[, <Объект UI>]]) {
 // ...
}
```

Через параметр *<Объект UI>* доступны следующие свойства:

- `index` — индекс вкладки;
- `tab` — ссылка на заголовок вкладки;
- `panel` — ссылка на содержимое вкладки.

Пример обработки различных событий приведен в листинге 12.19.

### Листинг 12.19. Модуль UI Tabs, обработка событий

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>События в модуле UI Tabs</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/redmond/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js"
type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript">
 $(document).ready(function() {
 $("#tabs").tabs({
```

```

event: "mouseover", // Выбор вкладки при наведении курсора
selected: 1, // Делаем вторую вкладку активной
cache: true, // Делаем только один запрос на сервер
spinner: "Загрузка...",
select: function(e, ui) {
 // Делаем недоступной первую вкладку
 if (ui.index == 0) return false;
},
load: function() { // Событие после загрузки данных
 $("#div1").append("Событие load
");
},
show: function(e, ui) { // Событие при отображении вкладки
 var msg = "Событие show
";
 msg += "Индекс вкладки: " + ui.index + "
";
 msg += "Текст заголовка: " + ui.tab.innerHTML + "
";
 msg += "Текст содержимого: " + ui.panel.innerHTML + "
";
 $("#div1").append(msg);
}
}),
//-->
</script>
</head>
<body style="font-size:10pt; font-family:Verdana">
<div id="tabs">

Вкладка 1
Вкладка 2
Вкладка 3

<div id="tabs1"><p>Содержимое вкладки 1</p></div>
<div id="tabs2"><p>Содержимое вкладки 2</p></div>
</div>
<div id="div1"></div>
</body>
</html>

```

## 12.8. Модуль UI Dialog – диалоговые окна

Модуль UI Dialog позволяет создавать пользовательские диалоговые окна. Для загрузки модуля переходим на страницу <http://jqueryui.com/download>. Оставляем установленными флаги **UI Core, Draggable, Resizable, Dialog, 1.7.2**, выбираем тему из списка (например, **Smoothness**), а затем щелкаем на кнопке **Download**. Модули **Draggable** и **Resizable** необходимы только в том случае, если требуется перемещать окно или изменять его размеры соответственно. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы следующие файлы:

- `jquery-ui-1.7.2.custom.css` и каталог `images` (расположены в папке `css\smoothness`);
- `jquery-1.3.2.min.js` (расположен в папке `js`);
- `jquery-ui-1.7.2.custom.min.js` (расположен в папке `js`);

- jquery.bgiframe.min.js (расположен в папке development-bundle1\external\bgiframe).

Все указанные файлы необходимо подключить к скрипту в том порядке, в котором они перечислены. В качестве примера выведем диалоговое окно с параметрами по умолчанию (листинг 12.20).

### **Листинг 12.20. Диалоговое окно с параметрами по умолчанию**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Модуль jQuery UI Dialog. Окно по умолчанию</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/smoothness/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript"
 src="development-bundle1/external/bgiframe/jquery.bgiframe.min.js">
</script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#dialog").dialog();
});
//-->
</script>
<style type="text/css">
.ui-dialog { font-size: 8pt; font-family: "Verdana", sans-serif; }
.ui-dialog-title { font-size: 10pt; font-weight: bold; }
.ui-dialog-content { font-size: 10pt; }
</style>
</head>
<body>
<div id="dialog" title="Заголовок окна">
<p>Это текст внутри окна</p>
</div>
</body>
</html>
```

Результат выполнения листинга 12.20 показан на рис. 12.5.

Как видно из приведенного примера, шаблоном для диалогового окна является элемент DIV.

```
<div id="dialog" title="Заголовок окна">
<p>Это текст внутри окна</p>
</div>
```

Текст, расположенный внутри элемента, выводится в окне, а значение параметра title становится заголовком. По умолчанию можно перемещать окно и изменять его размеры, а щелчком на кнопке “×” в правом верхнем углу окно можно закрыть.

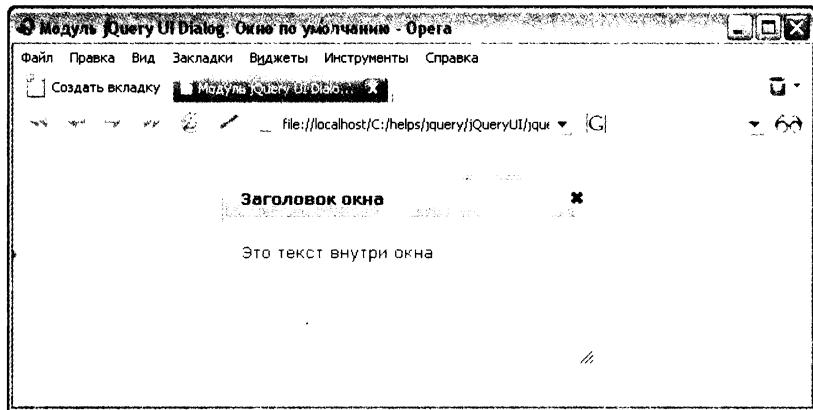


Рис. 12.5. Диалоговое окно с параметрами по умолчанию

Для создания диалогового окна и управления им предназначен метод `dialog()`. Форматы метода:

```
dialog([<Объект с опциями>])
dialog(<Параметр>[, <Название опции>[, <Значение опции>]])
```

Параметр `<Объект с опциями>` представляет собой объект, состоящий из пар “опция/значение”. Могут быть указаны следующие опции.

- `autoOpen` — если указано значение `true`, то диалоговое окно будет автоматически открываться сразу после вызова метода `dialog()`. Если указано значение `false`, то открыть окно можно только явно с помощью параметра `open`.

```
$("#dialog").dialog({
 autoOpen: false
});
$("#dialog").dialog("open");
```

Значением опции по умолчанию является `true`.

- `title` — задает текст в заголовке окна. Если опция не указана, то заголовком окна станет текст в параметре `title` тега `<DIV>`.
- `position` — определяет начальное положение окна. Может быть указан массив координат левого верхнего угла в пикселях.

```
position: [50, 50]
```

Вместо чисел могут быть указаны строковые значения `"center"`, `"left"`, `"right"`, `"top"` или `"bottom"`. Например, выведем окно в правом нижнем углу.

```
position: ["right", "bottom"]
```

В качестве значения может быть также указана строка.

```
position: "center"
```

- `modal` — если указано значение `true`, то с помощью создания дополнительного слоя между окном и другими элементами будет имитирован модальный режим. В этом режиме другие элементы будут доступны только после закрытия диалогового окна. Значением по умолчанию является `false`.

- `width` — ширина окна в пикселях; значение по умолчанию — 300.
- `maxWidth` — максимальная ширина окна в пикселях при изменении размера.
- `minWidth` — минимальная ширина окна в пикселях при изменении размера; значение по умолчанию — 150.
- `height` — высота окна в пикселях. По умолчанию высота окна определяется содержимым. Если высота недостаточна, то появятся полосы прокрутки.
- `maxHeight` — максимальная высота окна в пикселях при изменении размера.
- `minHeight` — минимальная высота окна в пикселях при изменении размера. Значение по умолчанию — 150.
- `draggable` — если установлено значение `true`, то окно можно будет перемещать с помощью мыши, а если `false`, то нельзя. Для работы опции необходимо подключение модуля UI Draggable. Значение по умолчанию — `true`.
- `resizable` — если установлено значение `true`, то можно будет изменять размеры окна, а если `false`, то нельзя. Для работы опции необходимо подключение модуля UI Resizable. Значение по умолчанию — `true`.
- `show` — эффект при открытии окна. Значение по умолчанию — `null`.
- `hide` — эффект при закрытии окна. Значение по умолчанию — `null`.
- `zIndex` — значение атрибута `z-index` для диалогового окна. Значение по умолчанию — 1000.
- `closeOnEscape` — если указано значение `true`, то окно можно закрыть с помощью клавиши `<Esc>`, а если `false`, то нельзя. Значение по умолчанию — `true`.
- `dialogClass` — дополнительные стилевые классы, которые будут применены к окну. Если классов несколько, то следует указать их через пробел.

```
$("#dialog").dialog({
 dialogClass: "dialog-font-size dialog-font-family"
});

<style type="text/css">
.dialog-font-size { font-size: 8pt; }
.dialog-font-family { font-family: "Verdana", sans-serif; }
</style>
```

- `bgiframe` — если указано значение `true`, то будет исправлена ошибка в браузере Internet Explorer 6. Ошибка заключается в том, что элементы `SELECT` помещаются поверх других элементов независимо от значения атрибута `z-index`. Обратите внимание: для работы этой опции необходимо подключить модуль BgIframe. Значение по умолчанию — `false`.
- `stack` — если указано значение `true`, то при выделении окно будет отображаться поверх всех других открытых диалоговых окон. Значение по умолчанию — `true`.
- `buttons` — позволяет разместить кнопки в диалоговом окне и назначить им обработчики. Выведем кнопку и после щелчка на ней закроем окно.

```
$("#dialog").dialog({
 buttons: {
```

```

 "Закрыть": function() {
 $(this).dialog("close"); // Закрываем окно
 }
 });
}

```

Обратите внимание на то, что внутри анонимной функции доступен указатель `(this)` на текущее диалоговое окно.

Первый формат метода `dialog()` позволяет задать значения при создании окна. Чтобы изменить или получить значения опций уже после создания, необходимо использовать второй формат метода `dialog()`. Для этого в первом параметре указывается значение `"option"`, во втором — название опции, а в третьем — новое значение.

```
$("#dialog").dialog("option", "title", "Новый заголовок");
```

Если необходимо получить значение опции, то третий параметр указывать не нужно. Получим заголовок окна.

```
alert($("#dialog").dialog("option", "title"));
```

Для примера выведем диалоговое окно после щелчка на кнопке. В окне разместим надпись, текстовое поле и две кнопки. С помощью первой кнопки можно будет заменить заголовок окна на значение, введенное в текстовое поле. Вторая кнопка позволяет закрыть окно. Кроме того, создадим кнопку, при щелчке на которой выводится исходный HTML-код созданного диалогового окна (листинг 12.21).

### **Листинг 12.21. Вывод диалогового окна после щелчка на кнопке**

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Модуль jQuery UI Dialog</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/smoothness/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js" .
 type="text/javascript"></script>
<script type="text/javascript"
 src="development-bundle/external/bgiframe/jquery.bgiframe.min.js">
</script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#dialog").dialog({
 autoOpen: false, // Открывать ли окно сразу
 bgiframe: true, // Решение проблемы с IE6
 closeOnEscape: true, // Закрывать ли при нажатии Esc
 title: "Новый заголовок", // Заголовок
 position: ["center", 50], // Местоположение окна
 width: 500, // Ширина окна
 height: "auto", // Высота окна
 draggable: false, // Перемещение
 resizable: false, // Изменение размера
 });
}

```

```

modal: false, // Модальное окно или нет
show: null, // Эффект при открытии окна
hide: null, // Эффект при закрытии окна
buttons: { // Описание кнопок
 "Закрыть": function() {
 $(this).dialog("close"); // Закрыть окно
 },
 "Изменить заголовок": function() {
 $(this).dialog("option", "title", $("#txt1").val());
 $("#txt1").val("");
 }
},
});
// $(".ui-dialog-titlebar").hide(); // Спрятать заголовок
// $(".ui-dialog-titlebar-close").hide(); // Спрятать крестик
$("#btn1").click(function() {
 $("#dialog").dialog("open"); // Открыть окно
});
$("#btn2").click(function() {
 $("#div1").text($(".ui-dialog").html());
}),
);
//-->
</script>
<style type="text/css">
.ui-dialog { font-size: 8pt; font-family: "Verdana", sans-serif; }
.ui-dialog-title { font-size: 10pt; font-weight: bold; }
.ui-dialog-content { font-size: 10pt; }
</style>
</head>
<body>
<div id="dialog" title="Заголовок окна">
<p><span class="ui-icon ui-icon-info"
style="float:left; margin-right:.3em;">
Это текст внутри окна.

Новый текст заголовка:

<input type="text" id="txt1" size="40"></p>
</div>
<input type="button" value="Открыть окно" id="btn1">
<input type="button" value="Вывести исходный код окна" id="btn2">
<div id="div1"></div>
</body>
</html>

```

Результат выполнения листинга 12.21 показан на рис. 12.6.

Обратите внимание на две закомментированные строки.

```

$(".ui-dialog-titlebar").hide(); // Спрятать заголовок
$(".ui-dialog-titlebar-close").hide(); // Спрятать крестик

```

Модуль UI Dialog не предоставляет стандартных методов для скрытия заголовка или крестика. Однако, изучив исходный HTML-код диалогового окна, можно найти способ скрыть эти два элемента. Это мы и сделали.

