ГЛАВА 7

JavaScript и проверка достоверности форм

Имея дело с формой, неизбежно приходится решать судьбу данных, введенных в этой форме. Одно из первых практических применений JavaScript состояло в предоставлении способа проверки достоверности данных на стороне клиента вместо того, чтобы обращаться за этой услугой к серверу, посылая ему данные и получая результаты их проверки обратно. В то время проверка достоверности форм была чем-то особенным, не имела никакой практической поддержки в прикладном программном интерфейсе API и никакого реального внедрения в браузерах. Вместо этого программистам приходилось связывать вместе события и основные операции над текстом, чтобы предоставить удобный пользовательский интерфейс.

Ныне проверка достоверности форм находится в намного лучшем состоянии. В современные браузеры внедрен прикладной программный интерфейс API, предоставляющий обширные средства проверки достоверности форм в HTML и CSS. В распоряжении разработчиков веб-приложений имеются также регулярные выражения, которые намного более эффективны для проверки достоверности данных, несмотря на всю сложность их синтаксиса, чем, например, последовательный перебор строки по символам.

В этой главе основное внимание уделяется возможностям JavaScript в отношении форм. И хотя она посвящена в основном проверке достоверности форм, в ней также рассматриваются основные усовершенствования во взаимодействии Java Script с формами и новые прикладные программные интерфейсы API, доступные для обращения с формами.

Проверка достоверности форм в HTML и CSS

Как упоминалось выше, проверка достоверности форм прошла долгий путь развития с тех пор, как появился язык JavaScript. Чтобы оценить текущее состояние проверки достоверности форм, нужно рассмотреть не только возможности Java Script, но и HTML5 и CSS. Начнем со стороны HTML. За последние несколько лет язык HTML получил дальнейшее развитие, и в него было внедрено немало средств благодаря трудам группы WHATWG (Web Hypertext Application Technology Working Group — Рабочая группа по разработке гипертекстовых приложений для Интернета).

Эта организация способствовала дальнейшей эволюции и обновлению языка HTML до того, что теперь называется HTML5. Но обсуждение спецификации HTML5 выходит за рамки этой главы, и поэтому мы отсылаем читателей за дополнительными сведения к книге *HTML5 Programmer's Reference* Джонатана Рейда (Jonathan Reid), вышедшей в издательстве Apress в 2015 г.

Особо следует отметить усовершенствования HTML5 в отношении элементов управления форм. Эти усовершенствования можно разделить на следующие две обширные категории: внедрение новых элементов управления или стилей оформления элементов управления (полей ввода URL, селекторов данных и прочих), а также проверка достоверности форм. Сначала мы уделим внимание последней категории. Простая проверка достоверности форм была перенесена в язык HTML без необходимости прибегать к средствам JavaScript. Такая проверка достоверности доступна благодаря внедрению некоторых атрибутов в элементы управления формы. В качестве простого примера служит атрибут `required`, который применяется вместе с элементами ввода и требует обязательного наличия значения в поле перед тем, как форма будет передана на рассмотрение. Характерный пример разметки простой формы приведен в листинге 7.1.

Листинг 7.1. Простая форма

```
<!DOCTYPE html>
<html>
<head>
  <title>A basic form</title>
</head>
<body>
<h2>A basic form</h2>
<p>Please provide your first and last names.</p>

<form>
  <fieldset>
    <label for="firstName">First Name:</label><br/>
    <input id="firstName" name="firstName" type="text" required/><br/>
    <label for="lastName">Last Name:</label><br/>
    <input type="text" name="lastName" id="lastName"/><br/>
  </fieldset>
  <input type="submit" value="Submit the form"/>
  <input type="reset" value="Reset the form"/>
</form>

</body>
</html>
```

В данной форме обратите внимание на поле ввода с идентификатором `firstName`, где введен упоминавшийся ранее атрибут `required`. Если попытаться передать эту форму без заполнения данного поля, то получится результат, аналогичный приведенному на рис. 7.1.

