Расширение библиотеки jQuery

Кит Вуд

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

MANNING

УДК 004.738.5:004.45jQuery
ББК 32.973.202-018.2
В88

Кит Вуд
В88 Расширение библиотеки jQuery / пер. с анг. Киселева А.Н. – М.: ДМК Пресс, 2014. – 400 с.: ил.

ISBN 978-5-97060-071-9

jQuery – одна из наиболее популярных библиотек для разработки клиентских сценариев на JavaScript. В ней предусмотрено большое количество точек интеграции, посредством которых можно добавлять собственные селекторы и фильтры, расширения, анимационные эффекты и многое другое. Эта книга покажет вам, как это делается.

Из книги вы узнаете, как писать расширения и как проектировать их, чтобы максимально обеспечить возможность их многократного использования. Вы также научитесь писать новые виджеты и эффекты для jQuery UI. Наряду с этим вы исследуете особенности создания расширений для применения в таких ключевых аспектах библиотеки, как технология Ajax, события, анимация и проверка данных.

Издание предназначено для веб-программистов разной квалификации, уже использующих jQuery в своей работе.

УДК 004.738.5:004.45jQuery
ББК 32.973.202-018.2

ISBN 978-1-617291-03-6 (анг.)
ISBN 978-5-97060-071-9 (рус.)

Содержание

Глава 2

Первое расширение

Эта глава охватывает следующие темы:

- архитектура jQuery;
- создание простого расширения коллекций.

jQuery – это библиотека JavaScript, упрощающая взаимодействие с элементами веб-страниц. Обычно с ее помощью отыскиваются элементы либо прямым выбором, либо путем обхода дерева DOM, и затем к ним применяются некоторые операции. С помощью jQuery можно манипулировать элементами – создавать или удалять их, изменять значения атрибутов и свойств – и добавлять к ним обработчики событий, откликающиеся на действия пользователя. К элементам можно применять анимационные эффекты, изменяющие значения их свойств в течение некоторого интервала времени. Кроме того, jQuery поддерживает технологию Ajax, позволяя выполнять запросы к серверу и получать дополнительную информацию, не разрушая текущего содержимого страницы.

В предыдущей главе я упоминал, что jQuery может далеко не все, поэтому она предоставляет возможность расширения через множество точек интеграции, что привело к бурному росту количества расширений, создаваемых сообществом.

В этой главе сначала рассматривается архитектура jQuery, позволяющая расширениям действовать наравне со встроенным кодом, а затем будет представлено простое расширение коллекций (одно из тех, что воздействуют на множество выбранных элементов), чтобы продемонстрировать доступные возможности. В остальных главах подробно исследуются все точки интеграции, описывается порядок их использования для увеличения возможностей jQuery и формулируется свод правил разработки расширений.

2.1. Архитектура jQuery

Для удобства разработки исходный код jQuery разбит на несколько файлов, однако в процессе сборки они объединяются в единственный

файл, который затем минифицируется для последующей передачи в эксплуатацию либо оставляется в исходном виде для тестирования и отладки. В каждом файле сосредоточен код, реализующий определенный аспект функциональности jQuery, и некоторые из них имеют точки интеграции, дающие другим разработчикам возможность встраивать свои расширения.

Точка интеграции – это атрибут или функция внутри jQuery, где можно зарегистрировать новую функциональность или определенный тип (такой как коллекция функций или расширение Ajax), которые можно интерпретировать подобно стандартным особенностям. Когда в библиотеке возникает необходимость обратиться к расширению, она выполняет обратные вызовы в его код.

На рис. 2.1 изображена схема взаимосвязей между файлами, или модулями, входящими в состав jQuery.

