Одна из проблем с формами в HTML 5 состоит в том, что поля формы — это всего лишь пустые ячейки для заполнения, не более того. На сервере, конечно же, требуется валидация данных, но для создания осмысленного интерфейса приходится дублировать эту проверку в браузере с помощью JavaScript. Учитывая, что формы используются практически на каждой веб-странице — формы поиска, регистрации, добавления комментария и т. д., — было бы здорово, если бы в браузеры была включена валидация тех типов данных, которые чаще всего вводятся пользователем.

Как вы уже догадались, формы HTML5 позволяют делать именно это.

Мы ❤ HTML, и теперь он тоже нас ❤

HLML5 позволяет создавать формы гораздо быстрее. Кроме того, в HTML5 появился ряд новых приятных возможностей — например, к уже существующим типам HTTP-запросов, которые могут отправлять формы (`get` и `post`), было добавлено два новых: `update` и `delete`. Однако самое потрясающее обновление, которое оценят не только разработчики, но и их боссы и клиенты, заключается в том, что теперь жизнь каждого из них значительно упростилась — новые типы ввода данных поддерживают особые пользовательские интерфейсы и, самое главное, обладают встроенными возможностями сообщений об ошибках.

Со временем проверка для основных типов данных с помощью JavaScript станет ненужной, однако сейчас не стоит посыпать ее нафталином — JavaScript будет необходим до наступления светлого будущего, когда у каждого будет HTML5-браузер (или ваш начальник решит, что пользователям древних браузеров придется ограничиться проверкой форм только со стороны сервера). В главе 12 мы познакомим вас с методологией под названием «заполнение» (polyfill), которая гарантирует, что старые браузеры (и только старые браузеры) получат помощь от JavaScript, в то время как вы будете создавать современный код, согласующийся со стандартом.

ПРИМЕЧАНИЕ

Эти усовершенствования форм пока что реализованы далеко не везде. Наиболее широко их поддержка внедрена в Opera, на втором месте браузеры WebKit и Firefox. На момент написания этой главы кое-что поддерживается даже в Internet Explorer 10 Platform Preview 2. Золотые времена настают!

Новые типы ввода данных

Новые поля форм были основой спецификации, которая превратилась в HTML5, и именно они представляют собой воплощение идеи обратно совместимого расширения. Расширения в данном случае — это в основном новые значения атрибута `type` элемента `input`. В HTML 5 говорится, что браузеры по умолчанию должны использовать `<input type=text>`, если вы не указали другое или неизвестное значение атрибута `type`. Таким образом, устаревшие версии, не распознающие новые расширения, обратятся к значению по умолчанию и предложат пользователю обычное текстовое поле ввода. Эту ситуацию можно распознать

с помощью сценария и по необходимости заполнение, для того чтобы старые браузеры имитировали новое поведение.

Спецификация не накладывает никаких ограничений на то, как браузеры должны оформлять новые типы ввода данных, сообщать об ошибках и т. д. Различные браузеры и устройства будут использовать различные пользовательские интерфейсы; сравните, к примеру, как выглядит поле выбора в Safari для ПК и iPhone (рис. 3.1).

