Глава 14

Таблицы стилей — управляем сайтом легко

Современные текстовые процессоры, например, Microsoft Word, позволяют пользователю определять *стиль*, т. е. набор правил оформления и форматирования, который может применяться к различным частям документа. С другой стороны, в стандартном HTML для присвоения элементу определенных свойств (цветов, размеров, положения на странице) их приходится каждый раз описывать заново, даже если в документе находится 10 или 100 таких элементов.

Каскадные таблицы стилей (Cascading Style Sheets, сокращенно CSS) предназначены для хранения информации о стилях Web-страниц. Механизм CSS позволяет отделить *форматирование* HTML-документа от его *содержания* и, таким образом, экономично и удобно использовать однажды созданные правила для оформления любого количества документов.

В настоящее время CSS — одна из самых перспективных технологий, облегчающих и автоматизирующих разработку Web-сайтов. В этой главе мы научимся создавать стили и применять их к своим страницам. Стандарты CSS стремительно развиваются, поэтому мы рассмотрим только устоявшееся "ядро" языка, одинаково хорошо понимаемое всеми современными браузерами.

14.1. Общие свойства таблиц стилей

Итак, CSS можно определить как набор стилевых правил для дополнительного форматирования тегов HTML.

Хотя таблицы стилей часто сравнивают со стилевыми указаниями Microsoft Word, между ними нет прямой аналогии. Прежде всего, это связано с тем, что установки CSS действуют на *все* элементы, заключенные внутри тега, которому присвоен стиль. Так, описав некоторый стиль для тега `<body>`, мы автоматически присваиваем его всему содержимому Web-страницы. В объектно-

ориентированном программировании (ООП) такое поведение объектов называется *наследованием*. С другой стороны, стилевые указания обладают и свойством, называемым в ООП *полиморфизмом* — вложенные элементы могут *переопределять* свойства элементов, в которые они вложены, формируя таким образом собственные стили.

Чтобы лучше понять структуру документа с точки зрения таблиц стилей, рассмотрим хорошо нам знакомый простейший пример:

```
<html><head>
    <title>Заголовок окна</title>
</head><body>
    <h1>Заголовок первого уровня</h1>
    <p>Абзац текста</p>
</body></html>
```

Теперь представим себе этот документ в виде *дерева* с единственным *корнем* — элементом `<html>` и двумя *дочерними элементами* — вложенными в `<html>` тегами `<head>` и `<body>`. Элемент `<head>` имеет дочерний элемент `<title>`, для которого он сам является *родителем*. Для элемента `<html>` элемент `<title>` будет *потомком*, а сам элемент `<html>` — *предком* `<title>`. Элемент `<body>` имеет дочерние элементы `<h1>` и `<p>`, являющиеся между собой *сестрами* (или братьями, если кому-то так больше нравится). При этом тег `<h1>` — *предшествующий* элемент, а `<p>` — *следующий*.

Как видим, все достаточно просто. Важно, что каждый тег нашего документа имеет только одного родителя, за исключением корневого элемента `<html>`, у которого родителя нет. Любой элемент в этой иерархии может либо *определить* собственный стиль, либо *унаследовать* его от родителя или от более отдаленного предка. Легко увидеть, что при этом возможны достаточно сложные наложения правил и некоторые из них могут противоречить друг другу. В CSS предусмотрена и такая возможность, ведь таблицы стилей не случайно названы *каскадными*.

Это означает, что каждому правилу приписан определенный *вес*. Если к конкретному элементу применимы несколько правил, используется то из них, которое имеет больший вес. В результате происходит накопление свойств элементов в соответствии с правилами наследования, и, тем самым, образуется *каскад свойств*, распространяющийся от предков к потомкам. Не вдаваясь в детали механизма наследования, заметим, что наибольший вес имеют стилевые правила, заданные *явно* для текущего элемента, затем учитывается значение атрибута, которое может быть *унаследовано* от родительского элемента или от вышестоящих предков, наконец, в последнюю очередь действует значение по умолчанию.