Этот результат приблизительно одинаков в браузерах Chrome и Internet Explorer 11 (в браузере Chrome пустое поле не обрамлено красной рамкой, тогда как в Internet Explorer такая рамка имеется и хорошо заметна). Если сделать поля

`firstName` и `lastName` обязательными для заполнения, то обрамляющая рамка появится вокруг каждого поля, но всплывающая подсказка будет связана только с первым полем, где не введены данные. А как насчет специализации всплывающей подсказки? Мы рассмотрим вскоре данный вопрос, но для этого нам придется обратиться к средствам JavaScript.

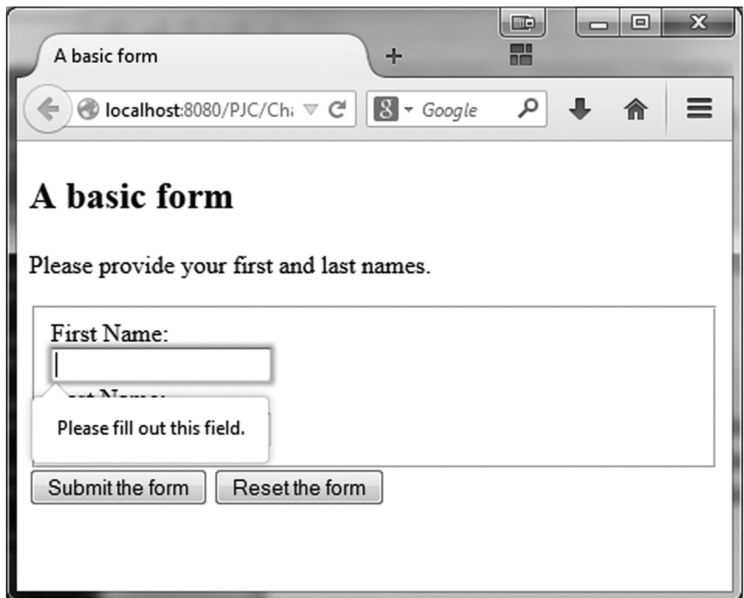

Рис. 7.1. В этой простой форме отсутствует имя того, кто ее заполнял

С помощью следующих атрибутов HTML можно активизировать и ряд других видов проверки достоверности.

- **Атрибут `pattern`.** Принимает регулярное выражение в качестве аргумента. Заключать регулярное выражение в знаки косой черты необязательно. Языковые средства HTML для регулярных выражений такие же, как и для Java Script (со всеми их достоинствами и недостатками). Этот атрибут присоединяется к элементу ввода. Следует, однако, иметь в виду, что типы ввода `email` и `url` подразумевают ввод значений, согласующихся с адресами электронной почты и URL соответственно. Проверка достоверности по шаблону не действует в версиях браузеров Safari 8, iOS Safari 8.1 и Opera Mini.
- **Атрибут `step`.** Требует, чтобы введенное значение было кратно указанной величине шага. Его действие ограничивается типами ввода даты и времени, а также `number`, `range`. Проверка достоверности по шагу действует в версиях браузеров Chrome 6.0, Firefox 16.0, IE 10, Opera 10.62 и Safari 5.0.
- **Атрибуты `min/max`.** Обозначают минимальное и максимальное значения, которые соответствуют не только числам, но и датам и времени. Такая проверка достоверности действует в версиях браузеров Chrome 41, Opera 27 и Chrome 41 для Android.

- **Атрибут `maxlength`.** Обозначает максимальную длину в символах для ввода данных в поле. Действителен только для типов ввода `text`, `email`, `search`, `password`, `tel` и `url`. Такая проверка достоверности обычно не особенно препятствует пользователю вводить слишком много данных в том поле, к которому присоединен данный атрибут. Она действует во всех современных браузерах.

Проверку достоверности в целом можно отключить на уровне формы двумя способами. С одной стороны, можно ввести атрибут `formnovalidate` в разметку кнопки Submit, а с другой стороны — ввести атрибут `novalidate` в сам элемент разметки формы.