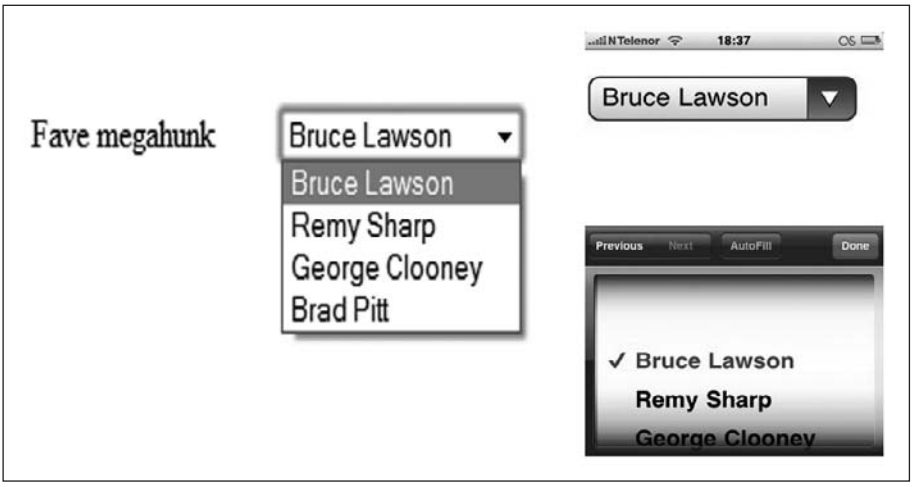

Рис. 3.1. Одно и то же поле выбора в Safari/Windows (*слева*) и Safari/iPhone

Точно так же не существует единых правил относительно того, как браузер будет сообщать об ошибках. Рисунок 3.2 иллюстрирует ошибку «необходимо заполнить текстовое поле перед отправкой данных» в Opera, Firefox и Google Chrome. Кроме того, вы видите ту же ошибку в локализованной японской версии Opera. Так как сообщения являются частью браузера, они автоматически локализуются, что означает меньше работы для разработчика и лучшее впечатление от работы с браузером у пользователя.

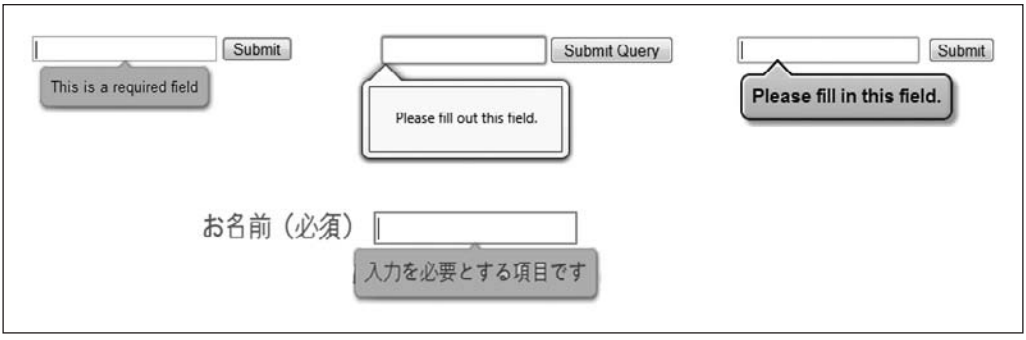

Рис. 3.2. Автоматически сгенерированные сообщения об ошибке в Opera, Firefox, Chrome и Japanese Opera (*внизу*)

ТИП ВВОДА ДАННЫХ EMAIL

Строка `<input type=email>` сообщает браузеру, что форма не должна быть отправлена, пока пользователь не ввел нечто, похожее на правильный адрес электронной почты, то есть браузер не проверяет, существует ли такой адрес, а только определяет правильность его формата. Пользователь может отправить форму с незаполненным полем, только если к этому полю не добавлен атрибут `required` (это верно для всех типов ввода данных).

Атрибут `multiple` означает, что в качестве значения поля может быть указан список из допустимых электронных адресов, разделенных запятыми. Это, конечно, не значит, что пользователь должен вводить эти адреса вручную; браузер может открывать специальное окно со списком контактов пользователя из почтового клиента (из памяти телефона), где нужные адреса можно отметить флажками, а затем самостоятельно незаметно от пользователя конструировать список.

Современные браузеры пока что не такие услужливые, но поскольку данный тип однозначно определен и легко считывается и распознается компьютером, браузеры теперь понимают, чего добивается разработчик, и могут, вероятно, предоставить более тесно связанный с контекстом пользовательский интерфейс. Например, экспериментальное дополнение Firefox Contacts (http://mozillalabs.com/blog/2010/03/contacts-in-the-browser) формирует список контактов из различных источников, чтобы при заполнении поля `<input type=email>` пользователь мог выбрать в нем нужный адрес. Он также делает эту информацию доступной для скриптов сайта с помощью рабочей версии W3C Contacts API (http://www.w3.org/2009/dap/contacts/).

ТИП ВВОДА ДАННЫХ URL

Строка `<input type=url>` заставляет браузер проверять, что пользователь ввел правильный URL-адрес. Браузер может помочь пользователю это сделать — Opera, например, автоматически добавляет http:// в начало URL в случаях, когда пользователь не указывает протокол (то есть не добавляет в начале http:// или ftp:// или какое-то еще общепринятое обозначение протокола). (URL не обязательно должен относиться к веб-узлу; страница может быть, например, сетевым HTML-редактором, в котором пользователям иногда требуется псевдопротокол `tel:`.