Так, если для элемента `<body>` в стиле определен синий цвет, а для элемента `<p>` — зеленый, то цвет выделенного текста в конструкции тегов

```
<p>Стили — это <strong>важно</strong>!</p>
```

зависит от того, был ли в стиле задан цвет для тега `<strong>`. Если это так, заданный цвет будет применен, в противном случае слово `важно` будет выведено тем же *зеленым* цветом, что и весь абзац, поскольку тег `<p>` — потомок `<body>`, переопределивший его атрибут цвета с синего на зеленый.

Независимо от способа задания, таблица стилей состоит из набора *операторов*. При этом каждый оператор представляет собой либо *директиву*, либо *правило*. Операторы могут разделяться пробелами или символом перевода строки.

Директива начинается с символа `@` (at) и следующего за ним без пробела ключевого слова. Назначение директив — описания шрифтов, кодировок символа, печатных страниц и другого *внешнего* по отношению к стилевому форматированию содержимого.

Правило состоит из имени-*селектора* и *блока деклараций*, заключенного в фигурные скобки, например:

```
p {
  font-size: 12pt;
  font-family: Arial;
  font-style: normal
}
```

Здесь `p` — это *селектор*, за которым следует блок из трех *деклараций*, каждая из которых состоит из *названия свойства* (например, `font-size`), символа двоеточия и *значения свойства* (например, `12pt`). Декларации должны разделяться символом точки с запятой, построчное их расположение не имеет значения, но удобнее, когда на одной текстовой строке находится только одна.

Согласно определению CSS, все его элементы не зависят от регистра символов.

Подобно документам HTML, таблицы стилей могут содержать *комментарии*, которые служат для пояснения операторов. Комментарий начинается с символов `/*` и заканчивается символами `*/`. Вложенные комментарии не допускаются.

14.2. Способы определения стилей

Существует три основных способа применения таблицы стилей к документу:

1. *Связывание* (linking) — можно связать HTML-документ с таблицей стилей, хранящейся в отдельном файле.
2. *Внедрение* (embedding) — можно встроить таблицу стилей в документ с помощью тега `<style>`, указанного в заголовке документа `<head>`.

3. *Задание стиля для отдельного элемента* (inline) — можно определять элементы стиля "на лету", т. е. указывать их в тегах документа, например, в теге абзаца <p>.

Наиболее перспективен первый способ, называемый также *внешними* таблицами стилей. Очевидно, что он позволяет применить однажды созданный и сохраненный в файле стиль ко всем страницам вашего сайта. Хранить таблицу стилей следует в текстовом файле с расширением css, который можно создать при помощи любого текстового редактора, например, Блокнота Windows. При сохранении такого файла достаточно поменять ему тип с txt на css по алгоритму, описанному в *разд. 4.3*.

Для связывания таблицы стилей с документом HTML в последнем записывается тег следующего вида:

```
<link rel="stylesheet" type="text/css" href="URL">.
```

Здесь атрибут URL, как всегда, обозначает абсолютный или относительный адрес стилевого файла, а остальные атрибуты служат для того, чтобы сообщить браузеру, что на странице используется CSS. Обычно этот тег располагают в теге <head> или между тегами <head> и <body>.

К внешним таблицам стилей следует отнести и директиву @import, позволяющую включать в таблицу стилей другие ранее подготовленные таблицы. Она должна содержать абсолютный или относительный URL импортируемой таблицы и может задаваться в одном из следующих видов:

```
@import "mystyle.css";
@import url(mystyle.css);
```

В случае, когда разработанные стили действуют только в одном документе, первый способ можно заменить вторым, называемым *внутренними* таблицами стилей.

При этом указывать тег <link> не нужно, а таблица стилей будет описана непосредственно внутри документа в теге <style>. Общий вид этого описания показан в листинге 14.1.

Листинг 14.1. Общий вид таблицы стилей

```
<style type="text/css">
<!--
  операторы таблицы стилей
-->
</style>
```

Операторы таблицы заключены в тег HTML-комментария `<!--...-->` для того, чтобы "спрятать" стили от старых браузеров, которые могут показать их просто как неформатированный текст страницы. В настоящее время комментирование потеряло актуальность, т. к. браузеров, не использующих стилей, почти не осталось.