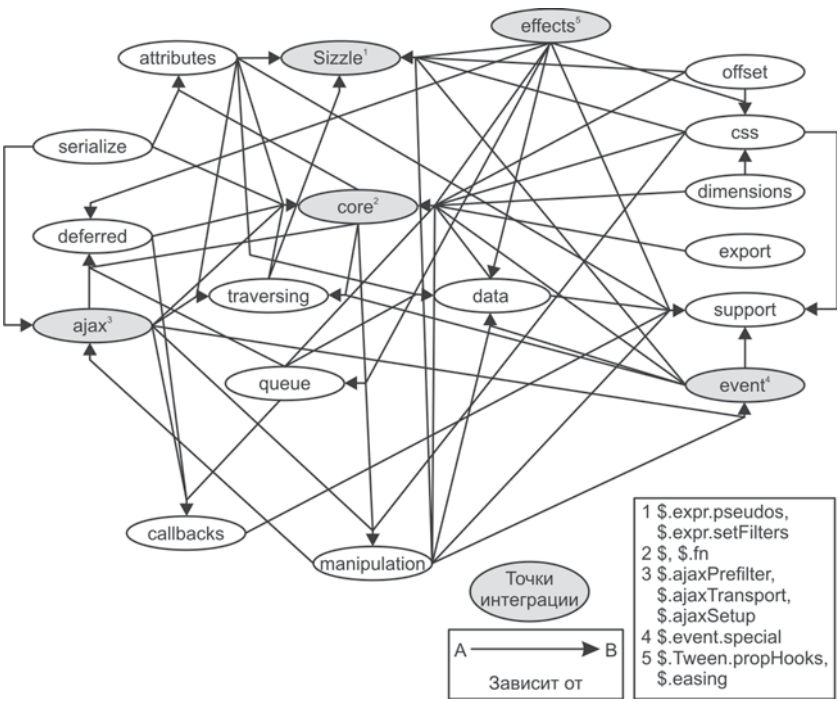

Рис. 2.1 ❖ Модули jQuery, взаимосвязи между ними и точки интеграции расширений

К числу основных модулей, поддерживающих возможность расширения jQuery (закрашены на рис. 2.1), относятся: библиотека *Sizzle*, реализующая функции выбора элементов из дерева DOM; модуль *core*, включающий саму функцию `jQuery`; модуль *ajax*, реализующий поддержку технологии Ajax; модуль *event*, реализующий механизм событий; и модуль *effects*, реализующий механизм анимации.

2.1.1. Точки интеграции с библиотекой jQuery

Все имеющиеся точки интеграции с библиотеками jQuery и jQuery UI перечислены в табл. 2.1 и описываются в следующих разделах. Напомню, что идентификатор `$` – это псевдоним функции `jQuery` (если он не был освобожден вызовом функции `noConflict`).

Таблица 2.1. Точки интеграции с библиотекой jQuery

Точка интеграции	Назначение	Примеры	Где обсуждается
`$`	Вспомогательные функции	`$.trim` `$.parseXML`	Глава 6
`$.ajaxPrefilter`	Предварительные фильтры Ajax	`$.ajaxPrefilter('script', ...)`	Глава 12
`$.ajaxSetup`	Преобразователи типов данных Ajax	`$.ajaxSetup({converters: {'text xml', $.parseXML}})`	Глава 12
`$.ajaxTransport`	Транспортные механизмы Ajax	`$.ajaxTransport('script', ...)`	Глава 12
`$.easing`	Функции управления переходами при воспроизведении анимации	`$.easing.swing` `$.easing.easeOutBounce`	Глава 10
`$.effects`	Визуальные эффекты jQuery UI (jQuery UI 1.8-)	`$.effects.clip` `$.effects.highlight`	Глава 10
`$.effects.effect`	Визуальные эффекты jQuery UI (jQuery UI 1.9+)	`$.effects.effect.clip` `$.effects.effect.highlight`	Глава 10
`$.event.special`	Нестандартные события	`$.event.special.mouseenter` `$.event.special.submit`	Глава 13

`$.expr.filters`
`$.expr[':']`
`$.expr.setFilters`
	Селекторы (jQuery 1.7-)	`$.expr.filters.hidden`
		`$.expr.setFilters.odd`
			Глава 3

`$.expr.pseudos`
`$.expr[':']`
`$.expr.setFilters`
	Селекторы (jQuery 1.8+)	`$.expr.pseudos.enabled`
		`$.expr.setFilters.first`
			Глава 3

`$.fn`	Расширения коллекций	`$.fn.show` `$.fn.append`	Глава 5
`$.fx.step`	Анимация атрибутов (jQuery 1.7-)	`$.fx.step.opacity`	Глава 11

Таблица 2.1 (окончание)

Точка интеграции	Назначение	Примеры	Где обсуждается
`$.Tween.propHooks`	Анимация атрибутов (jQuery 1.8+)	`$.Tween.propHooks.scrollTop`	Глава 11
`$.validator.addMethod`	Правила для расширения Validation	`$.validator.addMethod( 'USPhone', ..., ...)`	Глава 14
`$.widget`	Виджеты jQuery UI	`$.widget('ui.tabs', ...)`	Главы 8, 9