ТИП ВВОДА ДАННЫХ DATE

Тип `date` — один из моих любимых. Все мы видели веб-страницы, в которых пользователю необходимо ввести дату полета, концерта и т. д. Поскольку не очень понятно, как это делать (в формате DDMM-YYYY, или MM-DD-YYYY, или DD-MMM-YY?), разработчики с помощью JavaScript создают виджеты для выбора даты, которые от сайта к сайту значительно различаются внешним видом, удобством использования и доступностью.

`<input type=date>` решает эту проблему, открывая доступ к встроенному в браузер виджету выбора даты. Opera, к примеру, открывает виджет с календарем (рис. 3.3).

На смартфонах BlackBerry в BlackBerry Device Software версии 5.0 в качестве элемента управления для ввода даты использовался тот же Java-компонент, что и в приложении «Календарь» BlackBerry (хотя он не встроен в приложение «Календарь») (рис. 3.4).

Рис. 3.3. Opera 10.5 открывает виджет с календарем

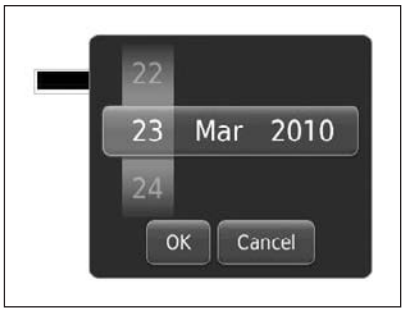

Рис. 3.4. Так визуализируется `<input type=date>` в браузере BlackBerry

Безусловно, это только начало. Вполне возможно, что в будущем браузер сможет не только выводить блестящий новенький виджет выбора даты, но и делать что-то намного более интеллектуальное, например вызывать встроенное приложение «Календарь», чтобы вы могли выбрать дату исходя из вашего расписания. Главное — теперь браузер может понимать, что вы хотите ввести. Раньше виджеты выбора даты были с точки зрения браузера не более чем элементами `<div>`, `<span>` и ссылками с большим количеством JavaScript-кода, отвечающего за поведение. Теперь браузер знает, что вы собираетесь ввести дату и время, и может предложить вам усовершенствованные элементы управления и обеспечить интеграцию с другими сведениями о дате/времени.

ТИП ВВОДА ДАННЫХ TIME

`<input type=time>` позволяет вводить время по 24-часовой шкале и проверяет правильность введенных данных. И снова выбор пользовательского интерфейса остается за браузером — это может быть просто ввод чисел с выводом ошибки в случае, если количество часов превышает 24 или количество минут превышает 59, или же он может быть разработан более детально: например, в виде циферблата, стрелки которого можно перетаскивать курсором мыши. Интерфейс может также учитывать смещение часового пояса.

ТИП ВВОДА ДАННЫХ DATETIME

Типы `date` и `time`, с которыми мы только что познакомились, можно объединять с помощью `<input type=datetime>` для проверки точной даты и времени. Поле ввода местного времени и даты работает так же, как и `datetime`, за исключением того, что пользователь не может добавить (или изменить) смещение часового пояса.

ТИП ВВОДА ДАННЫХ MONTH

`<input type=month>` позволяет вводить месяц и выполняет проверку введенных данных. Хотя внутри эта информация хранится в виде числа от 1 до 12, браузер может предложить пользователю выбрать месяц по названию. Вы могли бы, например, использовать поле выбора из 12 вариантов (январь–декабрь), но такой вариант не универсален с точки зрения выбора языка. Если вы будете использовать тип ввода данных HTML5 `month`, французская

локализация браузера может, к примеру, в раскрывающемся списке заменить «Январь» на «Janvier». Такие тонкости — в большей степени забота браузера и *в меньшей степени — ваша*. И это абсолютно нормально.

ТИП ВВОДА ДАННЫХ WEEK

`<input type=week>` позволяет вводить номер недели и выполняет проверку введенных данных. Хотя это может выглядеть как простое поле, позволяющее пользователю вводить число, на практике все оказывается сложнее: в некоторых годах 53 недели. Поэтому для седьмой недели 2010 года используется формат 2010-W07.