Стилевой файл CSS также имеет общий вид, показанный в листинге 14.1.

Наконец, для определения стиля конкретного тега достаточно указать в его открывающей части атрибут `style` со значением, включающим одну или несколько деклараций CSS:

```
<h1 style="color: blue">Заголовок</h1>.
```

Здесь с помощью свойства CSS с названием `color` мы определили цвет *текущего* заголовка первого уровня как синий.

Этот метод указания стиля поддерживается всеми дочерними тегами тега `<body>`.

Перечисленные три способа определения стиля упорядочены иерархически, что имеет значение при разрешении одинаковых правил, заданных на разном уровне:

- наивысший приоритет имеет оперативное указание стиля в конкретном теге;
- вторым по старшинству является указание стиля в теге `<style>`;
- низший приоритет имеет указание в стилевом файле.

Отсюда очевидно, что третий способ обычно служит лишь для "разового" переопределения свойств какого-либо тега, указанных в стилевом файле. При злоупотреблении оперативным определением стиля теряется основное преимущество CSS — разделение содержания и форматирования.

Приведем также пример, иллюстрирующий поддержку наследования в CSS:

```
<div style="color: blue">
   <h1>Это заголовок внутри раздела</h1>
   <p>Это абзац внутри раздела</p>
</div>
```

Здесь теги `<h1>` и `<p>` унаследуют синий цвет шрифта от тега `<div>`.

14.3. Виды селекторов. Определение классов

Мы уже знаем, что таблица стилей состоит из деклараций, имеющих общий вид:

```
селектор { свойство1: значение1; … ; свойствоN: значениеN }.
```

Чаще всего используется так называемый *селектор типа*, представляющий собой наименование тега HTML без символов < и > (например, `H1`, `P`, `TD`):

```
h1 { color: blue; font-size: 12pt; text-line: center }.
```

Если это определение сделано в теге `<style>` или в стилевом файле, то все заголовки первого уровня, определенные тегом `<h1>`, будут выведены синим шрифтом размером 12 пунктов с выравниванием по центру. Для прочих свойств будут действовать значения по умолчанию.

Существует также *универсальный селектор* `*`, относящийся ко всем элементам документа, но обычно его успешно заменяет `body`.

Если вы хотите определить одно и то же свойство для нескольких тегов HTML, то можно указать для них отдельные декларации:

```
P { font-size: 12pt }
UL { font-size: 12pt }
```

или более компактно перечислить селекторы через запятую:

```
P, UL { font-size: 12pt }.
```

Подобный прием называется *группированием* селекторов.

Иногда возникает необходимость определения двух и более деклараций стиля для одного тега: например, для тега элемента списка `<li>` может понадобиться один вид оформления, если он находится в нумерованном списке и другой — в маркированном. Такие указания легко сделать с помощью *контекстных определений*, задав точную последовательность тегов, для которой будет применен стиль:

```
OL LI { list-type-style: decimal }
UL LI { list-type-style: square }
```

Обратите внимание, что названия элементов *не разделены* запятыми. В противном случае всем тегам списка будет присвоен один и тот же стиль.

Контекстное определение может относиться не только к элементам-родителям, но и к более отдаленным предкам:

```
div * i { color: blue; }.
```

Данное правило будет применяться ко всем элементам `<i>`, которые находятся внутри любых элементов (универсальный селектор), непосредственно заключенных в тег `<div>`.

Селекторы контекстного определения называют также *селекторами потомков*.

Таблицы стилей позволяют также *создавать классы* — совокупности определений, каждое из которых может использоваться в нужном месте страницы. Например, вы можете определить два варианта стиля заголовка `<h1>`.

Определение вариантов отличается от указания стиля лишь тем, что к названию тега добавляется наименование класса, отделенное точкой:

```
H1.blue { color: blue }
H1.red { color: red }.
```

Теперь, включая в документ тег `<h1>`, можно указать в нем конкретный стиль при помощи атрибута `class`:

```
<h1 class=red>Красный заголовок</h1>
<h1 class=blue>Синий заголовок</h1>.
```

Рекомендуется выбирать названия классов, отражающие их назначение.