CSS

Проверка достоверности вводимых данных была внедрена не только в HTML5, но и в спецификации CSS. Элементы формы, находящиеся в недостоверном состоянии, могут быть доступны через псевдокласс `:invalid`. К сожалению, реализация этого псевдокласса оставляет желать лучшего. Во-первых, элементы формы проверяются на их достоверность при загрузке страницы. Так, если в форме имеется следующее стилевое оформление:

```
:invalid { background-color: yellow }
```

то при загрузке страницы многие поля будут выделены желтым цветом фона. И во-вторых, в браузерах Chrome и Internet Explorer псевдокласс `:invalid` применяется только в элементах формы, тогда как в браузере Firefox — ко всей форме в целом, если в форме имеется любой недостоверный элемент. Характерный пример применения псевдокласса `:invalid` демонстрируется в листинге 7.2.

Листинг 7.2. Применение псевдокласса `:invalid`

```
<!DOCTYPE html>
<html>
<head>
  <title>A basic form</title>
  <style>
    :invalid {
      background-color: yellow
    }
  </style>
</head>
<body>
<h2>A basic form</h2>

<p>Please provide your first and last names.</p>


<form>
  <fieldset>
    <label for="firstName">First Name:</label><br/>
    <input id="firstName" name="firstName" type="text" required/><br/>
    <label for="lastName">Last Name:</label><br/>
```

```
    <input type="text" name="lastName" id="lastName"/><br/>
  </fieldset>
  <input type="submit" value="Submit the form"/>
  <input type="reset" value="Reset the
form"/>
</form>

</body>
</html>
```

В примере из листинга 7.2 вся форма отображается в окне браузера Firefox на желтом фоне, поскольку один из ее элементов находится в недействительном состоянии. Чтобы устранить этот недостаток, достаточно изменить стилевое оформление с `:invalid` на `input:invalid`. Это позволит добиться согласованного поведения формы во всех браузерах.

Для проверки достоверности в CSS предоставляются и другие псевдоклассы, в том числе следующие.

- **`:valid`**. Охватывает элементы, находящиеся в действительном состоянии.
- **`:required`**. Получает элементы разметки, в атрибуте `required` которых установлено логическое значение `true`.
- **`:optional`**. Получает элементы разметки, в которых отсутствует установленный атрибут `required`.
- **`:in-range`**. Предназначен для тех элементов разметки, которые находятся в своих минимальных или максимальных пределах. Не поддерживается в браузере Internet Explorer.
- **`:out-of-range`**. Предназначен для тех элементов разметки, которые выходят за свои минимальные или максимальные пределы. Не поддерживается в браузере Internet Explorer.

И наконец, рассмотрим эффекты красного свечения и всплывающего сообщения. В частности, красное свечение появляется вокруг недостоверного элемента в окне браузера Firefox после передачи формы на рассмотрение. (А в браузере Internet Explorer вокруг недостоверного элемента появляется красная рамка, но без свечения.) Эффект красного свечения воспроизводится в браузере Firefox с помощью псевдокласса `:-moz-ui-invalid`, но такое поведение можно переопределить следующим образом:

```
:-moz-ui-invalid { box-shadow: none }
```

К сожалению, в браузере Internet Explorer такой эффект не воспроизводится с помощью псевдокласса. Это означает, что мы достигли предела в проверке достоверности форм только средствами HTML и CSS. Но ведь у проверки достоверности форм имеются одни возможности, которые хотелось быть реализовать, а другие — не реализовать. И для этой цели нам, увы, придется снова обратиться к услугам JavaScript.

Проверка достоверности форм в JavaScript

Благодаря главным образом действующему стандарту HTML5 в языке JavaScript теперь появился согласованный прикладной программный интерфейс API для проверки достоверности форм. Он опирается на относительно простой критерий

проверки достоверности: имеется ли у данного элемента формы процедура проверки достоверности? Если таковая имеется, то проходит ли элемент проверку достоверности? Если он не проходит такую проверку, то почему? С этим процессом тесно связаны точки логического доступа для кода JavaScript через вызовы методов или перехват событий. И хотя эта система надежна, нельзя сказать, что она не требует некоторого усовершенствования. Но не будем забегать вперед.