Opera позволяет открыть пользовательский интерфейс выбора даты и помещает в поле ввода номер недели, соответствующий любой выбранной дате, а не дате в формате YYYY-MM-DD (рис. 3.5).

Рис. 3.5. `<input type=week>` в Opera

ТИП ВВОДА ДАННЫХ NUMBER

Как это ни удивительно, но тип ввода данных `number` выдает ошибку, если пользователь не ввел численные символы. Он не предназначен для ввода номеров телефона, так как их часто указывают с пробелами, скобками, значком плюса, дефисами и т. п.; используйте для телефонов `<input type=tel>`.

Он прекрасно работает с атрибутами `min`, `max` и `step`. В Opera и Chrome он выглядит как счетчик, значения которого можно ограничить сверху и снизу, а шаг задается с помощью атрибута `step`; при этом пользователь может ввести число с клавиатуры (рис. 3.6). Стрелки счетчика находятся за пределами поля ввода в Opera, но внутри этого поля в Chrome. В спецификации ничего не говорится относительно того, как должен выглядеть пользовательский интерфейс этих новых элементов управления.

Рис. 3.6. Способ визуализации `<input type=number>` в Opera (*слева*) и Chrome (*справа*)

В настольных браузерах, не предлагающих для числовых данных особый пользовательский интерфейс, возникает неприятная ситуация. В настоящее время ввод буквенных симво-

лов в поля `type=number` в Opera и Chrome не приводит к ошибке валидации. По-моему, такое поведение не согласуется с логикой. Подобное случается из-за того, что браузер не занимается проверкой введенных данных — он даже не заменяет текущее значение поля введенными пользователем нецифровыми символами. К сожалению, пользовательский интерфейс предполагает, что вы ввели буквы и они были одобрены.

ТИП ВВОДА ДАННЫХ RANGE

`<input type=range>` отображается в виде ползунка. На рис. 3.7 показано, как это выглядит в Google Chrome.

Рис. 3.7. Визуализация `<input type=range>` в Chrome

Раньше ползунки приходилось имитировать и для этого требовалось перехватывать входные данные и использовать JavaScript и изображения для стрелок. Так как они не были встроены в браузеры, обеспечение удобного управления с клавиатуры требовало особой аккуратности и дополнительного кода. Теперь, когда ползунки встроены в HTML, с разработчика снимается ответственность, и за счет этого уменьшается объем кода и повышается доступность для пользователей клавиатуры.

Чтобы подробнее узнать об этом, посмотрите пример в разделе «Как собрать все это вместе» далее в этой главе. Данный тип прекрасно работает с атрибутами `min`, `max` и `step` (см. далее).

ТИП ВВОДА ДАННЫХ SEARCH

Этому типу ввода данных требуется поисковый термин. В Safari есть уникальный атрибут, не предусмотренный в спецификации, который добавляет историю недавних запросов (с помощью `results=n`). Разница между типами ввода данных `search` и `text` чисто стилистическая; в Safari на Mac `search` оформляется тем стилем, который используется по умолчанию в операционной системе (с закругленными углами), что тем не менее можно изменить с помощью специального CSS-кода (спасибо Уилфреду Нэсу).

```
input[type="search"] {-webkit-appearance: textfield;}
```

ТИП ВВОДА ДАННЫХ TEL

Тип `tel` используется для ввода телефонного номера. Особой проверки не производится; более того, не требуется вводить только цифры — многие номера часто пишутся с использованием других знаков, например +44 (0) 208 123 1234.

Поскольку мобильные телефоны «знают» свой собственный номер, мы надеемся, что в большинстве мобильных телефонов появится функция автозаполнения таких полей. Пока это нигде не реализовано, но iPhone умеет открывать экран ввода телефонного номера (рис. 3.8).

Рис. 3.8. Клавиатура iPhone для заполнения полей типа `<input type=tel>`

ТИП ВВОДА ДАННЫХ COLOR

`<input type=color>` позволяет пользователю ввести значение цвета, выбрав нужный вариант в палитре. Пока это реализовано только в BlackBerry (рис. 3.9) и Opera.

Рис. 3.9. Визуализация `<input type=color>` на устройстве BlackBerry

НЕ ЗАБЫВАЙТЕ ОБ АТРИБУТЕ NAME!