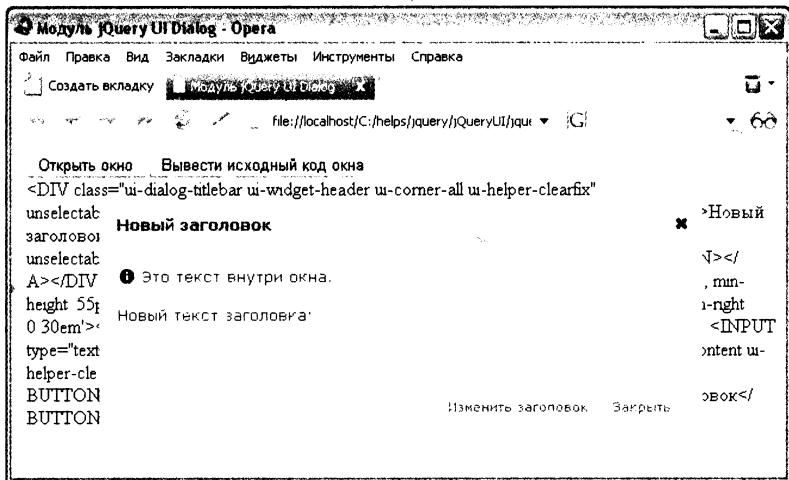


Рис. 12.6. Результат выполнения листинга 12.21

Если у нас несколько диалоговых окон и необходимо, чтобы только одно окно было без заголовка, то можно воспользоваться следующим кодом.

```
// Спрятать заголовок
$("#dialog").dialog({...}).parent()
.find(".ui-dialog-titlebar").hide();
// Спрятать крестик
$("#dialog").dialog({...}).parent()
.find(".ui-dialog-titlebar-close").hide();
```

Второй формат метода `dialog()` позволяет также управлять созданным окном. Для этого в первом параметре указываются следующие значения:

- `open` — открывает окно;
- `close` — закрывает окно;
- `destroy` — удаляет окно и возвращает элемент DIV в первоначальное состояние. Чтобы вновь открыть окно, его необходимо заново создать;
- `disable` — временно запрещает использование окна;
- `enable` — разрешает использование окна, если ранее оно было запрещено с помощью значения `disable`;
- `moveToTop` — помещает окно поверх всех других открытых диалоговых окон;
- `isOpen` — возвращает значение `true`, если диалоговое окно уже открыто.

Обработать события, происходящие с диалоговым окном, можно двумя способами.

```
dialog({
 <Событие>: <Функция обратного вызова>
});
или
bind(<Событие>, <Функция обратного вызова>)
```

В параметре `<Событие>` могут быть указаны следующие события (в скобках указано значение для метода `bind()`):

- `open (dialogopen)` — при открытии окна;
- `beforeclose (dialogbeforeclose)` — при попытке закрыть окно. Если внутри функции обратного вызова вернуть значение `false`, то окно закрыто не будет;
- `close (dialogclose)` — при закрытии окна;
- `focus (dialogfocus)` — в момент получения окном фокуса;
- `dragStart` — в начале перемещения окна;
- `drag` — наступает постоянно во время перемещения окна;
- `dragStop` — в конце перемещения окна;
- `resizeStart` — в начале изменения размеров окна;
- `resize` — наступает постоянно во время изменения размеров окна;
- `resizeStop` — в конце изменения размеров окна.

Для шести последних событий мне не удалось назначить обработчик элементу с идентификатором окна через метод `bind()`. Однако можно назначить обработчик элементам, имеющим класс `ui-dialog`. В этом случае название события указывается в нижнем регистре.

```
$("#dialog").dialog();
$(".ui-dialog").bind("dragstart", function() {
 $("#div1").append("Событие dragStart
");
}).bind("drag", function() {
 $("#div1").append("Событие drag
");
}).bind("dragstop", function() {
 $("#div1").append("Событие dragStop
");
});

<div id="dialog" title="Заголовок">Текст окна</div>
<div id="div1"></div>
```

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>[, <Объект UI>]]) {
 // ...
}
```

Значение параметра `<Объект UI>` зависит от события. Например, при перемещении окна доступны объекты `position`, `absolutePosition` и `offset`, возвращающие положение левого верхнего угла окна относительно других элементов или документа. Получить значения можно через свойства `top` и `left`. При изменении размера окна доступны объекты `position`, `originalPosition`, `size`, `originalSize`. Последние два объекта содержат информацию о размерах окна. Получить значения можно через свойства `width` и `height`. Пример обработки различных событий приведен в листинге 12.22.

### **Листинг 12.22. Модуль UI Dialog, обработка событий**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
```

```

<title>События в модуле UI Dialog</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/smoothness/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript"
 src="development-bundle/extrernal/bgiframe/jquery.bgiframe.min.js">
</script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#dialog").dialog({
 draggable: true,
 resizable: true,
 bgiframe: true,
 beforeclose: function() {
 if (!$("#ckb1").is(":checked")) {
 alert("Для закрытия необходимо установить флажок...");
 return false;
 }
 },
 dragStop: function(e, ui) {
 var msg = "position.top = " + ui.position.top + "
";
 msg += "position.left = " + ui.position.left + "
";
 msg += "offset.top = " + ui.offset.top + "
";
 msg += "offset.left = " + ui.offset.left + "
";
 $("#div1").append(msg + "
");
 },
 resizeStop: function(e, ui) {
 var msg = "position.top = " + ui.position.top + "
";
 msg += "position.left = " + ui.position.left + "
";
 msg += "size.width = " + ui.size.width + "
";
 msg += "size.height = " + ui.size.height + "
";
 $("#div1").append(msg + "
");
 },
 buttons: {
 "Закрыть окно": function() {
 $(this).dialog("close");
 }
 }
 });
//-->
</script>
</head>
<body style="font-size:8pt; font-family:Verdana;">
<div id="dialog" title="Заголовок окна">
<input type="checkbox" id="ckb1"> Я хочу закрыть окно
</div>
<div id="div1"></div>
</body>
</html>

```

## 12.9. Модуль UI Datepicker — календарь

Модуль UI Datepicker позволяет создать календарь. Для загрузки модуля переходим на страницу <http://jqueryui.com/download>. Оставляем установленными флаги UI Core, Datepicker, 1.7.2, выбираем тему из списка (например, Humanity), а затем щелкаем на кнопке Download. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы следующие файлы:

- jquery-ui-1.7.2.custom.css и каталог images (расположены в папке css\humanity);
- jquery-1.3.2.min.js (расположен в папке js);
- jquery-ui-1.7.2.custom.min.js (расположен в папке js);
- ui.datepicker-ru.js (расположен в папке development-bundle\ui\i18n).

Все указанные файлы необходимо подключить к скрипту в том порядке, в котором они перечислены. В качестве примера выведем календарь с параметрами по умолчанию (листинг 12.23).

### Листинг 12.23. Календарь с параметрами по умолчанию

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Datepicker. Календарь по умолчанию</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/humanity/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript"
 src="development-bundle/ui/i18n/ui.datepicker-ru.js"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#datepicker1").datepicker();
 $("#datepicker2").datepicker();
});
//-->
</script>
</head>
<body style="font-size:8pt; font-family:Verdana">
<table border="0"><tr><td>
<div id="datepicker1"></div>
</td><td valign="top">
<input type="text" id="datepicker2">
</td></tr></table>
</body>
</html>
```

Результат выполнения листинга 12.23 показан на рис. 12.7.

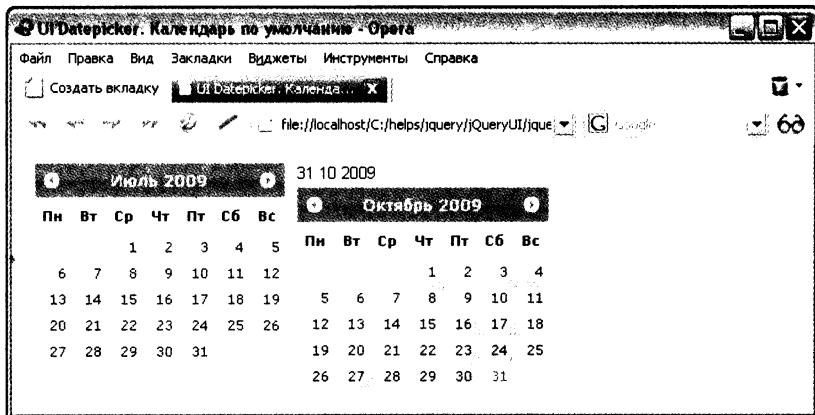


Рис. 12.7. Календарь с параметрами по умолчанию

Как видно из примера, шаблоном для календаря может быть элемент DIV:

```
<div id="datepicker1"></div>
```

или элемент INPUT.

```
<input type="text" id="datepicker2">
```

В первом случае календарь показывается постоянно, во втором случае — только тогда, когда текстовое поле получит фокус ввода. После выбора даты она отображается в поле ввода, а календарь скрывается.

Для создания календаря и управления им предназначен метод `datepicker()`. Формат метода:

```
datepicker([<Объект с опциями>])
```

Параметр `<Объект с опциями>` представляет собой объект, состоящий из пар “опция/значение”. Могут быть указаны следующие опции.

- `showAnim` — задает тип анимации при открытии календаря. Могут быть заданы значения "show", "slideDown", "fadeIn" или один из эффектов, определенных в jQuery UI Effects. Значение по умолчанию — "show".
- `showOptions` — если в опции `showAnim` указан один из эффектов, определенных в jQuery UI Effects, то с помощью этой опции можно передать дополнительные настройки. Значение по умолчанию — {}.
- `duration` — позволяет указать длительность эффекта анимации при открытии календаря. Может быть задано значение в миллисекундах или одно из строковых значений ("fast", "normal" или "slow"). Значение по умолчанию — "normal".
- `constrainInput` — если указано значение `false`, то в текстовое поле можно ввести строку любого формата. По умолчанию опция принуждает строго соблюдать формат даты, определенный в опции `dateFormat`. Значение по умолчанию — `true`.
- `dayNames` — массив с полными названиями дней недели (начиная с воскресенья). Значение по умолчанию:

```
['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
 'Friday', 'Saturday']
```

- dayNamesMin — массив с названиями дней недели из двух букв (начиная с воскресенья). Значение по умолчанию:  
['Su', 'Mo', 'Tu', 'We', 'Th', 'Fr', 'Sa']
- dayNamesShort — массив с названиями дней недели из трех букв (начиная с воскресенья). Значение по умолчанию:  
['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']
- monthNames — массив с полными названиями месяцев. Значение по умолчанию:  
['January', 'February', 'March', 'April', 'May', 'June', 'July',  
'August', 'September', 'October', 'November', 'December']
- monthNamesShort — массив с названиями месяцев из трех букв. Значение по умолчанию:  
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',  
'Oct', 'Nov', 'Dec']
- prevText — текст подсказки, появляющийся при наведении курсора мыши на значок, который служит ссылкой для перехода на предыдущий месяц. Если файл стилей не подключен, то этот текст становится текстом ссылки. Значение по умолчанию — "Prev".
- nextText — текст подсказки, появляющийся при наведении курсора мыши на значок, который служит ссылкой для перехода на следующий месяц. Если файл стилей не подключен, то этот текст становится текстом ссылки. Значение по умолчанию — "Next".
- navigationAsDateFormat — если указано значение `true`, то указание в опциях `nextText`, `prevText` и `currentText` строки специального формата будет приводить, например, к выводу названий месяцев в качестве текста подсказки.