Классы могут формироваться и без привязки к конкретному тегу, в этом случае их имя начинается с символа точки, например:

```
.normal {
  font-family: Times New Roman; font-size: 12pt; color: maroon
}
```

Здесь описан класс для форматирования текста с именем `normal`. Его действие может распространяться на любые текстовые элементы страниц, для чего достаточно указать в нужном теге атрибут `class="normal"`:

```
<font class="normal">текст</font>.
```

Обратите внимание, что в атрибуте `class` HTML имя класса CSS указывается *без точки*.

CSS позволяет создавать *подмножества* значений атрибута `class`, например правило

```
P.normal.under { text-decoration: underline }
```

будет применено к элементам с атрибутом `class="normal under other"`, но не применено к элементам с атрибутом `class="normal other"`.

Селектор идентификатора в CSS имеет вид `#id`, где после символа `#` записано значение атрибута `id` некоторого элемента. Например, после декларации

```
#p1 { letter-spacing: 0.3em }
```

правило стиля будет применено к элементу с атрибутом `id="p1"`:

```
<p id="p1">Текст разрежен</p>.
```

Несмотря на то, что `id` описывается в HTML как уникальный атрибут тега, все современные браузеры покажут форматирование любого числа элементов, помеченных одинаковым `id` и связанных с соответствующим селектором идентификатора.

В спецификации CSS описаны и другие классы селекторов, но вопрос полной поддержки их браузерами пока не решен.

Основные виды селекторов стиля представлены на рис. 14.1. Внимательный анализ и чтение комментариев листинга 14.2, соответствующего рис. 14.1, помогут вам закрепить полученные знания. На компакт-диске с примерами этот документ записан под именем Glava_14\Selectors.html.

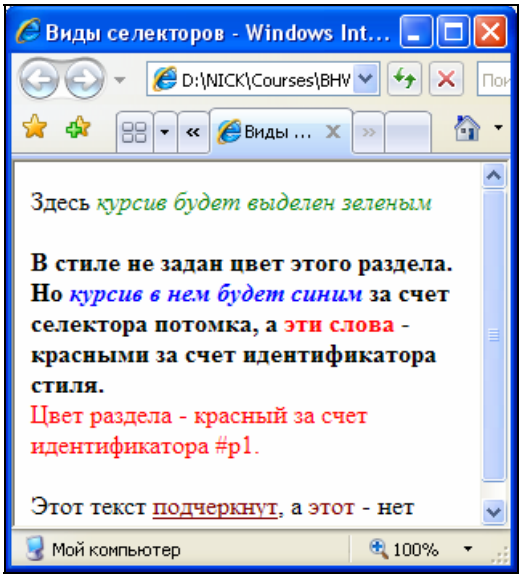

Рис. 14.1. Виды селекторов CSS

Листинг 14.2. Виды селекторов CSS

```
<html><head>
<style type="text/css">
 p { color: black; } /* черный цвет абзацев */
 p i { color : green; }   /* зеленый цвет для курсива в абзаце */
 div * i { color: blue; } /* ... и синий - в разделе внутри любого
элемента */
 #p1 { color: red } /* красный цвет для атрибута id="p1" */
 font.green { color: green; } /* два селектора класса */
 font.red { color: red; }     /* для элемента font */
 .normal { /* класс без привязки к тегу*/
  font-family: Times New Roman; font-size: 12pt; color: maroon
 }
 .normal.under { text-decoration: underline } /* подмножество класса normal */
```

```
</style>
<title>Виды селекторов</title>
</head>
<body>
 <p>Здесь <i>курсив будет выделен зеленым</i></p>
 <div><b>В стиле не задан цвет этого раздела.<br>
  Но <i>курсив в нем будет синим</i> за счет селектора потомка,
  а <span id="p1">эти слова</span> - красными за счет идентификатора
стиля.
 </b></div>
 <div id="p1">Цвет раздела - красный за счет идентификатора #p1.
 <p>Этот текст <font class="normal under">подчеркнут</font>,
 а <font class="normal">этот</font> - нет</p>
 </div></body></html>
```

Все рассмотренные селекторы основывались на *положении элементов в иерархии документа*. Для некоторых целей форматирования такого подхода недостаточно. Прежде всего, это относится к отображению *различных состояний* элемента и выделению *отдельных частей его содержимого*. Так, гиперссылка может иметь состояния "не посещена", "выделена" и "посещена", в абзацах текста может понадобиться выделить первую строку или букву и т. п.