Чтобы проверить элемент формы на достоверность, проще всего вызвать для него метод `checkValidity()`. Такой метод теперь имеется у объекта JavaScript, поддерживающего каждый элемент формы. Этот метод получает доступ к ограничению на проверку достоверности, установленному в HTML-разметке проверяемого элемента. Текущее значение элемента проверяется по каждому ограничению. Если оно не соответствует любому из ограничений, то метод `checkValidity()` возвращает логическое значение `false`, а иначе — логическое значение `true`. Вызовы метода `checkValidity()` не ограничиваются отдельными элементами. Они могут быть сделаны и относительно дескриптора формы. В таком случае вызов метода `checkValidity()` делегируется каждому из элементов формы. Если в результате всех подчиненных вызовов возвращается логическое значение `true`, то форма признается в целом достоверной. А если в результате любого из подчиненных вызовов возвращается логическое значение `false`, то форма считается недостоверной.

Помимо получения простого логического ответа о достоверности проверяемого элемента формы, можно выяснить причину, по которой он не прошел проверку достоверности. Свойство `validity` любого элемента формы содержит объект `ValidityState` со сведениями обо всех возможных причинах, по которым этот элемент не прошел проверку достоверности. Перебрав свойства этого объекта, можно выяснить конкретную причину непрохождения элементом проверки достоверности. Эти свойства перечислены в табл. 7.1.

Таблица 7.1. Свойства объекта `ValidityState`

Свойство	Пояснение
`valid`	Обозначает, является ли значение элемента достоверным. Проверку достоверности элемента формы следует начинать именно с этого свойства
`valueMissing`	Обозначает, что у элемента формы отсутствует обязательное значение
`patternMismatch`	Обозначает непрохождение проверки по шаблону, указанному в регулярном выражении
`rangeUnderflow`	Обозначает, что проверяемое значение оказывается меньше минимальной величины
`rangeOverflow`	Обозначает, что проверяемое значение оказывается больше максимальной величины
`stepMismatch`	Обозначает, что проверяемое значение не соответствует достоверной величине шага
`tooLong`	Обозначает, что проверяемое значение оказывается больше допустимой максимальной длины в символах
`typeMismatch`	Обозначает, что проверяемое значение не соответствует типу ввода **`email`** или **`url`**
`customError`	Принимает логическое значение **`true`**, если возникла специальная ошибка
`badInput`	Обозначает универсальную причину, когда браузер считает, что значение недостоверно ни по одной из перечисленных выше причин; не реализовано в браузере Internet Explorer

Процесс проверки достоверности элемента формы, собственно, состоит в проверке свойства `validity`. Рассмотрим этот процесс на конкретном примере. В листинге 7.3 приведена подходящая для этой цели часть HTML-разметки формы.

Листинг 7.3. HTML-разметка формы

```
<body>
<h2>A basic form</h2>

<p>Please fill in the requested information.</p>

<form id="nameForm">
  <div id="fields">
    <label for="firstName">First Name:</label><br/>
    <input id="firstName" name="firstName" type="text"
           class="foo" required/><br/>
    <label for="lastName">Last Name:</label><br/>
    <input type="text" name="lastName" id="lastName" required/><br/>

    <label for="phone">Phone</label><br/>
    <input type="tel" id="phone"/><br/>
    <label for="age">Age (must be over 13):</label><br/>
    <input type="number" name="age" id="age" step="2"
           min="14" max="100"/><br/>
    <label for="email">Email</label><br/>
    <input type="email" id="email"/><br/>
    <label for="url">Website</label><br/>
    <input type="url" id="url"/><br/>
  </div>

  <div id="buttons">
    <input id="overallBtn" value="Check overall validity"
           type="button"/>
    <input id="validBtn" type="button" value="Display validity"/>
    <input id="submitBtn" type="submit" value="Submit the form"/>
    <input type="reset" id="resetBtn" value="Reset the form"/>
  </div>
</form>

<div>
  <h2>Validation results</h2>
  <div id="vResults"></div>
  <div id="vDetails"></div>
</div>

</body>
```

Обратите внимание на то, что в данной форме кнопки проверки, размеченные элементами `submit`, `reset` и `validity`, находятся в отдельном дескрипторе `<div>`. Благодаря этому упрощается применение метода `document.querySelectorAll()` для извлечения только нужных полей данной формы, также находящихся в отдельном

дескрипторе `<div>`. А теперь перейдем непосредственно к коду JavaScript, приведенному в листинге 7.4.