2.1.2. Селекторы

В состав библиотеки jQuery входит механизм Sizzle выбора элементов. Это самостоятельная библиотека, реализующая поддержку выбора и позволяющая находить требуемые элементы внутри веб-страницы. Всегда, когда это возможно, она делегирует выполнение операций функциям браузера, чтобы добиться максимальной скорости работы. Операции, которые не могут быть выполнены браузером, выполняются программным кодом на JavaScript. Например, чтобы найти все элементы `label`, следующие сразу за элементами `input` (например, метки флажков), в пределах элемента с атрибутом `id="preferences"`, можно использовать вызов:

```
$('#preferences input + label')...
```

Библиотека Sizzle дает возможность выбирать элементы по именам узлов, по значениям атрибутов `id` и `class` или по признаку вложенности. Можно также использовать селекторы псевдоклассов, включая определяемые спецификацией каскадных таблиц стилей (Cascading Style Sheets, CSS), и другие, добавляемые самой библиотекой Sizzle, такие как `:checked`, `:even` и `:not`. Комбинируя несколько селекторов в одной строке с условием выбора, можно отбирать только те элементы, которые действительно необходимы.

Селекторы псевдоклассов

Согласно спецификации CSS: «Псевдоклассы классифицируют элементы не по их именам, атрибутам или содержимому, а по другим характеристикам; по характеристикам, которые в принципе не могут быть получены из дерева документа»[1]. Эти селекторы начинаются с двоеточия (`:`) и включают определение условия выбора: по позиции (`:nth-child(n)`), по содержимому (`:empty`) и по отрицанию (`:not(selector)`).

---

[1] W3C, Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, «5.10 Pseudo-elements and pseudo-classes», http://www.w3.org/TR/2011/REC-CSS2-20110607/selector.html#pseudo-elements.

Вы можете добавлять собственные селекторы псевдоклассов и интегрировать их в процесс выбора с помощью точки интеграции `$.expr.pseudos` (или `$.expr.filters` в версиях jQuery ниже 1.8). В конечном счете селектор – это обычная функция, которая получает элемент в аргументе и возвращает `true`, если элемент соответствует селектору, и `false` в противном случае.

Расширения, добавленные через точку интеграции `$.expr.setFilters`, могут фильтровать элементы, опираясь на их позиции в текущем множестве совпавших элементов. Расширение должно быть реализовано в виде функции, возвращающей отфильтрованное множество элементов (в версии jQuery 1.8 и выше) или логический флаг, определяющий необходимость включения (в версии jQuery 1.7 и ниже).

Подробнее о добавлении собственных селекторов и фильтров в jQuery/Sizzle рассказывается в главе 3.

2.1.3. Расширения коллекций

Расширения коллекций выполняют операции с множествами элементов, полученными в результате процедуры выбора или последовательного обхода дерева DOM. К тому же они являются одними из распространенных расширений jQuery.

Для внедрения в библиотеку jQuery эти расширения должны подключаться к точке интеграции `$.fn` в виде функций, реализующих их возможности. Если заглянуть в реализацию библиотеки, можно заметить, что `$.fn` является простым псевдонимом для `$.prototype`. То есть любые функции, подключаемые к данной точке интеграции, будут доступны в любых объектах-коллекциях jQuery, полученных в результате вызова функции `jQuery` с селектором или элементами DOM. Они могут вызываться относительно этих коллекций в соответствующем контексте.

Все расширения коллекций должны возвращать текущий набор элементов или новый, если они реализуют какие-либо преобразования, чтобы их можно было включать в цепочки вызовов вместе с другими методами jQuery, – эта особенность является ключевой парадигмой выполнения операций в jQuery.

Основные принципы разработки расширений будут представлены в главе 4, а в главе 5 описывается реализующая эти принципы инфраструктура поддержки расширений, которую можно использовать для создания новых расширений коллекций.

2.1.4. Вспомогательные функции

Функции, не предназначенные для работы с коллекциями выбранных элементов (такие как встроенные функции `trim` и `parseXML`), могут внедряться в мир jQuery непосредственно через точку интеграции `$`. В действительности внедрять вспомогательные функции необязательно (их можно определить как обычные автономные функции), но они часто используют другие возможности jQuery, и подобным внедрением достигается единообразие использования библиотеки jQuery. Включение их в библиотеку также помогает уменьшить захламление глобального пространства имен, снизить вероятность конфликтов и позволяет хранить родственные функции в одном месте.