```
$("#datepicker").datepicker({
 prevText: "MM", // Полное название месяца
 nextText: "MM", // Полное название месяца
 showButtonPanel: true,
 currentText: "dd.mm.yy", // Дата формата 27.07.2009
 navigationAsDateFormat: true
});
```

- isRTL — если используется язык с написанием справа налево, то в этой опции указывается значение `true`. Значение по умолчанию — `false`.
- firstDay — устанавливает день недели, который будет отображен первым в календаре. Для воскресенья указывается значение 0, для понедельника значение 1. Значение по умолчанию — 0.
- dateFormat — формат возвращаемой даты. Значение по умолчанию — "mm/dd/yy".
- shortYearCutoff — значение для определения, к какому столетию принадлежит год, заданный двумя цифрами. Если указано число от 0 до 99, то значение используется как есть. Если указана строка, то она преобразуется в число и прибавляется к текущему году. В этом случае в качестве значения будут использоваться две последние цифры года. Если год из двух цифр меньше значения опции

или равен ему, то год принадлежит к текущему столетию, а если больше — то к предыдущему. Значение по умолчанию — "+10". Например, текущий год 2009. По умолчанию к этому году прибавляется значение 10 и отбрасываются две первые цифры. Таким образом, значением опции будет число 19. При сравнении дата 01.01.19 будет преобразована в 01.01.2019, а дата 01.01.20 — в 01.01.1920.

- `showButtonPanel` — если указано значение `true`, то будет показана панель с двумя кнопками (рис. 12.8). Первая кнопка предназначена для перехода к сегодняшней дате, а вторая служит для закрытия календаря. Значение по умолчанию — `false`.

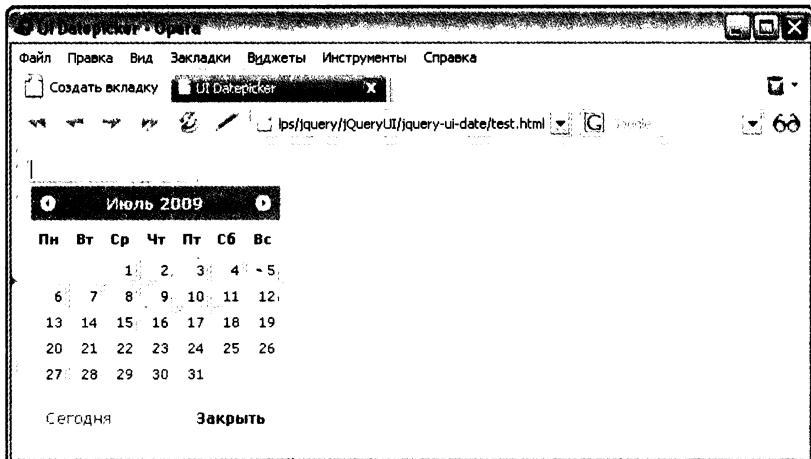


Рис. 12.8. Две кнопки, добавляемые с помощью опции `showButtonPanel`

- `currentText` — текст на кнопке, предназначеннной для перехода к сегодняшней дате. Кнопка будет доступна, если опция `showButtonPanel` имеет значение `true`. Значение по умолчанию — "Today".
- `gotoCurrent` — если указано значение `true`, то кнопка, по умолчанию предназначенная для перехода к сегодняшней дате, будет указывать на ранее выбранную дату, а не на текущую. Кнопка будет доступна, если опция `showButtonPanel` имеет значение `true`. Значение по умолчанию — `false`.
- `closeText` — текст на кнопке, предназначеннной для закрытия календаря. Кнопка будет доступна, если опция `showButtonPanel` имеет значение `true`. Значение по умолчанию — "Done".
- `changeMonth` — если указано значение `true`, то можно будет выбирать месяц из раскрывающегося списка (рис. 12.9). Значение по умолчанию — `false`.
- `changeYear` — если указано значение `true`, то можно будет выбирать год из раскрывающегося списка (см. рис. 12.9). Значение по умолчанию — `false`.
- `yearRange` — позволяет указать диапазон лет, отображаемых в раскрывающемся списке. Можно указать относительные значения (например, "-30:+1") или абсолютные (например, "1970:2009"). Раскрывающийся список будет доступен, если опция `changeYear` имеет значение `true`. Значение по умолчанию — "-10:+10".

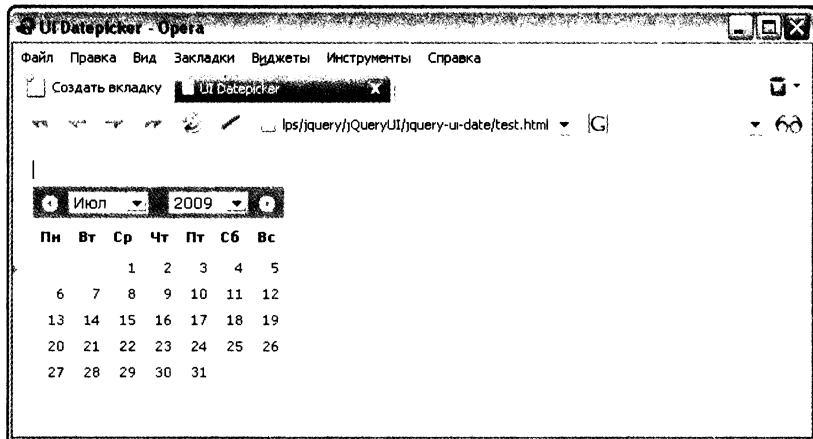


Рис. 12.9. Раскрывающиеся списки, предназначенные для выбора месяца и года

- `numberOfMonths` — количество одновременно показываемых месяцев. В качестве значения может быть указано число или массив из двух элементов. Например, число 3 означает, что необходимо вывести календарь на три месяца (результат изображен на рис. 12.10). При указании массива в первом элементе указывается количество строк, а во втором — количество показываемых месяцев в строке. Например, массив [2, 3] означает следующее: вывести календарь в две строки по три месяца на строке (результат показан на рис. 12.11). Значение по умолчанию — 1.

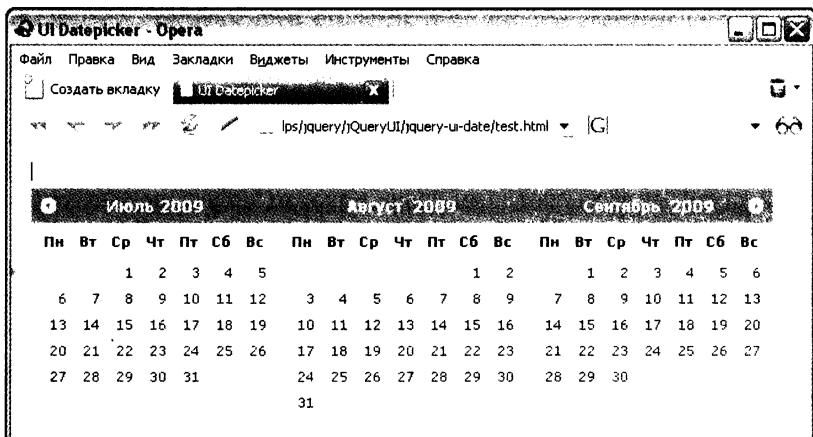


Рис. 12.10. Вид календаря при значении опции `numberOfMonths`, равном 3

- `stepMonths` — количество месяцев, на которое сдвигается календарь после щелчка на кнопках со стрелками влево и вправо, предназначенными для перемещения по месяцам. Значение по умолчанию — 1.
- `showCurrentAtPos` — смещение текущего месяца относительно левого верхнего угла при отображении календаря сразу на несколько месяцев. Значение по умолчанию — 0. Отобразим текущий месяц вторым из трех отображаемых.

```

$("#datepicker").datepicker({
 numberOfMonths: 3,
 showCurrentAtPos: 1
});

```

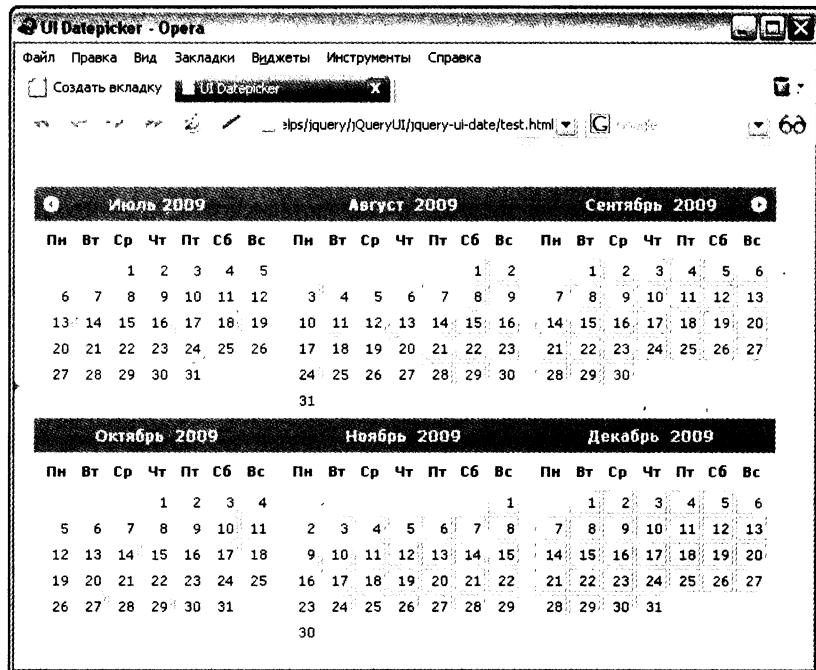


Рис. 12.11. Вид календаря при значении опции *numberOfMonths*, равном [2, 3]

- *showOtherMonths* — если указано значение *true*, то на календаре будут отображаться номера дней предыдущего и следующего месяцев без возможности их выбора. Значение по умолчанию — *false*.
  - *showMonthAfterYear* — если указано значение *true*, то в заголовке календаря название месяца будет следовать после года. Значение по умолчанию — *false*.
  - *defaultDate* — позволяет задать дату, которая будет подсвечена при первом открытии календаря. Значение по умолчанию — *null* (текущая дата).
  - *maxDate* — позволяет задать максимально возможную для выбора дату. Значение по умолчанию — *null* (ограничение отсутствует).
  - *minDate* — позволяет задать минимально возможную для выбора дату. Значение по умолчанию — *null* (ограничение отсутствует). В опциях *defaultDate*, *maxDate* и *minDate* дату можно указать следующими способами.
- ◊ Объект *Date*. Ограничим минимальную дату первым числом марта 2009 года.  
*minDate: new Date(2009, 2, 1)*

- ◊ Количество дней относительно текущей даты.

```
maxDate: 10
minDate: -10
```

- ◊ Стока специального формата, определяющая период. Могут быть указаны следующие периоды:

- ◊ y — год (например, `minDate: "-1y"`);
- ◊ m — месяц (например, `maxDate: "+1y +5m"`);
- ◊ w — неделя. (например, `minDate: "-12w"`);
- ◊ d — день (например, `maxDate: "+10d"`).

- `hideIfNoPrevNext` — если указано значение `true`, то при достижении края диапазона дат, значок, предназначенный для перемещения по месяцам, будет скрыт. Значение по умолчанию — `false` (значок становится неактивным).
- `showOn` — действие, которое будет приводить к отображению календаря. Могут быть указаны следующие значения:
  - ◊ `focus` — при получении фокуса полем ввода (значение по умолчанию);
  - ◊ `button` — при щелчке на кнопке;
  - ◊ `both` — при получении фокуса полем ввода, а также при щелчке на кнопке.
 Если указаны значения `button` или `both`, то после текстового поля автоматически будет добавлена кнопка.
- `buttonText` — позволяет задать текст на кнопке, с помощью которой можно отобразить календарь. Кнопка будет доступна, если в опции `showOn` указаны значения `button` или `both`. Значение по умолчанию — `"..."`.
- `buttonImage` — URL-адрес изображения, которое будет отображаться на кнопке (или вместо нее), с помощью которой можно отобразить календарь. Кнопка будет доступна, если в опции `showOn` указаны значения `button` или `both`. Если в опции `buttonText` был определен текст, то он становится значением параметра `alt`. Значение по умолчанию — `" "`.
- `buttonImageOnly` — если указано значение `true`, то для отображения календаря будет использоваться изображение вместо кнопки. Значение по умолчанию — `false`.
- `appendText` — позволяет задать текст, который будет отображаться после текстового поля, служащего для выбора даты. Значение по умолчанию — `" "`.
- `altField` — позволяет задать дополнительное поле, которое должно быть обновлено, после выбора даты. В качестве значения указывается селектор jQuery. Значение по умолчанию — `" "`.
- `altFormat` — формат для вывода даты в дополнительном текстовом поле, ссылка на который указана в опции `altField`.

Определить значения опций по умолчанию для всех календарей позволяет метод `$.datepicker.setDefaults()`. Формат метода:

```
$.datepicker.setDefaults(<Объект с опциями>);
```

В качестве примера переопределим некоторые опции локализации.

```
$datepicker.setDefaults(
 $.datepicker.regional["ru"], {
 prevText: "MM",
 nextText: "MM",
 currentText: "dd.mm.yy",
 dateFormat: "dd.mm.y"
 })
) ;
```

Чтобы изменить или получить значения опций уже после создания, необходимо использовать второй формат метода `datepicker()`. Для этого в первом параметре указывается значение `"option"`, во втором — название опции, а в третьем — новое значение.

```
$("#datepicker").datepicker("option", "dateFormat", "dd M yy");
```

Если необходимо получить текущее значение опции, третий параметр указывать не нужно. Пример — отображение текущего формата даты на экране:

```
alert($("#datepicker").datepicker("option", "dateFormat"));
```

Для примера выведем несколько календарей с различными настройками (листинг 12.24). Первый календарь постоянно отображен. Второй показывает сразу три месяца за раз и имеет ограничение на диапазон дат. Отображение календаря происходит при получении фокуса ввода текстовым полем. Кроме того, с помощью кнопки можно изменить формат даты. Третий календарь имеет раскрывающиеся списки с месяцами и годами и открывается с помощью щелчка на изображении. При выборе даты она отображается в поле ввода в одном формате, а затем дублируется в дополнительное текстовое поле в другом формате.

#### Листинг 12.24. Варианты календарей

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Datepicker. Варианты календарей</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/humanity/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript"
 src="development-bundle/ui/i18n/ui.datepicker-ru.js"></script>
<script type="text/javascript">

```

```

);
$("#datepicker1").datepicker(); // Статичный календарь
// Календарь, показывающий сразу три месяца и имеющий ограничения
$("#datepicker2").datepicker({
 maxDate: "+1y", // Максимальная дата
 minDate: "-12w", // Минимальная дата
 showButtonPanel: true, // Показать панель с кнопками
 numberOfMonths: 3, // Показывать сразу 3 месяца
 stepMonths: 3, // Сдвигать календарь на 3 месяца
 hideIfNoPrevNext: true,
 navigationAsDateFormat: true
});
// Календарь, открывающийся с помощью щелчка на изображении и
// заполняющий дополнительное текстовое поле
$("#datepicker3").datepicker({
 changeMonth: true, // Выпадающий список с месяцами
 changeYear: true, // Выпадающий список с годами
 yearRange: "1970:2009", // Диапазон в списке с годами
 showOn: "button", // Открывать при щелчке на кнопке
 buttonImage: "calendar.gif", // Адрес изображения
 buttonImageOnly: true, // Использовать изображение вместо кнопки
 appendText: " Обязательное поле",
 altField: "#txt1", // Ссылка на дополнительное поле
 altFormat: "dd M yy" // Формат даты в дополнительном поле
});
$("#btn1").click(function() { // Изменение формата даты
 $("#datepicker2").datepicker("option", "dateFormat", "dd M yy");
});
});
//-->
</script>
</head>
<body style="font-size:8pt; font-family:Verdana">
<div id="datepicker1"></div>

<input type="text" id="datepicker2">
<input type="button" value="Изменить формат" id="btn1">

<input type="text" id="datepicker3"> <input type="text" id="txt1">
</body>
</html>

```

В опциях dateFormat, altFormat и некоторых других опциях, задающих формат отображения даты, может быть использована комбинация следующих служебных символов:

- d — номер дня месяца без предваряющего нуля (например, 1.07.2009);
- dd — номер дня месяца с предваряющим нулем (например, 01.07.2009);
- o — день с начала года без предваряющих нулей (например, 5);
- oo — день с начала года с предваряющими нулями (например, 005);
- D — короткое название дня недели (например, "срд" для среды), названия задаются с помощью опции dayNamesShort;
- DD — полное название дня недели (например, "среда"), названия задаются с помощью опции dayNames;

- `m` — номер месяца без предваряющего нуля (например, `1.7.2009`);
- `mm` — номер месяца с предваряющим нулем (например, `1.07.2009`);
- `M` — короткое название месяца (например, "Июл" для июля), названия задаются с помощью опции `monthNamesShort`;
- `MM` — полное название месяца (например, "Июль"), названия задаются с помощью опции `monthNames`;
- `y` — год из двух цифр (например, `01.07.09`);
- `yy` — год из четырех цифр (например, `01.07.2009`);
- `@` — количество миллисекунд, прошедших с 1 января 1970 года (например, `1248811200000`);
- `'...'` — если в шаблоне необходимо вывести текст, содержащий специальные символы, то его необходимо заключить в одинарные кавычки (например, шаблон `'date: dd.mm.yy'` будет соответствовать выводу `date: 29.07.2009`);
- `' '` — если в шаблоне необходимо вывести одинарную кавычку, то ее следует удвоить.