Листинг 7.4. Проверка достоверности формы

```
window.addEventListener( 'DOMContentLoaded', function () {
  var validBtn = document.getElementById( 'validBtn' );
  var overAllBtn = document.getElementById( 'overallBtn' );
  var form = document.getElementById( 'nameForm' );
    // или document.forms[0]
  var vDetails = document.getElementById( 'vDetails' );
  var vResults = document.getElementById( 'vResults' );

  overallBtn.addEventListener( 'click', function () {
    var formValid = form.checkValidity();
    vResults.innerHTML = 'Is the form valid? ' + formValid;
  } );

  validBtn.addEventListener( 'click', function () {
    var output = '';
    var inputs = form.querySelectorAll( '#fields > input' );

    for ( var x = 0; x < inputs.length; x++ ) {
      var el = inputs[x];
      output += el.id + ' : ' + el.validity.valid;
      if (! el.validity.valid) {
        output += ' [';
        for (var reason in el.validity) {
          if (el.validity[reason]) {
            output += reason
          }
        }
        output += ']';
      }
      output += '<br/>'
    }

    vDetails.innerHTML = output;
  } );
} );
```

Весь блок кода в данном примере связан с событием, наступающим по завершении загрузки документа, построенного по модели DOM. Напомним, что в данном случае не предпринимается попытка ввести обработчики событий в те элементы формы, которые еще не были созданы. Сначала извлекаются нужные элементы в пределах страницы, а именно: две кнопки проверки достоверности, области вывода `div` и форма. Затем устанавливается обработка событий для общей проверки достоверности. Но на этот раз ради простоты проверяется достоверность всей формы в целом. А результаты этой проверки выводятся в области `vResults`.

Второй обработчик событий охватывает проверку отдельного состояния достоверности каждого элемента формы. Соответствующие элементы извлекаются с помощью метода `querySelectorAll()` для получения всех полей по их идентификаторам в дескрипторе `<div>`. (Сделать это намного проще, чем писать обширный CSS-селектор для обнаружения тех типов ввода, которые не включают в себя кнопки передачи, сброса и проверки достоверности формы.) После получения нужных элементов остается лишь перебрать их и проверить содержимое свойства `valid`, подчиненного их свойству `validity`. Если данное свойство содержит логическое значение `false`, т.е. элемент формы недостоверный, то выводится причина его недостоверности. Опробуйте данный пример с самыми разными значениями, вводимыми в форме.

Данный пример выявляет ряд интересных особенностей. Прежде всего, если загрузить страницу и щелкнуть на кнопке Display validity (Показать достоверность), то поля `firstName` и `lastName` окажутся недостоверными, как и следовало ожидать, поскольку они пусты. Но поля `phone`, `age`, `email`, и `url` окажутся достоверными, хотя они также пусты! Если заполнение поля не является обязательным, то его пустое значение оказывается достоверным.

Следует также заметить, что поле `email` проходит две проверки достоверности: подразумеваемую проверку достоверности адреса электронной почты и проверку по шаблону. Попробуйте ввести неверный адрес электронной почты, в котором отсутствуют такие составляющие, как, например, `@foo.com`, и обнаружите непрохождение сразу нескольких проверок достоверности. А браузер Firefox сообщит, что значение не прошло проверку на соответствие типу ввода `typeMismatchbadInput`, если вы введете неполный адрес электронной почты (например, только имя пользователя без указания домена). При проверке достоверности формы нельзя полагаться только на свойство `valid`, поскольку нужно также знать причины, по которым элемент формы не прошел проверку достоверности, чтобы сообщить эту важную информацию пользователю. Ведь он не сможет в конечном итоге передать заполненную форму на рассмотрение, не соблюдя различные ограничения, накладываемые на проверку достоверности формы.