Вспомогательные функции не ограничиваются фиксированным списком параметров и могут принимать все, что потребуется для выполнения реализуемых ими операций.

Приемы добавления новых вспомогательных функций рассматриваются в главе 6.

2.1.5. Виджеты jQuery UI

Библиотека jQuery UI является официальной коллекцией компонентов пользовательского интерфейса, поведенческих особенностей и эффектов, реализованных на основе базовой библиотеки jQuery. Она предоставляет инфраструктуру поддержки виджетов, реализующую основные принципы создания новых компонентов или эффектов.

Виджеты создаются вызовом функции `$.widget` из библиотеки jQuery UI. Эта функция принимает имя нового виджета (включающее пространство имен, чтобы избежать конфликтов), необязательную ссылку на базовый «класс», наследуемый виджетом, и коллекцию функций, расширяющих и переопределяющих базовые возможности. Инфраструктура виджетов управляет их применением к выбранным элементам; устанавливает, извлекает и сохраняет параметры, определяющие внешний вид и поведение виджетов; а также удаляет виджеты, когда они становятся ненужными. Отображение функций, вызываемых пользователем, в методы, определяемые виджетом, выполняется внутренней функцией `$.widget.bridge`.

Подробнее об инфраструктуре поддержки и о виджетах jQuery UI рассказывается в главе 8. В главе 9 исследуются приемы использования модуля Mouse из библиотеки jQuery UI для реализации новых компонентов, использующих возможности взаимодействия с указателем мыши.

2.1.6. Эффекты jQuery UI

Еще одной важной составляющей библиотеки jQuery UI являются эффекты для анимации элементов веб-страниц. Большинство этих эффектов применяется для реализации постепенного сокрытия или появления элементов, как, например, `blind` и `drop`, но некоторые, такие как `highlight` и `shake`, служат для привлечения внимания пользователя к определенным элементам.

Внедрять новые эффекты в библиотеку jQuery UI можно с помощью точки интеграции `$.effects.effect` (или `$.effects`, в версиях jQuery UI 1.8 и ниже). После внедрения эти эффекты будут доступны (по именам) для использования с функциями `effect`, `show`, `hide` или `toggle` из библиотеки jQuery UI. Каждый эффект является функцией, добавляющей функцию обратного вызова в очередь `fx` текущего элемента и реализующей анимационный эффект, который будет воспроизведен после обработки предыдущего эффекта, включенного в очередь ранее.

Функции управления переходами (easings) определяют, как будет изменяться значение атрибута с течением времени, и могут применяться к анимационным эффектам для управления ускорением и замедлением изменений. Несмотря на то что функции переходов являются частью jQuery, библиотекой предлагаются только две такие функции – `linear` и `swing`. Библиотека jQuery UI добавляет еще 30 функций переходов. Добавление собственных функций переходов осуществляется с помощью точки интеграции `$.easing`. Эти функции должны возвращать величину изменения значения атрибута (в нормализованном виде – от 0 до 1) с учетом прошедшего времени с начала воспроизведения анимационного эффекта (также нормализованного на интервале от 0 до 1).

Подробнее о функциях переходов и эффектах jQuery UI рассказывается в главе 10.

2.1.7. Анимация свойств

Поддержка анимации в jQuery дает возможность изменять значения различных атрибутов выбранных элементов. Обычно эти атрибуты определяют внешний вид, в результате их изменения возникают эффекты перемещения элементов, изменение их размеров или рамок, а также изменение размеров шрифтов, которыми отображается содержимое элементов. Но библиотека jQuery поддерживает возможность изменения только простых числовых значений атрибутов (включая определение единиц измерения). Для анимации атрибутов

со сложными значениями необходимо добавлять собственные функции анимации.

Подключение новых механизмов анимации атрибутов со сложными значениями осуществляется через точку интеграции `$.Tween.propHooks` путем передачи двух функций – функции чтения и функции записи значения атрибута. В версиях jQuery 1.7 и ниже используется точка интеграции `$.fx.step`, которой должна передаваться одна функция, реализующая один шаг анимации.

Подробнее о добавлении новых анимационных эффектов рассказывается в главе 11.