Текст, не содержащий специальных символов, будет выводиться как есть (например, шаблон "Сегодня: dd.mm.yy" будет соответствовать выводу `Сегодня: 29.07.2009`).

Модуль UI Datepicker предоставляет также несколько предопределенных форматов даты.

- `$.datepicker.ATOM` — соответствует шаблону `"yy-mm-dd"`
- `$.datepicker.COOKIE` — соответствует шаблону `"D, dd M yy"`
- `$.datepicker.ISO_8601` — соответствует шаблону `"yy-mm-dd"`
- `$.datepicker.RFC_822` — соответствует шаблону `"D, d M y"`
- `$.datepicker.RFC_850` — соответствует шаблону `"DD, dd-M-y"`
- `$.datepicker.RFC_1036` — соответствует шаблону `"D, d M y"`
- `$.datepicker.RFC_1123` — соответствует шаблону `"D, d M yy"`
- `$.datepicker.RFC_2822` — соответствует шаблону `"D, d M yy"`
- `$.datepicker.RSS` — соответствует шаблону `"D, d M y"`
- `$.datepicker.TIMESTAMP` — соответствует шаблону `"@"`
- `$.datepicker.W3C` — соответствует шаблону `"yy-mm-dd"`

Для управления календарем предназначены следующие методы.

- `dialog` — открывает календарь в режиме диалогового окна. Формат метода:  
`datepicker("dialog", <Дата в виде строки>[,  
          <Функция обратного вызова>[, <Опции>[, <Позиция>]]])`

Во втором параметре указывается дата в формате, указанном в опции `dateFormat`. Календарь будет открыт на указанной дате. В параметре `<Функция обратного вызова>` можно указать ссылку на функцию, которая будет вызвана после выбора даты в календаре. Параметр `<Опции>` позволяет задать новые опции календаря, а в параметре `<Позиция>` можно указать координаты левого верхнего угла (например, `[150, 150]`).

- `setDate` — позволяет выбрать дату в календаре. Формат метода:  
`datepicker("setDate", <Дата>)`  
 Во втором параметре дату можно указать следующими способами:
  - ◊ с помощью объекта Date;
  - ◊ посредством количества дней относительно текущей даты (например, 7 или -12);
  - ◊ с помощью строки специального формата, определяющей период. Могут быть указаны следующие периоды:
    - ◊ у — год (например, "-1y");
    - ◊ m — месяц (например, "+1y +5m");
    - ◊ w — неделя (например, "-12w");
    - ◊ d — день (например, "+10d");
  - ◊ значение `null` — текущая дата.
- `getDate` — возвращает дату (объект Date), выбранную в календаре. Если дата не выбрана, то возвращается значение `null`.
- `show` — показывает календарь.
- `hide` — скрывает календарь.
- `disable` — временно запрещает использование календаря.
- `enable` — разрешает использование календаря, если ранее оно было запрещено с помощью метода  `disable`.
- `isDisabled` — возвращает `true`, если использование календаря было запрещено с помощью метода  `disable`, и `false` — в противном случае.
- `destroy` — удаляет календарь и возвращает все элементы в первоначальное состояние.

Пример использования различных методов приведен в листинге 12.25.

### Листинг 12.25. Пример использования различных методов

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Datepicker. Методы</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/humanity/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript"
 src="development-bundle/ui/i18n/ui.datepicker-ru.js"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#datepicker").datepicker();
-->
```

```

 $("#btn1").click(function() {
 $("#datepicker").datepicker("disable");
 });
 $("#btn2").click(function() {
 $("#datepicker").datepicker("enable");
 });
 $("#btn3").click(function() {
 if ($("#datepicker").datepicker("isDisabled")) {
 alert("Запрещено");
 }
 else {
 alert("Разрешено");
 }
 });
 $("#btn4").click(function() {
 $("#datepicker").datepicker("show");
 });
 $("#btn5").click(function() {
 $("#datepicker").datepicker("hide");
 });
 $("#btn6").click(function() {
 $("#datepicker").datepicker("setDate", "+1y");
 });
 $("#btn7").click(function() {
 alert($("#datepicker").datepicker("getDate"));
 });
 $("#btn8").click(function() {
 $("#datepicker").datepicker("dialog", "27.06.2009",
 function(dateText) {
 alert("Выбрана дата: " + dateText);
 }, { changeMonth: true, changeYear: true }, [200, 200]);
 });
});
//-->
</script>
</head>
<body style="font-size:8pt; font-family:Verdana">
<input type="button" value="Запретить" id="btn1">
<input type="button" value="Разрешить" id="btn2">
<input type="button" value="Проверить статус" id="btn3">
<input type="button" value="Открыть" id="btn4">
<input type="button" value="Закрыть" id="btn5">
<input type="button" value="Выбрать дату" id="btn6">
<input type="button" value="Получить дату" id="btn7">
<input type="button" value="Открыть диалог" id="btn8">

<input type="text" id="datepicker">
</body>
</html>

```

Обработать события, происходящие с календарем, можно следующим образом.

```

datepicker({
 <Событие>: <Функция обратного вызова>
});

```

В параметре *<Событие>* могут быть указаны следующие события.

- *beforeShow* — наступает перед открытием календаря. Указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Ссылка на поле ввода>]) {
 // ...
}
```

- *beforeShowDay* — наступает для каждой даты перед отображением календаря. Указывается ссылка на функцию следующего формата.

```
function <Название функции>(<Объект Date>) {
 // ...
}
```

Внутри функции необходимо вернуть массив со следующими элементами:

- ◊ 0 — значение *true*, если можно выбрать дату, или *false* — в противном случае;
- ◊ 1 — имя класса (или классов) для даты;
- ◊ 2 — текст всплывающей подсказки.

- *onChangeMonthYear* — при смене месяца или года в календаре. Указывается ссылка на функцию следующего формата.

```
function <Название функции>(<Год>, <Месяц>, <Объект>) {
 // ...
}
```

- *onSelect* — при выборе даты в календаре.
- *onClose* — после закрытия календаря (вне зависимости от того, выбрана дата или нет).

В событиях *onSelect* и *onClose* указывается ссылка на функцию следующего формата.

```
function <Название функции>(<Дата>, <Объект>) {
 // ...
}
```

Внутри функции обратного вызова доступен указатель (*this*) на поле ввода. Через параметр *<Дата>* можно получить дату в виде строки в формате, заданном опцией *dateFormat*. Параметр *<Объект>* представляет собой объект со следующими свойствами:

- *id* — идентификатор поля ввода, являющегося основой для календаря;
- *input* — ссылка на поле ввода;
- *selectedDay*, *selectedMonth*, *selectedYear* — день, месяц (нумерация начинается с нуля) и год, которые были выделены;
- *currentDay*, *currentMonth*, *currentYear* — день, месяц (нумерация начинается с нуля) и год, которые были выбраны пользователем; если дата не была выбрана, то эти свойства имеют значение 0.

Пример обработки различных событий приведен в листинге 12.26.

## Листинг 12.26. UI Datepicker, обработка событий

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Datepicker. Обработка событий</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/humanity/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript"
 src="development-bundle/ui/i18n/ui.datepicker-ru.js"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#datepicker").datepicker({
 changeMonth: true, // Выпадающий список с месяцами
 changeYear: true, // Выпадающий список с годами
 yearRange: "1970:2009", // Диапазон в списке с годами
 maxDate: "+1y", // Максимальная дата
 minDate: "-12w", // Минимальная дата
 beforeShow: function(elem) {
 $("#div1").html("Событие beforeShow
");
 $(elem).css("background-color", "#fff4dd");
 },
 beforeShowDay: function(d) {
 var curd = $.datepicker.formatDate("dd.mm.yy", new Date());
 var dat = $.datepicker.formatDate("dd.mm.yy", d);
 // Если дата является текущей, то указываем для ячейки
 // с датой класс cls1, а также задаем текст подсказки
 if (dat == curd) return [true, "cls1", "Это сегодня"];
 // Если число месяца равно 10, то делаем дату неактивной
 else if (d.getDate() == 10) return [false, "cls2"];
 else return [true, "cls2"];
 },
 onChangeMonthYear: function(y, m, obj) {
 var msg = "Событие onChangeMonthYear
";
 msg += "Год: " + y + "
Месяц: " + m + "
";
 $("#div1").append(msg);
 },
 onSelect: function(d, obj) {
 var msg = "Событие onSelect
Выбранная дата: " + d;
 $("#div1").append(msg + "
");
 },
 onClose: function(d, obj) {
 var msg = "Событие onClose
";
 if (obj.currentDay == 0) msg += "Дата не выбрана
";
 else {
 msg += "Выбранная дата: " + obj.currentDay + ".";
 msg += (obj.currentMonth + 1) + "." + obj.currentYear;
 msg += "
";
 }
 }
 });
});
```

```

 }
 $("#div1").append(msg);
 $(this).css("background-color", "#ffffff");
 }
}).css("background-color", "#ffffff");
});
//-->
</script>
<style type="text/css">
.cls1 { background-color: red; }
.cls2 { }
</style>
</head>
<body style="font-size:8pt; font-family:Verdana">
<input type="text" id="datepicker">
<div id="div1"></div>
</body>
</html>

```

Модуль UI Datepicker предоставляет также три функции, предназначенные для преобразования даты из одного формата в другой.

- `$.datepicker.formatDate()` — позволяет вывести дату в указанном формате. Синтаксис функции:

`$.datepicker.formatDate(<Формат даты>, <Объект Date>[, <Опции>])`

Пример:

```
alert($.datepicker.formatDate("dd.mm.yy", new Date(2009, 6, 29)));
// Выведет: 29.07.2009
```

- `$.datepicker.parseDate()` — производит разбор даты, представленной в виде строки, согласно указанному формату. Возвращает объект Date. Формат функции:

`$.datepicker.parseDate(<Формат даты>, <Дата в виде строки>[, <Опции>])`

Пример:

```
var d = $.datepicker.parseDate("dd.mm.yy", "29.07.2009");
alert(d.toString());
// Выведет: Wed, 29 Jul 2009 00:00:00 GMT+0400
```

В качестве еще одного примера передадим строку с неправильным форматом даты и выведем сообщение об ошибке.

```
try {
 var d = $.datepicker.parseDate("dd.mm.yy", "29.07");
}
catch(err) {
 alert(err); // Выведет: Unexpected literal at position 5
}
```

- `$.datepicker.ISO8601Week()` — возвращает номер недели в году по указанной дате. Формат функции:

`$.datepicker.iso8601Week(<Объект Date>)`

Пример:

```
alert($.datepicker.iso8601Week(new Date(2009, 6, 29)));
// Выведет: 31
```

В параметре <Опции> могут быть указаны следующие значения:

- dayNames — массив с полными названиями дней недели (начиная с воскресенья);
- dayNamesShort — массив с названиями дней недели из трех букв (начиная с воскресенья);
- monthNames — массив с полными названиями месяцев;
- monthNamesShort — массив с названиями месяцев из трех букв.

Кроме того, в функции `$.datepicker.parseDate()` в параметре <Опции> может быть указана опция `shortYearCutoff`, значение которой служит для определения, к какому столетию принадлежит год, заданный двумя цифрами. Указывается число от 0 до 99.

```
var d = $.datepicker.parseDate("dd-MM-y", "29-Июль-09", {
 shortYearCutoff: 9,
 monthNames: ['Январь', 'Февраль', 'Март', 'Апрель', 'Май', 'Июнь',
 'Июль', 'Август', 'Сентябрь', 'Октябрь', 'Ноябрь', 'Декабрь']
});
alert(d.toString());
// Выведет: Wed, 29 Jul 2009 00:00:00 GMT+0400
```

## 12.10. Модуль UI Progressbar — индикатор хода процесса

Модуль UI Progressbar позволяет создать индикатор процесса. Индикатор представляет прямоугольную область, внутри которой расположена полоса другого цвета. Ширина полосы зависит от какого-либо значения. Если значение равно нулю, то полоса не показывается, при значении 50 полоса будет начинаться от левого края области и продлится до середины, а если значение равно 100, то полоса будет занимать всю прямоугольную область.

Для загрузки модуля переходим на страницу <http://jqueryui.com/download>. Оставляем установленными флагки `UI Core`, `Progressbar`, `1.7.2`, выбираем тему из списка (например, `Sunny`), а затем щелкаем на кнопке `Download`. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы следующие файлы:

- `jquery-ui-1.7.2.custom.css` и каталог `images` (расположены в папке `css\sunny`);
- `jquery-1.3.2.min.js` (расположен в папке `js`);
- `jquery-ui-1.7.2.custom.min.js` (расположен в папке `js`).

Все указанные файлы необходимо подключить к скрипту в том порядке, в котором они перечислены. В качестве примера выведем индикатор процесса со значением 60% (листинг 12.27).

### Листинг 12.27. Модуль UI Progressbar, вывод индикатора хода процесса

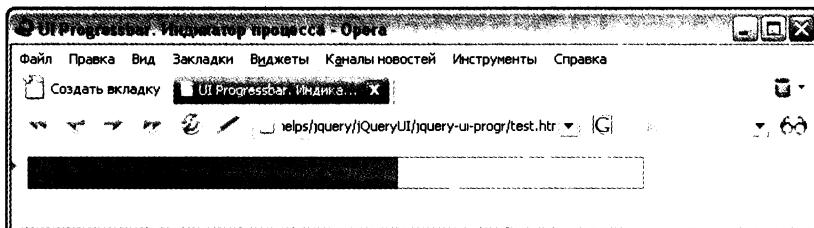
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
```

```

<head>
<title>UI Progressbar. Индикатор процесса</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/sunny/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#progressbar").progressbar({
 value: 60
 });
//-->
</script>
</head>
<body style="font-size:8pt; font-family:Verdana">
<div style="width: 500px;"><div id="progressbar"></div></div>
</body>
</html>

```

Результат выполнения листинга показан на рис. 12.12.