Конечно, теперь валидация на клиентской стороне встроена в браузер, но это не означает, что можно окончательно расслабиться. Не забывайте присваивать полям ввода (и группам переключателей) уникальные значения в атрибуте name, так как именно они позволяют получать введенные данные, переданные формой, на серверной стороне. Старые версии Opera требуют этого еще до выполнения валидации HTML5, и спецификация говорит ровно о том же.

Так как старые версии IE иногда путают id и name при использовании getElementByID, мы рекомендуем задавать одно и то же уникальное значение для атрибутов id и name каждого поля. Это делает формы более доступными:

```
<label for=f-email>Email address</label>
<input id=f-email name=f-email type=email>
```

Новые атрибуты

Помимо новых типов ввода данных, в HTML5 определен ряд новых атрибутов для элемента `<input>`, отвечающих за его поведение и ограничения: `autocomplete`, `min`, `max`, `multiple`, `pattern` и `step`. Есть еще один атрибут — `list`; он присоединяется к новому элементу, позволяя использовать новый метод ввода данных.

Атрибут list и <datalist>

Сочетание элемента `<input>` с атрибутом `list` и элементом `<datalist>` дает комбинированное поле — комбинацию раскрывающегося списка и текстового поля из одной строки. Оно позволяет пользователям вводить свой собственный текст, если их не устраивают предложенные в списке варианты.

Список создается внутри нового элемента `<datalist>`, `id` которого указывается в качестве значения атрибута `list`.

```
<input id=form-person-title type=text list=mylist>
    <datalist id=mylist>
        <option label=Mr value=Mr>
        <option label=Ms value=Ms>
        <option label=Prof value="Mad Professor">
    </datalist>
```

Сам `<datalist>` не отображается, но появляется в виде значений поля, похожего на поле выбора.

В предыдущем примере `type=text` делает возможным ввод произвольного текста; но `<datalist>` можно использовать и с другими типами ввода данных, например `url` и `email`.

Многие спрашивают, почему пара `<input>`/`<datalist>` не объединена в один элемент, как, например, `<select>`. На самом деле это связано с обратной совместимостью: в устаревших браузерах пара `<input>`/`<datalist>` превращается в `<input type=text>`, благодаря чему пользователь может хотя бы что-то ввести, а вы уже можете дополнить реализацию с помощью JavaScript.

Джереми Кейт приводит отличный пример подобной обратной совместимости в статье по адресу http://adactio.com/journal/4272/ (воспроизвожу ее с его разрешения):

```
<label for="source">How did you hear about us?</label>
<datalist id="sources">
    <select name="source">
        <option>please choose...</option>
        <option value="Television">Television</option>
        <option value="Radio">Radio</option>
        <option value="Newspaper">Newspaper</option>
        <option>Other</option>
    </select>
    If other, please specify:
</datalist>
<input id="source" name="source" list="sources">
```

Обратите внимание, что мы обрамили элементы `<option>` дополнительным тегом `<select>`, благодаря чему визуализация содержимого `datalist` имитирует разметку традиционных элементов выбора `dropdown` (раскрывающийся список). Браузеры, понимающие элемент `<datalist>`, проигнорируют все лишнее, сосредоточившись на элементах `<option>` — для них вложенный `<select>` попросту невидим. Текст «If other, please specify» также игнорируется. С другой стороны, не поддерживающие данную функциональность браузеры не увидят элемент `<datalist>` и покажут то, что они воспринимают как стандартный элемент `<select>`. Также они отобразят текст «If other…» и поле ввода, с которым связан `datalist`.

Другими словами, браузеры, поддерживающие `<datalist>`, считают каждый `<option>` частью списка данных (datalist) и кроме них ничего не видят. Браузеры, не поддерживающие `<datalist>`, считают все `<option>` частью `<select>` и видят дополнительный текст «If other, please specify»; элемент ввода, связанный со списком данных через атрибут `list`, превращается в простое текстовое поле ввода (рис. 3.10).