2.1.8. Поддержка Ajax

Поддержка технологии Ajax является одной из важнейших особенностей библиотеки jQuery. Она упрощает получение информации со стороны сервера и ее встраивание в текущую страницу без необходимости полностью обновлять ее. Поскольку серверы могут поставлять данные в разных форматах, jQuery определяет множество *предварительных фильтров*, управляющих процессом извлечения данных; *транспортов*, обеспечивающих собственно извлечение данных; и *преобразователей*, выполняющих преобразование полученных данных в требуемый формат. Каждый из этих механизмов может расширяться под определенные требования. На рис. 2.2 показано, как выглядит стандартная процедура выполнения запроса Ajax.

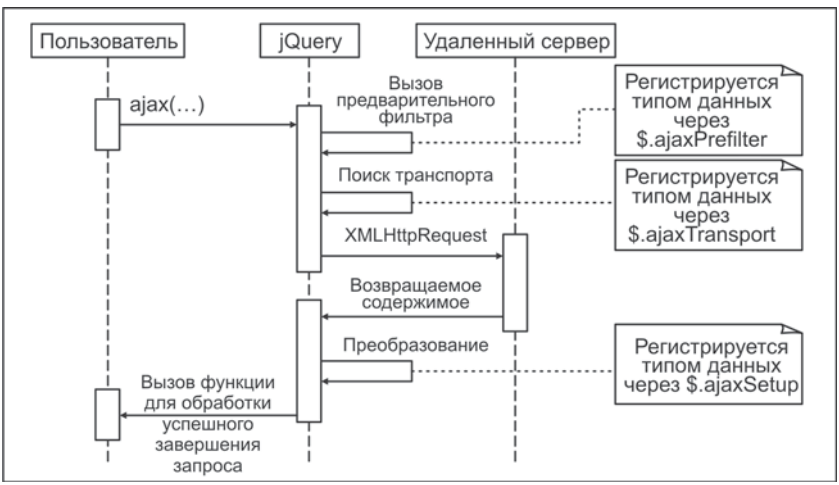

Рис. 2.2 ❖ Диаграмма стандартной процедуры выполнения запроса Ajax с точками интеграции

Вызовом функции `$.ajaxPrefilter` можно зарегистрировать собственную функцию, которая будет вызываться при попытке выполнить запрос данных в указанном формате, таком как `html`, `xml` или `script`. Этой функции передается ссылка на объект `XMLHttpRequest`, который будет использоваться для выполнения запроса, благодаря чему вы сможете настроить параметры запроса и даже отменить его.

Вызовом функции `$.ajaxTransport` можно зарегистрировать функцию, реализующую фактическое извлечение данных указанного формата. Эта возможность позволяет настроить доступ к данным (например, загрузить изображение непосредственно в элемент `Image`). По умолчанию используется объект `XMLHttpRequest`.

Обычно данные возвращаются сервером в текстовом формате, но иногда бывает желательно преобразовать их в некоторый другой формат, например превратить разметку XML в дерево DOM. Вызовом функции `$.ajaxSetup` можно зарегистрировать функцию, преобразующую исходные данные в другой формат, которые затем будут возвращены как результат выполнения запроса Ajax.

Подробнее об определении новых предварительных фильтров, транспортов, протоколов и преобразователей рассказывается в главе 12.

2.1.9. Обработка событий

Библиотека jQuery позволяет подключать к выбранным элементам обработчики событий, которые будут вызываться в ответ на действия пользователя. Эта возможность часто используется для обработки событий от мыши или клавиатуры, а также событий изменения состояния. При необходимости можно определить собственные события для обработки нестандартных ситуаций.

Регистрация новых событий выполняется вызовом функции `$.event.special`. При этом в вызов функции передается тип события, а также функции настройки событий, удаления поддержки событий, когда она станет не нужна, и возбуждения событий при наступлении соответствующих условий.

Подробнее о создании и обработке собственных событий рассказывается в главе 13.

2.1.10. Правила проверки данных

Несмотря на то что расширение Validation не является частью ядра библиотеки jQuery, оно используется достаточно часто и предоставляет свою точку интеграции, посредством которой можно регистри-

ровать собственные правила проверки данных. Эти правила можно использовать наряду со встроенными, такими как `required` и `number`, и проверять с их помощью наличие и правильность данных в полях форм перед отправкой на сервер.