*Рис. 12.12. Индикатор хода выполнения процесса со значением 60%*

Как видно из примера, шаблоном для индикатора является элемент DIV.

```
<div id="progressbar"></div>
```

Для создания индикатора процесса и управления им предназначен метод `progressbar()`. Формат метода:

```
progressbar([<объект с опциями>])
```

Параметр `<объект с опциями>` представляет собой объект, состоящий из пар “опция/значение”. Модуль UI Progressbar предоставляет только одно свойство `value`, которое задает процент выполнения процесса. По умолчанию свойство имеет значение 0.

```
$("#progressbar").progressbar({ value: 20 });
```

Чтобы изменить или получить значения опций уже после создания объекта, необходимо использовать второй формат метода `progressbar()`. Для этого в первом параметре указывается значение `"option"`, во втором — название опции, а в третьем — новое значение.

```
$("#progressbar").progressbar("option", "value", 10);
```

Если необходимо получить значение опции, то третий параметр указывать не нужно. Получим текущее значение опции value.

```
alert($("#progressbar").progressbar("option", "value"));
```

Установить новое значение можно также с помощью метода value.

```
$("#progressbar").progressbar("value", 20);
```

Обработать события, происходящие с индикатором процесса, можно двумя способами.

```
progressbar({
 <Событие>: <Функция обратного вызова>
});
```

или

```
bind(<Событие>, <Функция обратного вызова>)
```

В параметре <Событие> может быть указано событие change (progressbarchange для метода bind()), которое возникает при изменении значения опции value. В параметре <Функция обратного вызова> указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>[, <Объект UI>]]) {
 // ...
}
```

Для управления созданным индикатором предназначены три метода:

- disable — временно запрещает использование индикатора;
- enable — разрешает использование индикатора, если ранее оно было запрещено с помощью метода disable;
- destroy — удаляет компонент и возвращает все элементы в первоначальное состояние.

Пример управления индикатором и обработки событий приведен в листинге 12.28.

### Листинг 12.28. Модуль UI Progressbar, события и методы

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Progressbar. События и методы</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/sunny/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<!--
$(document).ready(function() {
 $("#progressbar").progressbar({
 value: 60,
 change: function() {
 $("#div1").append("Событие change
");
```

```

}).bind("progressbarchange", function() {
 $("#div1").append("Событие progressbarchange
");
});
$("#btn1").click(function() {
 var val = parseInt($("#txt1").val());
 if (!isNaN(val)) {
 $("#progressbar").progressbar("option", "value", val);
 //$("#progressbar").progressbar("value", val);
 }
 else alert("Необходимо ввести число");
});
$("#btn2").click(function() {
 alert($("#progressbar").progressbar("option", "value"));
});
$("#btn3").click(function() {
 $("#progressbar").progressbar("disable");
});
$("#btn4").click(function() {
 $("#progressbar").progressbar("enable");
});
});
//-->
</script>
<style type="text/css">
 /* Выводим анимированную картинку вместо полосы */
 .ui-progressbar-value { background-image:url(animate.gif); }
</style>
</head>
<body style="font-size:8pt; font-family:Verdana">
<div style="width: 500px;"><div id="progressbar"></div></div>

Ведите новое значение: <input type="text" id="txt1">

<input type="button" value="Изменить значение" id="btn1">
<input type="button" value="Получить значение" id="btn2">

<input type="button" value="Запретить" id="btn3">
<input type="button" value="Разрешить" id="btn4">

<div id="div1"></div>
</body>
</html>

```

## 12.11. Модуль UI Slider — шкала с бегунком

Модуль UI Slider позволяет создать шкалу с бегунком, который можно перемещать с помощью указателя мыши. Для загрузки модуля переходим на страницу <http://jqueryui.com/download>. Оставляем установленными флагги UI Core, Slider, 1.7.2, выбираем тему из списка (например, UI lightness), а затем щелкаем на кнопке Download. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы следующие файлы:

- jquery-ui-1.7.2.custom.css и каталог images (расположены в папке css\ui-lightness);
- jquery-1.3.2.min.js (расположен в папке js);
- jquery-ui-1.7.2.custom.min.js (расположен в папке js).

Все указанные файлы необходимо подключить к скрипту в том порядке, в котором они перечислены. В качестве примера выведем две шкалы с разными параметрами настройки (листинг 12.29).

### Листинг 12.29. Модуль UI Slider, шкала с бегунком

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Slider. Шкала с бегунком</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/ui-lightness/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#slider1").slider({
 range: true,
 values: [10, 65]
 });
 $("#slider2").slider({
 range: false,
 value: 50,
 });
});
//--
</script>
</head>
<body style="font-size:8pt; font-family:Verdana">

<div style="width: 500px;">
<div id="slider1"></div>

<div id="slider2"></div>
</div>

</body>
</html>
```

Результат выполнения листинга показан на рис. 12.13.

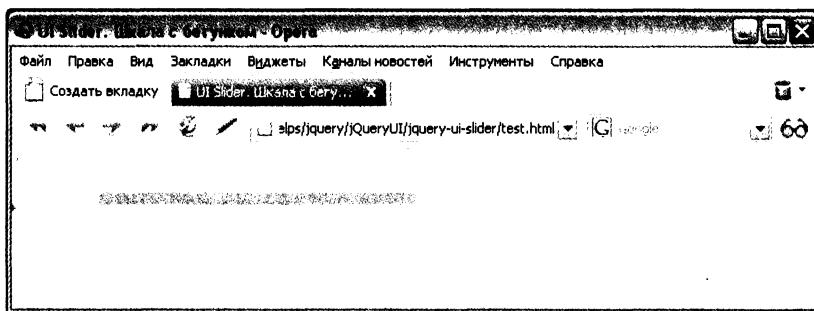


Рис. 12.13. Различные варианты шкалы

Как видно из примера, шаблоном для шкалы является элемент DIV.

```
<div id="slider1"></div>
```

Для создания шкалы с бегунком и управления ею предназначен метод `slider()`. Формат метода:

```
slider([<Объект с опциями>])
```

Параметр `<Объект с опциями>` представляет собой объект, состоящий из пар “опция/значение”. Могут быть указаны следующие опции.

- `value` — определяет положение бегунка на шкале. Если используется несколько бегунков, то опция задает значение только для первого. Значение по умолчанию — 0.
- `values` — позволяет задать положение нескольких бегунков. В качестве значения указывается массив. Если опция `range` имеет значение `true`, то массив должен состоять из двух элементов. Значение по умолчанию — `null`.
- `min` — минимальное значение шкалы. Значение по умолчанию — 0.
- `max` — максимальное значение шкалы. Значение по умолчанию — 100.
- `range` — позволяет указать область, которая будет выделена на шкале. Может принимать следующие значения:
  - ◊ `false` — без выделения (значение по умолчанию);
  - ◊ `true` — при использовании двух бегунков выделяется область между ними;
  - ◊ `min` — выделяется область слева от бегунка при горизонтальной ориентации и ниже при вертикальной;
  - ◊ `max` — выделяется область справа от бегунка при горизонтальной ориентации и выше при вертикальной.
- `orientation` — определяет расположение шкалы. Может принимать следующие значения:
  - ◊ `horizontal` — по горизонтали;
  - ◊ `vertical` — по вертикали;
  - ◊ `auto` — автоматическое определение ориентации (значение по умолчанию).
- `step` — позволяет задать шаг изменения значений шкалы. Значение по умолчанию — 1.
- `animate` — если указано значение `true`, то при щелчке на шкале бегунок будет перемещаться с анимацией. Значение по умолчанию — `false` (при щелчке на шкале бегунок перемещается моментально).

Первый формат метода `slider()` позволяет задать значения при создании компонента. Чтобы изменить или получить значения опций уже после создания компонента, необходимо использовать второй формат метода `slider()`. Для этого в первом параметре указывается значение “`option`”, во втором — название опции, а в третьем — новое значение.

```
$("#slider").slider("option", "value", 20);
```

Если необходимо получить значение опции, то третий параметр указывать не нужно. Получим местоположение всех бегунков на шкале.

```
var arr = $("#slider").slider("option", "values");
```

Второй формат метода `slider()` позволяет также управлять созданным компонентом. Для этого в первом параметре указываются следующие значения.

- `value` — позволяет получить или установить значение бегунка (если используется один бегунок). Формат метода:  
`slider("value" [, <Значение>])`
- `values` — позволяет получить или установить значения при использовании нескольких бегунков. Формат метода:  
`slider("values", <Индекс бегунка>, <Значение>)`
- `disable` — временно запрещает использование компонента.
- `enable` — разрешает использование компонента, если ранее оно было запрещено с помощью метода `disable`.
- `destroy` — удаляет компонент и возвращает все элементы в первоначальное состояние.

Обработать события, происходящие с компонентом, можно двумя способами.

```
slider({
 <Событие>: <Функция обратного вызова>
});

или
bind(<Событие>, <Функция обратного вызова>)
```

В параметре `<Событие>` могут быть указаны следующие события (в скобках указано значение для метода `bind()`):

- `start (slidestart)` — в начале перемещения бегунка;
- `slide (slide)` — возникает постоянно при перемещении бегунка;
- `stop (slidestop)` — в конце перемещения бегунка;
- `change (slidechange)` — происходит в конце перемещения бегунка, а также при изменении значения из программы с помощью методов `value` или `values`.

В параметре `<Функция обратного вызова>` указывается ссылка на функцию следующего формата.

```
function <Название функции>([<Объект event>[, <Объект UI>]]) {
 // ...
}
```

Через параметр `<Объект UI>` доступно два свойства:

- `value` — положение бегунка; если используется несколько бегунков, то свойство возвращает значение только для первого;
- `values` — значения всех бегунков через запятую (при использовании нескольких бегунков).

Создадим несколько компонентов с различными настройками. Добавим возможность изменения значений из программы, а также обработаем события, происходящие с компонентами (листинг 12.30).

### Листинг 12.30. Модуль UI Slider, опции, события и методы

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Slider. Опции, события и методы</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link type="text/css" rel="stylesheet"
 href="css/ui-lightness/jquery-ui-1.7.2.custom.css">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#slider1").slider({
 max: 200, // Максимальное значение шкалы
 min: 0, // Минимальное значение шкалы
 range: true, // Выделение участка между двумя бегунками
 values: [10, 65], // Местоположение бегунков
 orientation: "auto",
 step: 10, // Шаг
 animate: true // Плавное изменение положения бегунка
 });
 $("#slider2").slider({
 range: "min", // Выделение участка слева от бегунка
 value: 40, // Начальное положение бегунка на шкале
 orientation: "horizontal", // Горизонтальное расположение шкалы
 change: function(e, ui) {
 alert("Установлено значение: " + ui.value);
 }
 });
 $("#slider3").slider({
 values: [25, 50, 75], // Начальное положение трех бегунков
 change: function(e, ui) {
 alert("Установлены значения: " + ui.values);
 }
 });
 $("#slider4").slider({
 range: "max", // Выделение участка выше бегунка
 value: 60, // Начальное положение бегунка на шкале
 orientation: "vertical", // Вертикальное расположение шкалы
 start: function() {
 $("#div1").append("Событие start
");
 },
 slide: function() {
 $("#div1").append("Событие slide
");
 },
 change: function() {
```

```

 $("#div1").append("Событие change
") ;
 },
 stop: function() {
 $("#div1").append("Событие stop
");
 }
}).bind("slidestart", function() {
 $("#div1").html("Событие slidestart
") ;
}).bind("slide", function() {
 $("#div1").append("Событие slide (bind)
") ;
}).bind("slidechange", function() {
 $("#div1").append("Событие slidechange
") ;
}).bind("slidestop", function() {
 $("#div1").append("Событие slidestop
") ;
});
$("#btn1").click(function() {
 var val = parseInt($("#txt1").val());
 if (!isNaN(val)) {
 // $("#slider2").slider("option", "value", val);
 $("#slider2").slider("value", val);
 }
 else alert("Необходимо ввести число");
});
$("#btn2").click(function() {
 alert($("#slider2").slider("option", "value"));
 // alert($("#slider2").slider("value"));
});
$("#btn3").click(function() {
 var val = parseInt($("#txt2").val());
 if (!isNaN(val)) {
 $("#slider3").slider("values", 1, val);
 }
 else alert("Необходимо ввести число");
});
$("#btn4").click(function() {
 var arr = $("#slider3").slider("option", "values");
 alert(arr[1]);
 // alert($("#slider3").slider("values", 1));
});
});
//-->
</script>
</head>
<body style="font-size:8pt; font-family:Verdana">

<div style="width: 600px;">
<div id="slider1"></div>

<div id="slider2"></div>

<input type="text" id="txt1">
<input type="button" value="Изменить положение бегунка" id="btn1">
<input type="button" value="Получить значение" id="btn2">

<div id="slider3"></div>

<input type="text" id="txt2">
<input type="button" value="Изменить положение второго бегунка"
id="btn3">
<input type="button" value="Получить значение" id="btn4">


```

```
<div id="slider4"></div>
</div>

<div id="div1"></div>
</body>
</html>
```

## 12.12. Модуль UI Effects – визуальные эффекты

Модуль UI Effects предоставляет множество визуальных эффектов, а также добавляет возможность плавного изменения цвета и управления классами стилей. Все базовые методы расположены в модуле Effects Core. Чтобы плавно изменять цвета или управлять классами стилей, достаточно подключить только модуль Effects Core. Сами же эффекты можно подключать по мере надобности.