Чтобы задействовать возможности отображения, не описываемые в терминах иерархии элементов, в CSS были включены понятия *псевдоэлементов* и *псевдоклассов*.

Разница между этими понятиями в том, что псевдоклассы применяются к *различным типам* элементов, а псевдоэлементы — к *частям* элементов, таким, как первая строка или первая буква. Селекторы псевдоклассов и псевдоэлементов записываются в виде `элемент:псевдокласс`, например, `A:link`.

Не все существующие селекторы этих видов поддерживаются современными браузерами. Основные псевдоклассы и псевдоэлементы приведены в табл. 14.1.

***Таблица 14.1.** Основные псевдоклассы и псевдоэлементы*

Элемент	Описание
`:link`	Псевдокласс для непосещенной гиперссылки (применим к элементу `<a>`)
`:visited`	Псевдокласс для посещенной гиперссылки (применим к элементу `<a>`)

Таблица 14.1 *(окончание)*

Элемент	Описание
`:hover`	Псевдокласс применяется к элементу, когда пользователь навел на него курсор мыши, но не активировал щелчком
`:active`	Псевдокласс применяется к элементу, когда пользователь активировал его щелчком мыши
`:focus`	Псевдокласс применяется к элементу, когда он получает фокус ввода
`:first-line`	Псевдоэлемент применим к любому блочному элементу и задает стиль отображения его первой строки
`:first-letter`	Псевдоэлемент применим к любому блочному элементу и задает стиль отображения его первой буквы
`:before`	Псевдоэлемент используется для вставки автоматически генерируемого содержимого перед указанным элементом
`:after`	Псевдоэлемент служит для вставки автоматически генерируемого содержимого после указанного элемента

Два последних псевдоэлемента *не поддерживаются* Internet Explorer, что ограничивает их применение. Для генерации содержимого эти псевдоэлементы используют свойство `content`, а способ вставки задается значением свойства `display` (`block`, `inline` или `marker`). Например, для автоматического вывода текстовой метки в конце каждой Web-страницы, к которой применен стиль, можно указать такую декларацию:

```
BODY:after {
 content: "[конец документа]"; display: block; text-align: center;
}
```

Код примера, содержащий псевдоклассы и псевдоэлементы, приведен в листинге 14.3, а результат его исполнения — на рис. 14.2. На компакт-диске с примерами листингу 14.3 соответствует документ Glava_14\PseudoClass.html.

Листинг 14.3. Псевдоклассы и псевдоэлементы

```
<html><head>
<title>Псевдоклассы и псевдоэлементы</title>
<style type="text/css">
P:first-letter { /* Буквица для абзацев */
 font-size: X-LARGE; color: red; font-weight: 600
}
```

```
.normal:first-line {  /* Псевдоэлемент для выделения 1-й строки */
 text-transform: uppercase
}
p:hover { color:blue }  /* Выделяем активный абзац синим, */
a:hover { color:red  }  /* а активную ссылку -  красным   */
a:link { color: #00FF00 }   /* также меняем цвета обычной */
a:visited { color: #006600; /* и посещенной ссылок        */
  text-decoration: none } /* убираем у посещенной ссылки подчеркивание*/
</style>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<p class="normal">Первая строка этого абзаца будет выведена большими буквами.
<p>И у этого, и у предыдущего абзаца есть буквица.
<br>Ссылки <a href="After.html">1</a>, <a href="Selectors.html">2</a>
</body></html>
```