Регистрация собственных правил осуществляется вызовом функции `$.validator.addMethod`, которой передается имя правила; функция, возвращающая `true`, если элемент содержит допустимое значение, и `false` – в противном случае; и текст сообщения об ошибке. Для автоматического включения нового правила через атрибут `class` отдельного элемента можно воспользоваться функцией `$.validator.addClassRules`.

Подробнее об определении собственных правил проверки рассказывается в главе 14.

2.2. Простое расширение

Расширения для библиотеки jQuery могут делать практически все, что угодно, о чем свидетельствует широкий выбор имеющихся сторонних расширений. Это могут быть простые расширения, воздействующие на отдельные элементы или сложные, изменяющие внешний вид и поведение множества элементов, как, например, расширение Validation.

Чаще других создаются расширения коллекций, добавляющие новые функциональные возможности к коллекциям элементов, полученным в результате выполнения процедуры выбора или последовательного обхода дерева DOM. Простым примером расширений этого типа может служить расширение Watermark, реализующее отображение подсказки в поле ввода. Исследование процедуры создания такого расширения поможет вам получить более полное представление о том, как создаются расширения.

2.2.1. Текст подсказки

Чтобы сэкономить место внутри формы, метки полей ввода иногда опускаются и замещаются текстом подсказки (внутри самого поля ввода), который исчезает, когда поле получает фокус ввода. Если поле останется незаполненным, в нем вновь выводится текст подсказки. Чтобы не вводить пользователя в заблуждение и показать, что этот текст не является фактическим значением поля ввода, он обычно выводится серым цветом. Такие подсказки часто называют *водяными знаками* (watermark).

Текст подсказки можно было бы определять на этапе инициализации расширения, но лучше определять его отдельно для каждого поля и тем самым обеспечить отдельные настройки для каждого поля в отдельности. Для хранения текста подсказки идеально подходит атрибут `title`. Этот атрибут специально предназначен для хранения короткого описания поля, и его содержимое может отображаться браузером в виде всплывающей подсказки при наведении указателя мыши на поле ввода. Для слабовидящих это расширение может служить дополнительной подсказкой о том, в каком поле находится фокус ввода.

Когда поле ввода получает фокус ввода, необходимо удалить текст подсказки, если он присутствует, и изменить стиль отображения поля, чтобы оно выглядело как активное. Аналогично, когда поле теряет фокус ввода, необходимо восстановить в нем текст подсказки и установить неактивный стиль отображения, если поле осталось пустым. На рис. 2.3 изображено действие расширения Watermark в разные моменты ввода данных.

Рис. 2.3 ❖ Действие расширения Watermark: перед взаимодействием, после ввода имени и в готовности к отправке

Для большей гибкости визуальное оформление полей при отображении текста подсказки должно контролироваться с помощью CSS. При отображении подсказки можно присваивать полю некоторый класс CSS и удалять его при получении полем фокуса ввода или когда в нем присутствует текст, введенный пользователем. В этом случае фактическое визуальное оформление перекладывается на стили CSS и легко может быть переопределено пользователем.

2.2.2. Реализация расширения Watermark

Так как расширение применяется к элементам страницы, оно является расширением коллекций, то есть воздействует на коллекцию элементов, найденных в результате выбора по селектору или последо-

вательным обходом дерева DOM. Отсюда следует, что подключаться это расширение должно к точке интеграции `$.fn`.

В листинге 2.1 приводится полная реализация расширения Watermark.

Листинг 2.1. Расширение Watermark

```
// ❶ Объявление функции расширения
$.fn.watermark = function(options) {
    // ❷ Параметры настройки
    options = $.extend({watermarkClass: 'watermark'},
                       options || {});
    // ❸ При получении фокуса...
    return this.focus(function() {
        var field = $(this);
        // ❹ ...если содержит подсказку...
        if (field.val() == field.attr('title')) {
            // ❺ ...удалить подсказку
            field.val('').
                removeClass(options.watermarkClass);
        }
    // ❻ При потере фокуса ввода
    }).blur(function() {
        var field = $(this);
        // ❼ ...если пустое...
        if (field.val() == '') {
            // ❽ ...восстановить подсказку
            field.val(field.attr('title')).
                addClass(options.watermarkClass);
        }
    }).blur(); // ❾ Инициализировать поле
};
```