Для загрузки переходим на страницу <http://jqueryui.com/download>. Устанавливаем флагшки Effects Core, 1.7.2, а также флагшки всех эффектов. Затем щелкаем на кнопке Download. Распаковываем архив в отдельную папку. Из этого архива для работы модуля необходимы два файла:

- jquery-1.3.2.min.js (расположен в папке js);
- jquery-ui-1.7.2.custom.min.js (расположен в папке js).

### 12.12.1. Плавное изменение цвета

При рассмотрении метода animate() говорилось, что метод позволяет создавать произвольную анимацию только за счет изменения числовых атрибутов (например, height, width, opacity и др.). Модуль Effects Core добавляет возможность плавного изменения значений атрибутов, задающих цвет, — backgroundColor, borderBottomColor, borderLeftColor, borderRightColor, borderTopColor, color и outlineColor. Цвет может быть задан одним из трех способов:

- шестнадцатеричным значением, например #008800;
- с помощью функции rgb(), например rgb(0, 88, 0);
- названием цвета, например "green".

Пример использования метода animate() для плавного изменения цвета приведен в листинге 12.31.

#### Листинг 12.31. Модуль UI Effects, плавное изменение цвета

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Effects. Плавное изменение цвета</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<style type="text/css">
```

```

body { font-size:10pt; font-family:Verdana, sans-serif; }
#div1 {
 position:absolute; top:50px; left:50px; width:100px; height:100px;
 padding:2px; border:1px solid #000000; text-align:center;
 background-color:#000000; color:#e8e8e8;
}
</style>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#btn1").click(function() {
 var color = $("#txt1").val();
 if (color.length == 0) alert("Не заполнено поле");
 else $("#div1").animate({ backgroundColor: color }, 2000);
 });
//-->
</script>
</head>
<body>
Цвет: <input type="text" id="txt1">
<input type="button" value="Задать значение цвета"
id="btn1">

<div id="div1">Плавное изменение цвета</div>
</body>
</html>

```

## 12.12.2. Управление классами стилей

Модуль Effects Core предоставляет также возможность плавно и последовательно изменять значения атрибутов стилевого класса путем указания названия класса и продолжительности операции. Эта возможность отличает методы модуля Effects Core от стандартных методов библиотеки jQuery, где стилевой класс применяется сразу. Для управления классами стилей предназначены следующие методы.

- `addClass()` — добавляет указанный класс всем элементам коллекции. Формат метода:

```
addClass(<Название класса>[, <Продолжительность>[,
 <Функция обратного вызова>]])
```

- `removeClass()` — удаляет указанный класс у всех элементов коллекции. Формат метода:

```
removeClass([<Название класса>[, <Продолжительность>[,
 <Функция обратного вызова>]]])
```

- `toggleClass()` — добавляет указанный класс всем элементам коллекции, если он не был определен ранее, или удаляет указанный класс, если он был добавлен ранее. Формат метода:

```
toggleClass(<Название класса>[, <Продолжительность>])
```

- `switchClass()` — удаляет один класс и добавляет другой. Формат метода:

```
switchClass(<Удаляемый класс>, <Новый класс>[,
 <Продолжительность>])
```

В необязательном параметре *<Продолжительность>* может быть указано время выполнения операции в миллисекундах или следующие значения:

- *fast* — 200 миллисекунд;
- *normal* — 400 миллисекунд;
- *slow* — 600 миллисекунд.

В параметре *<Функция обратного вызова>* может быть указана ссылка на функцию, которая будет вызвана после окончания операции. Формат функции:

```
function <Название функции>() {
 // ...
}
```

Рассмотрим управление классами стилей на примере (листинг 12.32).

### Листинг 12.32. Модуль UI Effects, управление классами стилей

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Effects. Управление классами стилей</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js"
type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<style type="text/css">
body { font-size:10pt; font-family:Verdana, sans-serif; }
#div1 {
 position:absolute; top:50px; left:50px; width:100px;
 height:100px;
 padding:2px; border:1px solid #000000; text-align:center;
}
.cls1 { background-color:#000000; color:#ffffff; }
.cls2 { background-color:#FF0000; color:#008800; }
</style>
<script type="text/javascript">
<!--
$(document).ready(function() {
 $("#btn1").click(function() { // Добавить класс
 var elem = $("#div1");
 if (!elem.is(".cls1") && !elem.is(".cls2"))
 elem.addClass("cls1", 1000);
 });
 $("#btn2").click(function() { // Удалить класс
 var elem = $("#div1");
 if (elem.is(".cls1")) elem.removeClass("cls1", 1000);
 else if (elem.is(".cls2")) elem.removeClass("cls2", 1000);
 });
 $("#btn3").click(function() { // Переключить
 var elem = $("#div1");
 if (elem.is(".cls2")) elem.removeClass("cls2", 1000);
 else elem.toggleClass("cls1", 1000);
 });
});
```

```

});
$("#btn4").click(function() { // Заменить класс
 var elem = $("#div1");
 if (elem.is(".cls1")) elem.switchClass("cls1", "cls2", 1000);
 else if (elem.is(".cls2")) elem.switchClass("cls2", "cls1",
1000);
 else elem.addClass("cls2", 1000);
});
//-->
</script>
</head>
<body>
<input type="button" value="Добавить класс" id="btn1">
<input type="button" value="Удалить класс" id="btn2">
<input type="button" value="Переключить" id="btn3">
<input type="button" value="Заменить класс" id="btn4">

<div id="div1">Управление классами стилей</div>
</body>
</html>

```

После щелчка на кнопке **Добавить класс** вначале плавно изменится цвет фона с белого на черный, а затем поменяется цвет текста с черного на белый. После щелчка на кнопке **Удалить класс** также плавно будет удален стилевой класс, который был добавлен ранее. Переключение и замена классов производятся с помощью кнопок **Переключить** и **Заменить класс** соответственно.

### 12.12.3. Методы, позволяющие использовать эффекты

Главное преимущество модуля Effects Core — это, конечно же, эффекты. Использовать эффекты позволяют следующие методы.

- `show()` — использует эффект для отображения элемента. Формат метода:

```
show(<Эффект> [, <Объект с опциями эффекта> [, <Продолжительность> [, <Функция обратного вызова>]]])
```

В параметре `<Эффект>` могут быть указаны следующие эффекты: `blind`, `bounce`, `clip`, `drop`, `explode`, `fold`, `puff`, `pulsate`, `scale`, `slide`.

- `hide()` — использует эффект для скрытия элемента. Формат метода:

```
hide(<Эффект> [, <Объект с опциями эффекта> [, <Продолжительность> [, <Функция обратного вызова>]]])
```

В параметре `<Эффект>` могут быть указаны следующие эффекты: `blind`, `bounce`, `clip`, `drop`, `explode`, `fold`, `puff`, `pulsate`, `scale`, `slide`.

- `toggle()` — позволяет чередовать скрытие и отображение элементов. Если элемент скрыт, то он будет отображен, и наоборот. Формат метода:

```
toggle(<Эффект> [, <Объект с опциями эффекта> [, <Продолжительность> [, <Функция обратного вызова>]]])
```

В параметре `<Эффект>` могут быть указаны следующие эффекты: `blind`, `bounce`, `clip`, `drop`, `explode`, `fold`, `puff`, `pulsate`, `scale`, `slide`.

- `effect()` — позволяет выполнять произвольные эффекты. Формат метода:

```
effect(<Эффект>[, <Объект с опциями эффекта>[,
 <Продолжительность>[, <Функция обратного вызова>]]])
```

В параметре `<Эффект>` могут быть указаны следующие эффекты: `blind`, `bounce`, `clip`, `drop`, `explode`, `fold`, `highlight`, `puff`, `pulsate`, `scale`, `shake`, `size`, `slide`, `transfer`.

В необязательном параметре `<Продолжительность>` указывается время выполнения эффекта в миллисекундах или следующие значения:

- `fast` — 200 миллисекунд;
- `normal` — 400 миллисекунд;
- `slow` — 600 миллисекунд.

В параметре `<Функция обратного вызова>` может быть указана ссылка на функцию, которая будет вызвана по окончании эффекта. Формат функции:

```
function <Название функции>() {
 // ...
}
```

Пример использования методов `show()`, `hide()` и `toggle()` приведен в листинге 12.33.

### Листинг 12.33. Модуль UI Effects, методы `show()`, `hide()` и `toggle()`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Effects. Методы show(), hide() и toggle()</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
 type="text/javascript"></script>
<script type="text/javascript">
!--
$(document).ready(function() {
 $("#btn1").click(function() {
 $("#img1").show($("#effect").val(), {}, 2000);
 });
 $("#btn2").click(function() {
 $("#img1").hide($("#effect").val(), {}, 2000);
 });
 $("#btn3").click(function() {
 $("#img1").toggle($("#effect").val(), {}, 2000);
 });
});
//-->
</script>
</head>
<body>
Выберите эффект из списка и нажмите одну из кнопок

<select id="effect">
```

```

<option value="blind">Blind</option>
<option value="clip">Clip</option>
<option value="drop">Drop</option>
<option value="explode">Explode</option>
<option value="fold">Fold</option>
<option value="puff">Puff</option>
<option value="pulsate">Pulsate</option>
<option value="scale">Scale</option>
<option value="slide">Slide</option>
</select>
<input type="button" value="show()" id="btn1">
<input type="button" value="hide()" id="btn2">
<input type="button" value="toggle()" id="btn3">

</body>
</html>

```

## 12.12.4. Эффекты

В параметре <Эффект> метода `effect()` (и некоторых других методов) могут быть указаны следующие эффекты.

- `blind` — свертывание справа налево или снизу вверх или развертывание в противоположных направлениях. Рассмотрим опции эффекта.
  - ◊ `mode` — режим эффекта; возможные значения — `"show"` (показать) или `"hide"` (скрыть); значение по умолчанию — `"hide"`.
  - ◊ `direction` — направление движения; возможные значения — `"vertical"` (по вертикали) или `"horizontal"` (по горизонтали); значение по умолчанию — `"vertical"`.
- `bounce` — перемещение элемента по горизонтали или вертикали заданное количество раз с затуханием (или нарастанием) колебаний. Рассмотрим опции эффекта.
  - ◊ `mode` — режим эффекта; возможные значения — `"show"` (показать), `"hide"` (скрыть) или `"effect"`; значение по умолчанию — `"effect"`.
  - ◊ `direction` — направление движения; возможные значения — `"up"` (вверх), `"down"` (вниз), `"left"` (влево) или `"right"` (вправо). Если опция `mode` имеет значение `"effect"`, то опция `direction` задает начальное направление движения (движение с затуханием колебаний). Если опция `mode` имеет значение `"show"`, то задается, откуда появится элемент (движение с затуханием колебаний), а при значении `"hide"` — в какую сторону он исчезнет (движение с нарастанием колебаний). Значение по умолчанию — `"up"`.
  - ◊ `distance` — максимальная дистанция (в пикселях); значение по умолчанию — `20`.

◊ times — количество колебаний; значение по умолчанию — 5.

```
$("#div1").effect("bounce", {
 mode: "hide", // Скрыть элемент в конце эффекта
 direction: "right", // Элемент исчезнет вправо
 distance: 50, // Дистанция
 times: 5 // Количество колебаний
}, 500);
```

- clip — свертывание элемента с двух сторон к центру или развертывание от центра (эффект включения или выключения старого телевизора). Рассмотрим опции эффекта.

◊ mode — режим эффекта; возможные значения — "show" (показать) или "hide" (скрыть); значение по умолчанию — "hide".

◊ direction — направление движения; возможные значения — "vertical" (по вертикали) или "horizontal" (по горизонтали); значение по умолчанию — "vertical".

```
$("#div1").effect("clip", {
 mode: "show",
 direction: "horizontal"
}, 1000);
```

- drop — растворение элемента с перемещением в заданную сторону или проявление. Рассмотрим следующие опции эффекта.

◊ mode — режим эффекта; возможные значения — "show" (показать) или "hide" (скрыть); значение по умолчанию — "hide".

◊ direction — направление движения. Если опция mode имеет значение "show", то задается, с какой стороны появится элемент, а при значении "hide" — куда он исчезнет. Возможные значения — "up" (вверх), "down" (вниз), "left" (влево) или "right" (вправо); значение по умолчанию — "left".

```
$("#div1").effect("drop", {
 mode: "hide",
 direction: "up"
}, 1000);
```

- explode — элемент рассыпается на части во всех направлениях или собирается из множества частей. Рассмотрим опции эффекта.

◊ mode — режим эффекта; возможные значения — "show" (показать) или "hide" (скрыть); значение по умолчанию — "hide".

◊ pieces — количество частей; значение по умолчанию — 9.

```
$("#div1").effect("explode", {
 mode: "show",
 pieces: 18
}, 1000);
```

- fold — сворачивает элемент сначала частично по вертикали, а затем полностью по горизонтали или, наоборот, сначала разворачивает элемент частично по горизонтали, а затем — полностью по вертикали. Опции эффекта следующие.

- ◊ mode — режим эффекта; возможные значения — "show" (показать) или "hide" (скрыть); значение по умолчанию — "hide".
  - ◊ size — размер (в пикселях) части, оставляемой при начальном частичном свертывании; значение по умолчанию — 15.
  - ◊ horizFirst — если указано значение true, то начальное свертывание будет не по вертикали (снизу вверх), а по горизонтали (справа налево). Окончательное свертывание будет производиться снизу вверх. Значение по умолчанию — false.
- ```
$("#div1").effect("fold", {
    size: 30, // Ширина оставляемой части
    horizFirst: true // Начальное свертывание по горизонтали
}, 1000);
```
- highlight — эффект за счет изменения цвета фона. Рассмотрим опции эффекта.
 - ◊ mode — режим эффекта; возможные значения — "show" (показать) или "hide" (скрыть); значение по умолчанию — "show".
 - ◊ color — цвет фона; значение по умолчанию — "#fffff99".