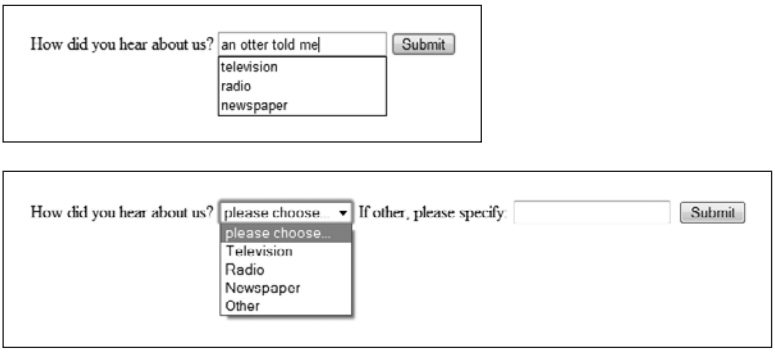

Рис. 3.10. Визуализация `<datalist>` в Opera (*вверху*) и упрощенный вариант в Safari (*внизу*)

Это отличный шаблон, который наверняка станет частью вашего ежедневного арсенала инструментов для кодирования форм — если только Рабочая группа не примет решение отобрать его у нас (см. примечание)!

ПРИМЕЧАНИЕ

Рабочая группа (Working Group) рассматривает вариант устранения данного метода плавного ухудшения характеристик через разметку, приводя не слишком убедительные обоснования — мол, веб-разработчики не так часто его применяют, к тому же его трудно описать в спецификации и реализовать (что делать с элементами `<script>` внутри `<datalist>`?).

Если такое произойдет, мы обязательно упомянем об этом на http://www.introducinghtml.com, а вам придется полагаться на помощь сценариев, добавляющих поддержку `<datalist>` в старые браузеры (через плавное ухудшение характеристик). Это будет ужасно.

Как и я сам, это не самое симпатичное в мире создание, однако оно работает (правда, в IE10 Platform Preview 2 вы в результате получаете комбинацию из `select` + `input`) и хорошо демонстрирует плавный перенос новых возможностей на ограниченные старые платформы.

АТРИБУТ AUTOFOCUS

Булев атрибут `autofocus` — способ задания установки фокуса на определенном элементе управления формы во время загрузки страницы. Раньше разработчику требовалось писать для этого JavaScript-код, используя `control.focus()`. Новый способ позволяет делать достаточно разумные вещи, например не устанавливать фокус на элементе, если пользователь в это время вводит данные в другом месте (проблема, типичная для старых JavaScript-скриптов, реализующих эту возможность).

На странице должно быть только одно такое поле ввода. Помните об удобстве использования и применяйте этот атрибут с осторожностью. Мы рекомендуем добавлять его только там, где поле формы служит для выполнения основной функции страницы — как, например, форма поиска.

АТРИБУТ PLACEHOLDER

Достаточно популярным у разработчиков является такой прием: изначально в поле ввода добавляется текст-подсказка, которая исчезает при установке фокуса на этом поле, а затем (при установке фокуса на другом элементе) снова появляется. Раньше для этого требовался JavaScript. Но теперь такое поведение можно задать с помощью атрибута `placeholder`. В спецификации говорится: «Для добавления более подробной подсказки или совета лучше использовать атрибут `title`».

Обычно для оформления подсказок используется более светлый оттенок, чем для обычного текста, который пользователь вводит в данном поле. Для стилизации можно использовать `::-webkit-input-placeholder`, `:-moz-placeholder` и `-ms-input-placeholder`. Чтобы обеспечить совместимость с будущими версиями браузеров, также добавляйте `-o-input-placeholder`. Это экспериментальный формат, а не часть официальной спецификации CSS.

Важно также помнить, что атрибут placeholder не заменяет элемент `<label>`.

АТРИБУТ REQUIRED

Новый атрибут `required` можно использовать для `<textarea>` и большинства полей ввода (кроме случаев, когда атрибут `type` принимает значения `hidden`, `image` или задает кнопочный тип, такой как `submit`). Браузер не разрешит пользователю отправить форму, если обязательные поля будут пустыми, и вернет ошибку.

Также мы рекомендуем добавлять к таким полям ARIA-атрибут `aria-required`, который будет полезен для вспомогательных технологий (ARIA подробно обсуждалась в главе 2).

АТРИБУТ MULTIPLE

Тип `<input type=file>` не является новым в HTML5, но теперь благодаря атрибуту `multiple` пользователь может загружать несколько файлов.

```
<input type=file multiple>
```

В HTML 5 так было делать нельзя, поэтому разработчикам для достижения того же эффекта приходилось прибегать к помощи Java-апплетов и Flash.