Расширение подключается к точке интеграции `$.fn` объявлением функции `$.fn.watermark` ❶, что обеспечивает возможность его внедрения в обработку выбираемых множеств элементов. Новый атрибут получает имя особенности, которую он реализует, и становится доступен по этому имени. Значением атрибута является функция, принимающая единственный аргумент (`options`) и добавляющая новые особенности к целевым полям. В аргументе функция ожидает получить единственный параметр настройки: имя класса CSS, используемого для оформления внешнего вида поля, чтобы показать, что оно содержит текст подсказки. Для этого параметра предусмотрено значение по умолчанию, которое может быть переопределено пользователем ❷. Значение по умолчанию определяется как объект с атрибутом `watermarkClass` и значением `'watermark'`, который расширяется другими параметрами настройки, возможно, переопределяющими значе-

ния по умолчанию. Конструкция || {} гарантирует замену аргумента options пустым объектом, если он не был передан пользователем.

Чтобы обеспечить возможность включения этого расширения в цепочку вызовов других расширений, что является одним из основных элементов философии jQuery, функция должна возвращать множество элементов, над которыми выполняется операция ❸. Так как большинство встроенных функций jQuery также возвращают тот же самый набор, можно просто вернуть результат вызова стандартной функции jQuery.

К каждому полю в выбранном множестве добавляется обработчик события focus ❸. Внутри этого обработчика сохраняется ссылка на текущее поле, представленное объектом jQuery, потому что она еще будет использоваться неоднократно. Затем текущее содержимое поля сравнивается со значением его атрибута title ❹. Если они равны, текст подсказки замещается пустой строкой ❺, и из атрибута class удаляется класс, указанный в параметрах настройки.

Так как работа продолжается с тем же набором полей и со всеми ними выполняются одни и те же операции, мы можем добавить в цепочку дополнительный обработчик blur ❻. И снова обработчик сохраняет ссылку на текущее поле для последующего использования. На этот раз он сравнивает текущее значение поля с пустой строкой ❼ и в случае выполнения условия восстанавливает текст подсказки (из атрибута title) и добавляет класс CSS ❽.

В заключение необходимо вызвать только что определенный обработчик blur ❾, чтобы инициализировать поле в зависимости от его текущего значения.

2.2.3. Удаление текста подсказок

Так как строки подсказок устанавливаются в качестве значений всех полей, они могут быть отправлены на сервер и обработаны им, если не предусмотреть некоторые дополнительные операции. Подсказки – это артефакты пользовательского интерфейса, поэтому перед отправкой формы их следует удалить.

Самый простой способ реализовать удаление подсказок – определить еще одно расширение коллекций, выполняющее эту задачу. В листинге 2.2 представлена эта дополнительная функция.

Листинг 2.2. Удаление подсказок из значений полей

```
// ❶ Объявление функции расширения
$.fn.clearWatermark = function() {
```

```
    // ❷ Для каждого поля...
    return this.each(function() {
        // ❸ ...если содержит подсказку...
        var field = $(this);
        if (field.val() == field.attr('title')) {
            // ❹ ...удалить ее
            field.val('');
        }
    });
};
```

И снова расширение подключается к точке интеграции `$.fn` в виде функции `$.fn.clearWatermark`, потому что данное расширение так же выполняет операции с множеством элементов страницы ❶. Оно так же обеспечивает возможность включения в цепочки вызовов, возвращая ссылку на текущее множество выбранных элементов обращением к функции `each` ❷. Для каждого выбранного элемента сравнивается его текущее значение с атрибутом `title` ❸, и если они равны, в значение поля записывается пустая строка ❹.

Эта функция должна вызываться непосредственно перед отправкой формы на сервер, в противном случае текст подсказок будет участвовать в дальнейшей обработке. При необходимости вы легко сможете восстановить подсказки во всех полях, вызвав обработчик `blur`.

2.2.4. Применение расширения Watermark

Чтобы задействовать расширение Watermark, поместите предыдущий код непосредственно в веб-страницу или сохраните его в отдельном файле JavaScript (jquery.watermark.js) и подключите этот файл к странице. Аналогично для оформления внешнего вида расширения можно добавить соответствующие стили непосредственно в страницу или подключить их, если они хранятся в отдельном файле CSS (jquery.watermark.css). Если сохранить расширение и стили в отдельных файлах, это упростит повторное их использование в других страницах.

На рис. 2.4 показано, как выглядит страница, демонстрирующая применение расширения Watermark, а в листинге 2.4 приводится ее код разметки.

Рис. 2.4 ❖ Страница, демонстрирующая применение расширения Watermark