```
$("#div1").effect("highlight", { color: "#ff0000" }, 300);
```
 - puff — элемент увеличивается в размерах и растворяется или, наоборот, из полупрозрачного большого размера уменьшается до нормальных размеров. Рассмотрим опции эффекта.
 - ◊ mode — режим эффекта; возможные значения — "show" (показать) или "hide" (скрыть); значение по умолчанию — "hide".
 - ◊ percent — размер в процентах; значение по умолчанию — 150.

```
$("#div1").hide("puff", { percent: 200 }, 1000);
```
 - pulsate — мигание элемента заданное количество раз за счет изменения прозрачности элемента. Рассмотрим опции эффекта.
 - ◊ mode — режим эффекта; возможные значения — "show" (показать) или "hide" (скрыть); значение по умолчанию — "show".
 - ◊ times — количество миганий; значение по умолчанию — 5.

```
$("#div1").effect("pulsate", {
    mode: "hide",
    times: 2
}, 1000);
```
 - scale — сворачивает элемент в точку или разворачивает его из точки. Опции эффекта следующие.
 - ◊ mode — режим эффекта; возможные значения — "effect" (просто эффект), "show" (показать) или "hide" (скрыть); значение по умолчанию — "effect".
 - ◊ direction — направление движения; возможные значения — "both" (в точку), "vertical" (по вертикали) или "horizontal" (по горизонтали); значение по умолчанию — "both".

- ◊ `from` — начальные размеры элемента. Указывается объект с двумя свойствами: `height` и `width`. По умолчанию используются текущие размеры элемента.
- ◊ `origin` — позволяет указать точку исчезновения (или появления). В качестве значения указывается массив. Значение по умолчанию — `["middle", "center"]`.
- ◊ `percent` — конечный размер элемента в процентах; значение по умолчанию — 0.
- ◊ `scale` — определяет, какие части элемента будут изменяться. Возможные значения — `"both"` (каждая точка), `"box"` (изменяются размеры границ и внутренние отступы) и `"content"` (изменяются размеры любого содержимого элемента). Значение по умолчанию — `"both"`.


```
$("#div1").effect("scale", {
    mode: "hide", // Скрываем элемент
    direction: "both", // Сворачиваем в точку
    // Сворачиваем в правый верхний угол
    origin: ["top", "right"],
    percent: 0, // Конечный размер в процентах
    scale: "content"
}, 3000);
```
- `shake` — вибрация элемента по горизонтали или вертикали заданное количество раз. Рассмотрим опции эффекта.
 - ◊ `direction` — начальное направление движения; возможные значения — `"up"` (вверх), `"down"` (вниз), `"left"` (влево) или `"right"` (вправо); значение по умолчанию — `"left"`.
 - ◊ `distance` — максимальная дистанция (в пикселях); значение по умолчанию — 20.
 - ◊ `times` — количество перемещений; значение по умолчанию — 3.

```
$("#div1").effect("shake", {
  direction: "up", // Элемент начнет движение вверх
  distance: 15, // Дистанция
  times: 5 // Количество перемещений
}, 500);
```
- `size` — уменьшение (или увеличение) размеров элемента. Опции эффекта следующие.
 - ◊ `from` — начальные размеры элемента. Указывается объект с двумя свойствами: `height` и `width`. По умолчанию используются текущие размеры элемента.
 - ◊ `to` — конечные размеры элемента. Указывается объект с двумя свойствами: `height` и `width`.
 - ◊ `origin` — позволяет указать точку исчезновения (или появления). В качестве значения указывается массив. Значение по умолчанию — `["middle", "center"]`.
 - ◊ `scale` — определяет, какие части элемента будут изменяться. Возможные значения — `"both"` (каждая точка), `"box"` (изменяются размеры границ и

внутренние отступы) и "content" (изменяются размеры любого содержимого элемента). Значение по умолчанию — "both".

```
$("#div1").effect("size", {
    to: { height: 0, width: 0 }, // Сворачиваем в точку
    // Сворачиваем в левый нижний угол
    origin: ["bottom", "left"],
    scale: "both"
}, 3000);
```

- **slide** — сдвигание элемента в определенную сторону или выдвижение с какой-либо стороны. Опции эффекта следующие.

- ◊ mode — режим эффекта; возможные значения — "show" (показать) или "hide" (скрыть); значение по умолчанию — "show".
- ◊ direction — направление движения. Если опция mode имеет значение "show", то задается, с какой стороны появится элемент, а при значении "hide" — куда он исчезнет. Возможные значения — "up" (вверх), "down" (вниз), "left" (влево) или "right" (вправо); значение по умолчанию — "left".
- ◊ distance — позволяет указать расстояние, на которое сдвинется элемент. По умолчанию используется ширина или высота элемента (в зависимости от значения опции direction).

```
$("#div1").effect("slide", {
    mode: "hide", // Скрываем элемент
    direction: "up", // Сдвигаем вверх
    distance: 50 // Сдвинется на 50px вверх
}, 1000);
```

- **transfer** — трансформация одного элемента в другой. Рассмотрим опции эффекта.
 - ◊ className — стилевой класс, который будет добавлен к перемещаемому элементу; по умолчанию добавляется класс ui-effects-transfer.
 - ◊ to — ссылка на элемент (селектор jQuery), в который происходит трансформация.

Пример использования метода `effect()` и эффекта `transfer` приведен в листинге 12.34.

Листинг 12.34. Модуль UI Effects, эффект Transfer

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>UI Effects. Эффект Transfer</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script src="js/jquery-1.3.2.min.js" type="text/javascript"></script>
<script src="js/jquery-ui-1.7.2.custom.min.js"
    type="text/javascript"></script>
<style type="text/css">
body { font-size:10pt; font-family:Verdana, sans-serif; }
#div1 {
```

```

width:100px; height:100px;
padding:2px; border:1px solid #000000; text-align:center;
background-color:#000000; color:#ffffff;
}
#div2 {
position:absolute; top:250px; left:450px; width:50px; height:50px;
border:1px solid #000000; background-color:#0000ff;
}
.cls1 { border:1px dotted #000000; background-color:#ff0000; }
</style>
<script type="text/javascript">
<!--
$(document).ready(function() {
    $("#div1").click(function() {
        $(this).effect("transfer", {className: "cls1",
        to: "#div2"}, 2000);
    });
} );
//-->
</script>
</head>
<body>
<div id="div1">Щелкните мышью на этом блоке</div>
<div id="div2"></div>
</body>
</html>

```

Несколько слов в заключение

Вот и закончилось наше путешествие в мир jQuery. Материал этой книги описывает лишь базовые возможности библиотек jQuery и jQuery UI. В этом заключении мы рассмотрим, где можно найти дополнительную информацию.

Первым и самым важным источником информации является сайт <http://jquery.com/>. На нем вы найдете последнюю версию библиотеки jQuery, новости, а также ссылки на другие ресурсы в Интернете.

На сайтах <http://docs.jquery.com/> и <http://jqueryui.com/> размещена документация, которая обновляется в режиме реального времени. Библиотеки постоянно совершенствуются, появляются новые функции, изменяются параметры, добавляются компоненты и т.д. Регулярно посещайте эти сайты, и вы получите самую последнюю информацию.

На странице <http://docs.jquery.com/Tutorials> публикуются ссылки на учебные материалы, расположенные на других ресурсах в Интернете. Это — и перевод официальной документации на практически все языки мира (в том числе и на русский), а также различные обучающие статьи, описывающие тонкости работы с библиотекой jQuery.

Мир jQuery включает также множество самых разнообразных модулей, созданных сообществом программистов и доступных для свободного скачивания. На сайте <http://plugins.jquery.com/> вы найдете более 1500 готовых модулей, которые сможете использовать в своих веб-приложениях. Модули отсортированы по категориям, так что найти нужный модуль не составит особых проблем. При выборе модуля следует учитывать версию библиотеки jQuery. Советую также протестировать возможности в различ-

ных браузерах. Модули пишут программисты с разным уровнем подготовки, поэтому возможны ошибки. Вполне возможно, что когда-нибудь и вы сами станете автором какого-либо модуля.

И наконец, не следует забывать, что сам Интернет предоставляет множество ответов на самые разнообразные вопросы. Достаточно в строке запроса поискового портала (например, <http://yandex.ru>, <http://www.google.com/>, <http://www.rambler.ru/>) набрать свой вопрос. Наверняка, уже кто-то сталкивался с подобным вопросом и описал решение на каком-либо сайте. Задать свой вопрос можно также на различных форумах (например, советую регулярно посещать <http://forum.vingrad.ru/>) или на форуме моего сайта <http://wwwadmin.ru/forum/>.

Предметный указатель

\$

\$(), 13; 14; 15; 17; 37
\$.ajax(), 144; 147; 152; 157
\$.ajaxSetup(), 155; 157
\$.boxModel, 112
\$.browser, 111
\$.data(), 109
\$.datepicker.ATOM, 233
\$.datepicker.COOKIE, 233
\$.datepicker.iso9601Week(), 238
\$.datepicker.ISO_8601, 233
\$.datepicker.parseDate(), 238
\$.datepicker.regional, 231
\$.datepicker.RFC_1036, 233
\$.datepicker.RFC_1123, 233
\$.datepicker.RFC_2822, 233
\$.datepicker.RFC_822, 233
\$.datepicker.RFC_850, 233
\$.datepicker.RSS, 233
\$.datepicker.setDefaults(), 230
\$.datepicker.TIMESTAMP, 233
\$.datepicker.W3C, 233
\$.each(), 103
\$.extend(), 115
\$.fn.extend(), 114
\$.get(), 143; 144
\$.getJSON(), 139; 141
\$.getScript(), 142
\$.grep(), 104
\$.inArray(), 106
\$.makeArray(), 106
\$.map(), 105
\$.merge(), 106
\$.noConflict(), 14
\$.param(), 101
\$.post(), 146
\$.removeData(), 110

\$.support, 112
\$.support.boxModel, 113
\$.trim(), 109
\$.unique(), 107

A

accordion(), 203; 206
add(), 29
addClass(), 48; 249
after(), 39; 40; 41
AJAX, 117; 135
 данные с другого домена, 141; 142; 153
ajaxComplete(), 157
ajaxError(), 156
ajaxSend(), 156
ajaxStart(), 156
ajaxStop(), 157
ajaxSuccess(), 156
andSelf(), 34
animate(), 83; 248
append(), 28; 38; 39; 41
appendTo(), 16; 39; 41
attr(), 28; 50

B

before(), 39; 40; 41
bind(), 66; 72; 157; 170
blur(), 64
BOM, 11
Byte Order Mark, 11

C

change(), 64
children(), 30
click(), 59
clone(), 42
closest(), 32
contents(), 34
css(), 44

D

data(), 110; 212
datepicker(), 225
dblclick(), 60
dequeue(), 89
dialog(), 217; 221
die(), 73
disableSelection(), 179
draggable(), 163; 170
droppable(), 174; 176

E

each(), 27; 28; 54
effect(), 252; 253; 257
Effects Core, 248
empty(), 42
end(), 34
eq(), 26; 37; 54
error(), 59

F

fadeIn(), 81
fadeOut(), 81
fadeTo(), 81
filter(), 30
find(), 30
focus(), 64

G

get(), 25; 36; 37; 108

H

hasClass(), 50
height(), 45; 46
hide(), 79; 251
hover(), 49; 61
html(), 26; 27; 38; 48

I

index(), 26
innerHeight(), 46
innerWidth(), 46
insertAfter(), 40; 41
insertBefore(), 40; 41
is(), 35