Проверка достоверности и пользователи

До сих пор мы уделяли основное внимание техническим вопросам проверки достоверности форм. Но нам нужно также обсудить, когда именно должна выполняться такая проверка. Для этого имеется целый ряд возможностей. Простое применение прикладного программного интерфейса API для проверки достоверности означает, что такая проверка автоматически происходит в момент передачи формы на рассмотрение. А благодаря методу `checkValidity()` имеется возможность начать проверку достоверности в любом заданном элементе именно тогда, когда это потребуется. Но, как показывает практика, проверку достоверности лучше всего начинать как можно раньше, хотя все зависит от конкретного проверяемого элемента формы. Начнем с того, что проверку достоверности следует начинать при изменении значения в поле форме. Обработчик событий, связанный с подобным изменением, достаточно присоединить к элементу управления формой и вызвать для него метод `checkValidity()`. Работая с прикладным программным интерфейсом API, совсем нетрудно ответить на вопрос, когда именно следует начинать проверку достоверности формы.

Но что делать, если мы не работаем с прикладным программным интерфейсом API для проверки достоверности форм? Одно из значительных ограничений на работу с этим прикладным интерфейсом состоит в том, что в нем не предусмотрены возможности для специальных проверок достоверности форм. В частности, фрагмент специального кода нельзя привязать к функции для выполнения в виде процедуры проверки достоверности. Но в какой-то момент такая потребность может, без сомнения, возникнуть. В таком случае имеет общий и практический смысл связать проверку достоверности с обработчиком событий, наступающих при изменениях в элементах формы. Из данного правила возможны исключения. Рассмотрим поле, в котором требуется сделать Ajax-вызов для проверки достоверности значения, возможно, исходя из первых нескольких символов, введенных в данном поле. В этом случае проверку достоверности следует связать с событием от клавиатуры, внедрив также функциональные возможности автоматического заполнения вводимых данных. Характерный тому пример рассматривается в следующей главе, посвященной технологии Ajax.

На какой бы стадии ни было решено начать проверку достоверности, нужно учитывать также интересы пользователей. Ведь пользователю будет очень неприятно обнаружить после заполнения формы, что бóльшая часть введенных им данных оказалась недостоверной по самым разным причинам. Пользователям удобнее устранять ошибки непосредственно в полях ввода при заполнении формы, чем получать список ошибок после передачи формы на рассмотрение.

События проверки достоверности

Еще одно дополнение прикладного программного интерфейса API для проверки достоверности форм состоит в том, что недостоверные элементы формы способны теперь генерировать событие в связи с непрохождением проверки достоверности. Такое событие генерируется лишь в ответ на вызов метода `checkValidity()`. Этот метод может быть вызван как для самого проверяемого элемента, так и для содержащей его формы. Событие в связи с непрохождением проверки достоверности не всплывает. У форм такие события отсутствуют, несмотря на то, что формы могут быть недостоверными.

Событие проверки достоверности можно, как обычно, перехватить, вызвав метод `addEventListener()` для элемента управления, инициировавшего данное событие. Оказавшись в обработчике событий, сам объект события не предоставляет никакой информации о проверке достоверности. Для этого придется извлечь проверяемый элемент по ссылке `event.target`, а затем обратиться к его свойству `validity`, чтобы выяснить конкретную причину, по которой элемент оказался недостоверным. Но любопытно, что если вызвать метод `preventDefault()` для объекта события, то поведение стилевого оформления в браузере не будет затрагивать недостоверные элементы. Следует иметь в виду, что изменения в стилевом оформлении согласованно делаются только после передачи формы на рассмотрение. (Так, в браузере Firefox изменения в стилевом оформлении происходят лишь в том случае, если изменяется значение в элементе управления формы или фокус ввода переносится от него на другой элемент.) В разных браузерах это означает следующее.

- В браузере Chrome стилевое оформление не распространяется на недостоверные элементы формы, и хотя элементы снабжаются всплывающим сообщением, оно будет подавляться для недостоверного элемента.

- В браузере Firefox будет подавляться всплывающее сообщение, но не эффект красного свечения вокруг недостоверного элемента.
- В браузере Internet Explorer будет подавляться как всплывающее сообщение, так и красная рамка вокруг недостоверного элемента.

Рассмотрим пример, в котором демонстрируется такое поведение. Начнем с HTML-формы, которая отчасти нам уже знакома (листинг 7.5).