J

jQuery UI, 161
Accordion, 201
Datepicker, 224
Dialog, 215
Draggable, 161
Droppable, 172
Effects, 248
Progressbar, 239
Resizable, 195
Selectable, 190
Slider, 242
Sortable, 179
Tabs, 208
jQuery UI, методы
abort, 212
activate, 206
add, 211
cancel, 187
close, 221
destroy, 170; 176; 187; 193; 199; 206;
212; 221; 234; 241; 245
dialog, 233
disable, 170; 176; 187; 192; 199; 207;
212; 221; 234; 241; 245
enable, 170; 176; 187; 193; 199; 207;
212; 221; 234; 241; 245
getDate, 234

hide, 234
isDisabled, 234
isOpen, 221
length, 211
load, 212
moveToTop, 221
open, 221
refresh, 192
remove, 211
rotate, 212
select, 211
serialize, 186
setDate, 234
show, 234
toArray, 186
url, 212
value, 245
values, 245

jQuery UI, опции

- accept, 174
- active, 203
- activeClass, 174
- addClasses, 164; 174
- ajaxOptions, 211
- alsoResize, 198
- altField, 230
- altFormat, 230
- animate, 198; 244
- animated, 204
- animateDuration, 198
- animateEasing, 198
- appendText, 230
- appendTo, 166; 183
- aspectRatio, 197
- autoHeight, 203
- autoHide, 198
- autoOpen, 217
- autoRefresh, 192
- axis, 164; 181
- bgiframe, 218
- buttonImage, 230
- buttonImageOnly, 230
- buttons, 218
- buttonText, 230
- cache, 211
- cancel, 167; 183; 192; 198
- changeMonth, 227
- changeYear, 227
- clearStyle, 204
- closeOnEscape, 218
- closeText, 227
- collapsible, 203; 210
- connectToSortable, 167
- connectWith, 184
- constraintInput, 225
- containment, 164; 181; 197
- cookie, 210
- currentText, 227
- cursor, 164; 181
- cursorAt, 164; 181
- dateFormat, 226
- dayNames, 225; 239
- dayNamesMin, 226
- dayNamesShort, 226; 239
- defaultDate, 229
- delay, 165; 182; 192; 197
- dialogClass, 218
- disabled, 210
- distance, 165; 182; 192; 197
- draggable, 218
- dropOnEmpty, 184
- duration, 225
- event, 203; 210
- fillSpace, 204
- filter, 192
- firstDay, 226
- forcePlaceholderSize, 184
- fx, 210
- ghost, 197
- gotoCurrent, 227
- greedy, 174
- grid, 165; 182; 197
- handle, 167; 183
- handles, 198
- header, 203
- height, 218
- helper, 166; 182; 197
- hide, 218
- hideIfNoPrevNext, 230
- hoverClass, 174
- icons, 203
- iframeFix, 167

isRTL, 226
items, 181
max, 244
maxDate, 229
maxHeight, 197; 218
maxWidth, 197; 218
min, 244
minDate, 229
minHeight, 197; 218
minWidth, 197; 218
modal, 217
monthNames, 226; 239
monthNamesShort, 226; 239
navigation, 204
navigationAsDateFormat, 226
navigationFilter, 204
nextText, 226
numberOfMonths, 228
opacity, 164; 181
orientation, 244
panelTemplate, 211
placeholder, 183
position, 217
prevText, 226
range, 244
refreshPositions, 167
resizable, 218
revert, 164; 181
revertDuration, 164
scope, 167; 172; 174
scroll, 166; 182
scrollSensitivity, 166; 182
scrollSpeed, 166; 182
selected, 210
shortYearCutoff, 226; 239
show, 218
showAnim, 225
showButtonPanel, 227
showCurrentAtPos, 228
showMonthAfterYear, 229
showOn, 230
showOptions, 225
showOtherMonths, 229
snap, 165
snapMode, 166
snapTolerance, 166
spinner, 211
stack, 166; 218
step, 244
stepMonths, 228
tabTemplate, 211
title, 217
tolerance, 174; 183; 192
value, 240; 244
values, 244
width, 218
yearRange, 227
zIndex, 166; 183; 218

jQuery UI, опции эффектов

- className, 257
- color, 255
- direction, 253; 254; 255; 256; 257
- distance, 253; 256; 257
- from, 256
- horizFirst, 255
- mode, 253; 254; 255; 257
- origin, 256
- percent, 255; 256
- pieces, 254
- scale, 256
- size, 255
- times, 254; 255; 256
- to, 256; 257

jQuery UI, события

- accordionchange, 207
- accordionchangestart, 207
- activate, 177; 187
- add, 214
- beforeclose, 222
- beforeShow, 236
- beforeShowDay, 236
- beforeStop, 187
- change, 187; 207; 241; 245
- changestart, 207
- close, 222
- deactivate, 177; 188
- dialogbeforeclose, 222
- dialogclose, 222
- dialogfocus, 222
- dialogopen, 222
- disable, 214
- drag, 170; 222

dragstart, 170; 222
dragstop, 170; 222
drop, 177
dropactivate, 177
dropdeactivate, 177
dropout, 177
dropover, 177
enable, 214
focus, 222
load, 214
onChangeMonthYear, 236
onClose, 236
onSelect, 236
open, 222
out, 177; 187
over, 177; 187
progressbarchange, 241
receive, 188
remove, 188; 214
resize, 199; 222
resizestart, 199; 222
resizestop, 199; 222
select, 214
selectableselected, 193
selectableselecting, 193
selectablestart, 193
selectablestop, 193
selectableunselected, 193
selectableunselecting, 193
selected, 193
selecting, 193
show, 214
slide, 245
slidechange, 245
slidestart, 245
slidestop, 245
sort, 187
sortactivate, 187
sortbeforeStop, 187
sortchange, 187
sortdeactivate, 188
sortout, 187
sortover, 187
sortreceive, 188
sortremove, 188
sortstart, 187
sortstop, 188
sortupdate, 188
start, 170; 187; 193; 199; 245
stop, 170; 188; 193; 199; 245
tabsadd, 214
tabsdisable, 214
tabsenable, 214
tabsload, 214
tabsremove, 214
tabsselect, 214
tabsshow, 214
unselected, 193
unselecting, 193
update, 188

jQuery UI, эффекты
blind, 253
bounce, 253
clip, 254
drop, 254
explode, 254
fold, 254
highlight, 255
puff, 255
pulsate, 255
scale, 255
shake, 256
size, 256
slide, 257
transfer, 257

jQuery(), 13; 14; 15
jQuery, опции
async, 153
beforeSend, 154
cache, 153
complete, 86; 155
contentType, 153
data, 152
dataFilter, 155
dataType, 152
duration, 86
easing, 86
error, 155
global, 157
ifModified, 154
jsonp, 154
password, 154

processData, 153
queue, 86
scriptCharset, 154
success, 155
timeout, 153
type, 152
url, 152
username, 154
JSON, 131; 139; 153
JSONP, 135; 139; 141; 153; 154

K

keydown(), 63
keypress(), 63
keyup(), 63; 92

L

length, 25
live(), 72
load(), 57; 135; 137

M

map(), 35; 36
Microsoft.XMLHTTP, 119
mousedown(), 60
mouseenter(), 67
mouseleave(), 67
mousemove(), 61
mouseout(), 61
mouseover(), 61
mouseup(), 60
MSXML2.XMLHTTP, 119

N

next(), 33
nextAll(), 33
noConflict(), 14
not(), 29
Notepad++, 11

O
offset(), 54
one(), 70
outerHeight(), 46
outerWidth(), 46

P

parent(), 31
parents(), 32
position(), 54
prepend(), 38; 39; 41
prependTo(), 39; 41
prev(), 33
prevAll(), 34
progressbar(), 240

Q

queue(), 89

R

ready(), 13; 57
remove(), 42
removeAttr(), 52
removeClass(), 48; 249
removeData(), 110; 111
replaceAll(), 43
replaceWith(), 43
resizable(), 196; 198
resize(), 58

S

scroll(), 58
scrollLeft(), 55
scrollTop(), 55
select(), 65
selectable(), 192
serialize(), 100
serializeArray(), 100
show(), 79; 251
siblings(), 32
size(), 25
slice(), 26
slideDown(), 79

slider(), 244
slideToggle(), 79
slideUp(), 79
sortable(), 181; 185
stop(), 82; 87
submit(), 64
switchClass(), 249

T

tabs(), 210; 211
text(), 25; 37
toggle(), 59; 79; 251
toggleClass(), 48; 50; 249
trigger(), 68; 69; 70; 72; 73
triggerHandler(), 68; 69; 70; 71; 72; 73

U

unbind(), 71
unload(), 58

V

val(), 22; 48; 52

W

width(), 45; 46
wrap(), 40
wrapAll(), 41
wrapInner(), 40

X

XML, 127; 147; 148
XMLHttpRequest, 119

A

Аккордеон, 201
Анимация
изменение цвета, 248
прерывание, 87
создание, 83
управление очередью, 89
Атрибуты CSS
высота элемента, 45; 46

задать значение, 44
нескольких атрибутов, 44
получение значения, 44
ширина элемента, 45; 46

B

Вложение элементов, 40
Выделение элементов, 190
Высота элемента, 45
Вычисление положения элементов, 54

Д

Диалоговые окна, 215
Добавление
класса, 48
содержимого
перед элементом, 39
после элемента, 39
элементов, 29
Доступ
к параметрам тегов, 50
к элементу по индексу, 25

З

Замена элемента, 43

И

Изменение
атрибутов CSS, 44
параметров тегов, 50
прозрачности, 81
размеров, 195
содержимого элементов, 37
цвета, 248
Индикатор процесса, 239

К

Календарь, 224
Класс
добавление, 48; 249
переключение, 48; 249
удаление, 48; 249

Клонирование элементов, 42

Кодировка, 11

 UTF-8, 11

 однобайтовая, 11

M

Методы объектов JavaScript

 encodeURIComponent(), 121; 130;
 136; 152
 getElementById(), 9; 119
 getElementsByTagName(), 131
 isNaN(), 48
 item(), 131
 open(), 120
 parseInt(), 48
 preventDefault(), 77
 send(), 120
 setRequestHeader(), 120
 stopPropagation(), 75

O

Обертывание элементов, 40

Обработка конфликтных ситуаций, 14

Объединение массивов, 106

Объекты JavaScript

 document, 25; 37; 119
 event, 65; 69; 73; 75; 77
 window, 119

Отображение элемента, 79

Очистка содержимого, 42

П

Панель навигации, 204

Панель с вкладками, 208

Параметры тегов

 задать значение, 50
 значение параметра value, 52
 изменение значения, 50
 получение значения, 50
 нескольких параметров, 51
 удаление, 52

Перебор элементов, 25; 103

Перемещение элементов, 41; 161

Плавное изменение цвета, 248

Подключение библиотеки jQuery, 12

Поиск в массиве, 104

 элемента, 106

Получение

 данных с другого домена, 141; 142
 значений параметров тегов, 50

C

Свойства объектов JavaScript

 data, 131
 firstChild, 131
 innerHTML, 25; 37; 119
 length, 131
 onreadystatechange, 121
 parent, 119
 readyState, 121
 responseText, 121; 122; 152
 responseXML, 121; 131; 152
 status, 121
 top, 119
 value, 119

Селекторы

 #Идентификатор, 17
 *, 17
 .Класс, 17
 :animated, 23
 :button, 21
 :checkbox, 21
 :checked, 22
 :contains('Текст'), 23
 :disabled, 21
 :empty, 21
 :enabled, 21
 :eq(Индекс), 22
 :even, 22
 :file, 21
 :first, 23; 37
 :first-child, 20
 :gt(Индекс), 22
 :has(Селектор), 21
 :header, 23
 :hidden, 23
 :input, 21
 :last, 23
 :last-child, 21
 :lt(Индекс), 22
 :not(Селектор), 21

:nth(Индекс), 22
:nth-child(N), 20
:odd, 22
:only-child, 21
:parent, 23
:password, 21
:radio, 21
:reset, 21
:selected, 22; 54
:submit, 21
:text, 21
:visible, 23
[Параметр!=‘Значение’], 19
[Параметр\$=‘Значение’], 20
[Параметр*=‘Значение’], 20
[Параметр], 19
[Параметр^=‘Значение’], 19
[Параметр=‘Значение’], 19
группирование, 18
привязка к параметрам тегов, 19
 к элементам документа, 18
 к элементам формы, 21
псевдоклассы, 20
Тег, 17
Тег#Идентификатор, 17
Тег.Класс, 17
Элемент1 ~ Элемент2, 18
Элемент1 + Элемент2, 18
Элемент1 > Элемент2, 18
Элемент1 Элемент2, 18
Элемент1, Элемент2, 18
Скрытие элемента, 79
События, 57
 всплытие, 74
 выведение курсора, 61; 67
 выгрузка документа, 58
 выделение содержимого, 65
 вызов
 обработчика, 70
 собственных событий, 68
 двойной щелчок на элементе, 60
 действия по умолчанию, 76
 документа, 57
 изменение
 данных, 64
 размеров окна, 58
 клавиатуры, 63
мыши, 59
наведение курсора, 61; 67
нажатие
 клавиши, 63
 кнопки мыши, 60
обработка, 57
 один раз, 70
ошибок, 59
окончание загрузки страницы, 57
отправка формы, 64
отпускание
 клавиши, 63
 кнопки мыши, 60
перемещение мыши, 61
получение фокуса, 64
потеря фокуса, 64
прерывание, 75
 действий по умолчанию, 76
прокрутка содержимого, 58
удаление обработчика, 71; 73
формирование структуры документа,
 57
формы, 64
щелчок на элементе, 59
Сортировка элементов, 179

Y

Удаление
 класса, 48
 параметра тега, 52
 элемента, 42
Управление классами стилей, 48

Ф

Фильтрация элементов, 30
Форма
 взаимосвязанные списки, 94
 кнопки, 99
 обработка данных, 91
 переключатель, 97
 поле
 ввода пароля, 91
 для ввода многострочного
 текста, 93
 получение всех значений, 100

список**III**

- с возможностью множественного выбора, 53; 94
 - с возможными значениями, 94
 - текстовое поле, 91
 - флажок, 97
 - Форматы функции `$()`, 15
 - Функции PHP
 - `header()`, 127; 130
 - `isset()`, 127
 - `json_decode()`, 134
 - `json_encode()`, 134
 - Ширина элемента, 45
 - Шкала с бегунком, 242
-
- Эффекты, 248; 253

Э

jQuery

Новый стиль
программирования
на JavaScript

Н.А. Прохоренок

JQUERY — это JavaScript-библиотека, обеспечивающая кросбраузерную поддержку приложений и предоставляющая функциональные возможности, полезные для самого широкого круга задач. В этой книге описываются все основные возможности библиотек jQuery и jQuery UI, позволяющие реализовать на веб-страницах красочные графические эффекты и анимацию, перемещение и сортировку элементов, общение с сервером без перезагрузки страницы и многое другое.

Благодаря широкому охвату материала и большому количеству примеров, начинающим разработчикам книга будет полезна как самоучитель, а те, кто уже имеет опыт разработки Web-приложений, смогут использовать ее как справочник. Обязательные требования к читателю — базовые знания языка JavaScript и PHP. Версии описываемых программных продуктов: jQuery 1.3.2, jQuery UI 1.7.2.

ISBN 978-5-8459-1603-7



Издательский дом "Вильямс"
www.williamspublishing.com



10010



9 785845 916037