Листинг 7.5. Форма с событиями проверки достоверности

```
<!DOCTYPE html>
<html>
<head>
  <title>A basic form</title>
  <style>
    input:invalid {
      background-color: yellow
    }
  </style>
</head>
<body>
<h2>A basic form</h2>

<p>Please provide your first and last names.</p>

<form id="nameForm">
  <fieldset>
    <label for="firstName">First Name:</label><br/>
    <input id="firstName" name="firstName" type="text" required/><br/>
    <label for="lastName">Last Name:</label><br/>
    <input type="text" name="lastName" id="lastName" required/><br/>
  </fieldset>
  <div>
    <input type="submit" value="Submit the form"/>
    <input type="reset" value="Reset the form"/>
  </div>
  <div>
    <input id="firstNameBtn" type="button"
           value="Check first name validity."/>
    <input id="formBtn" type="button" value="Check form validity"/>
    <input id="preventBtn" type="button"
           value="Prevent default behavior"/>
    <input id="restoreBtn" type="button"
           value="Restore default behavior"/>
  </div>
</form>

<div id="vResults"></div>

<script src="listing_7_5.js"></script>
```

```
</body>
</html>
```

Обратите внимание на то, что недостоверные элементы ввода были дополнены стилевым оформлением. Такое стилевое оформление не связано со стандартным поведением для события в связи с непрохождением проверки достоверности. А теперь рассмотрим код сценария, поддерживающего данную HTML-форму, как показано в листинге 7.6.

Листинг 7.6. Обработка событий проверки достоверности в сценарии JavaScript

```
window.addEventListener( 'DOMContentLoaded', function () {
  var outputDiv = document.getElementById( 'vResults' );
  var firstName = document.getElementById( 'firstName' );

  firstName.addEventListener("focus", function(){
    outputDiv.innerHTML = '';
  });

  function preventDefaultHandler( evt ) {
    evt.preventDefault();
  }

  firstName.addEventListener( 'invalid', function ( event ) {
    outputDiv.innerHTML = 'firstName is invalid';
  } );

  document.getElementById( 'firstNameBtn' ).addEventListener(
          'click', function () {
    firstName.checkValidity();
  } );

  document.getElementById( 'formBtn' ).addEventListener(
          'click', function () {
    document.getElementById( 'nameForm' ).checkValidity();
  } );

  document.getElementById( 'preventBtn' ).addEventListener(
          'click', function () {
    firstName.addEventListener( 'invalid', preventDefaultHandler );
  } );

  document.getElementById( 'restoreBtn' ).addEventListener(
          'click', function () {
    firstName.removeEventListener( 'invalid', preventDefaultHandler );
  } );

} );
```

Как обычно, весь код сценария активизируется после того, как наступит событие `DOMContentLoaded`. В коде этого сценария имеется элементарный обработчик событий в связи с непрохождением проверки достоверности данных в поле `firstName`.

Этот обработчик событий выводит результаты в области `vResults`, размеченной дескриптором `<div>`. Кроме того, в коде этого сценария введены обработчики событий, наступающих в специализированных кнопках. Сначала создаются две служебные кнопки: одна — для проверки достоверности данных в поле `firstName`, а другая — для проверки достоверности данных во всей форме в целом. И наконец, в коде этого сценария реализовано поведение для отмены или восстановления стандартного поведения, связанного с недостоверными элементами. Опробуйте этот сценарий!

Специальная настройка проверки достоверности

Теперь в нашем распоряжении имеются почти все инструментальные средства для полного контроля над проверкой достоверности форм. В частности, мы можем выбрать активизацию конкретной проверки достоверности, контролировать момент ее выполнения, перехватывать недостоверные события и предотвращать стандартное поведение, особенно в отношении стилевого оформления. Но, как обсуждалось ранее, мы не можем, к сожалению, специально настроить конкретные процедуры проверки достоверности. Что же нам тогда остается? Ведь нам хотелось бы проследить за тем, какое сообщение о результатах проверки достоверности всплывает, когда пользователь передает форму на рассмотрение. Но внешний вид всплывающего сообщения не подлежит специальной настройке. Напомним о ряде ограничений, наложенных на прикладной программный интерфейс API для проверки достоверности и его реализацию.

Чтобы изменить сообщение, которое появляется, когда поле оказывается недостоверным, следует вызвать функцию `setCustomValidity()`, связанную с элементом управления формы, передав ей в качестве аргумента символьную строку с текстом всплывающего сообщения. Но такое действие повлечет за собой разные побочные эффекты. Так, в браузере Firefox проверяемое поле будет воспроизведено как недостоверное с эффектом красного свечения во время загрузки страницы. А вызов той же самой функции `setCustomValidity()` в браузере Internet Explorer или Chrome не окажет никакого действия во время загрузки страницы. Как упоминалось ранее, стилевое оформление может быть отключено в браузере Firefox путем переопределения псевдокласса `:-moz-ui-invalid`. Но дело в том, что, когда вызывается функция `setCustomValidity()`, в свойстве `customError` объекта `ValidityState` проверяемого элемента управления формы устанавливается логическое значение `true`. Это означает, что в свойстве `valid` объекта `ValidityState` устанавливается логическое значение `false`, а следовательно, проверяемый элемент формы оказывается недостоверным. И все это, конечно, отражается в содержимом сообщения о проверке достоверности! В итоге функция `setCustomValidity()` оказывается практически бесполезной.

В качестве альтернативы можно было бы воспользоваться полизаполнением. Имеется длинный перечень полизаполнений не только для проверки достоверности форм, но и для других элементов HTML5, возможно, не имеющих поддержки в каждом браузере, с которым приходится работать. Этот перечень можно найти по следующему адресу:

```
https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills
```

Предотвращение проверки достоверности форм

Мы не рассмотрели еще один аспект проверки достоверности форм: ее отключение. Большая часть этой главы была посвящена применению нового прикладного программного интерфейса API для проверки достоверности и рассмотрению присущих ему ограничений. Но что, если возникнет некоторое препятствие (программная ошибка или несогласованность) для применения этого прикладного программного интерфейса? В таком случае нам останется одно из двух: отменить автоматическую проверку достоверности или заменить ее своей. Первое нас интересует больше, поскольку второе просто означает повторную реализацию прикладного программного интерфейса API по собственному усмотрению. Чтобы отключить проверку достоверности, достаточно ввести атрибут `novalidate` в элемент разметки формы. А для того чтобы предотвратить проверку достоверности при выборе кнопки передачи формы, следует ввести атрибут `formnovalidate` в разметку этой кнопки, хотя это не отменит проверку достоверности формы в целом. Если же прикладной программный интерфейс API для проверки достоверности потребуется заменить собственным, то для полной отмены проверки достоверности формы (на уровне родительского элемента) вряд стоит употреблять атрибут `novalidate`.

Резюме

В этой главе был рассмотрен новый прикладной программный интерфейс API для проверки достоверности форм в коде JavaScript. Несмотря на присущие этому интерфейсу ограничения, он является эффективным средством для автоматической проверки достоверности данных, вводимых пользователями в формах. Проверка достоверности формы происходит автоматически при ее передаче на рассмотрение, но в то же время ее можно осуществить в любой удобный момент по своему выбору. А при необходимости проверку достоверности можно отключить. Кроме того, имеется возможность специально настроить внешний вид достоверных и недостоверных элементов формы, а также сообщение, выводимое, когда элемент оказывается недостоверным.

Применение прикладного программного интерфейса API для проверки достоверности форм не лишено трудностей. Ему недостает ряда очень важных возможностей для специальной настройки проверки достоверности, в том числе стилевого оформления сообщений об ошибках или специальной настройки процедур проверки достоверности. А после исправления этих недостатков незначительные отличия в реализации среди основных браузеров могут быть устранены впоследствии. Одни официально признанные части прикладного программного интерфейса API (например, свойство `willValidate`) в настоящее время не реализованы, а другим (в частности, функции `setCustomValidity()`) присущи непреодолимые трудности.

В целом внедрение прикладного программного интерфейса API для проверки достоверности форм стало крупным шагом в дальнейшем развитии JavaScript, HTML, CSS и браузеров. Остается только надеяться, что он будет усовершенствован в будущем.