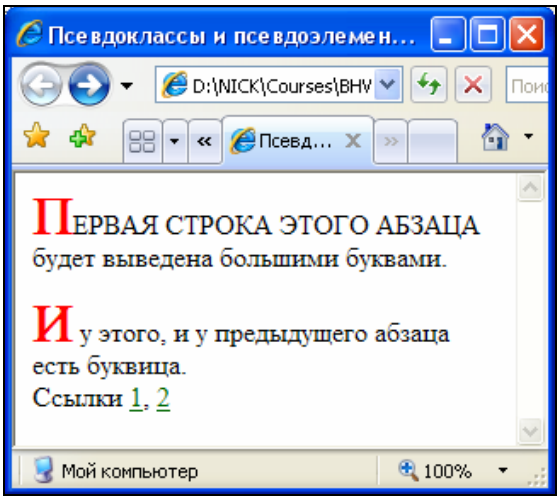

Рис. 14.2. Псевдоклассы и псевдоэлементы

При работе со стилями нам часто придется указывать *размеры* различных объектов. Все размеры в CSS задаются в виде числа, после которого следует *единица измерения*. Определенные в стандарте единицы измерения приведены в табл. 14.2.

Первые пять размеров в табл. 14.2 — *абсолютные*. Они применимы только тогда, когда известны точные физические характеристики устройства отображения, например, диагональ дисплея. Указание абсолютных размеров

в стиле может лишить пользователя возможности управлять размерами шрифта с помощью браузера, а также привести к тому, что при "слишком большом" или "слишком маленьком" разрешении монитора документ станет неудобочитаемым.

Таблица 14.2. *Единицы измерения в CSS*

Обозначение	Описание
`in`	Дюймы (1 дюйм = 2,54 см = 25,4 мм = 72 точки = 6 пик)
`cm`	Сантиметры (1 см = 10 мм = 0,39 дюйма = 2,36 пики = = 28,35 точки)
`mm`	Миллиметры (1 мм = 0,1 см = 0,039 дюйма = 0,24 пики = = 2,84 точки)
`pt`	Точки (1 точка = 1/12 пики = 1/72 дюйма = 0,035 см = = 0,35 мм)
`pc`	Пики (1 пика = 12 точек = 1/6 дюйма = 0,423 см = 4,23 мм)
`em`	Размер шрифта, равный размеру его наибольшей буквы
`ex`	Высота строчных букв шрифта
`px`	Пикселы
`%`	Проценты

Оставшиеся четыре размера — *относительные*, о них следует сказать подробней.

Единицы `em` и `ex` основаны на размере шрифта того элемента, к которому относится использующая их декларация. Величина `em` задает размер шрифта, равный размеру его наибольшей буквы, обычно это буква "M", откуда и произошло сокращение `em`. Величина `ex` интерпретируется как высота строчных букв шрифта, обычно это высота буквы "x", откуда произошло сокращение `ex`.

Указание размера в пикселах зависит как от физических размеров экрана дисплея, так и от его разрешения. Так, пиксел на экране с разрешением 640×480 всегда *больше*, чем на том же экране при установленном разрешении 1280×1024.

Процентные значения указывают на величину элемента в процентах от другой величины. Всюду, где CSS допускает проценты, в описании свойств указывается, какая величина является базовой для расчета значений в процентах. Обычно это соответствующее свойство родительского элемента.

14.4. Свойства шрифта, текста и цветов

Перейдем к более детальному рассмотрению свойств CSS. Основные свойства, применимые для управления шрифтами, приведены в табл. 14.3.

Таблица 14.3. *Свойства шрифта в CSS*

Свойство	Описание
`font-family`	Используется для указания шрифта или шрифтового семейства (см. табл. 14.4), которым будет отображаться документ или его часть. Задает список имен шрифтов и/или семейств, элементы списка разделяются запятыми, наименования шрифтов располагаются в порядке предпочтения
`font-style`	Задает *стиль шрифта* для отображения содержимого. Допустимы значения `normal` (обычный шрифт), `italic` (курсив), `oblique` (наклонный шрифт)
`font-variant`	Задает *вариант шрифта* для отображения содержимого. Допустимы значения `normal` (обычные буквы), `small-caps` (малые прописные)
`font-weight`	Определяет толщину линий (насыщенность) шрифта, имеет значения `lighter`, `normal`, `bold` и `bolder`. Можно задавать также числами `100`, `200`, `...`, `900`. Значение `400` примерно соответствует нормальной насыщенности, а 700 — начертанию, создаваемому тегом `<b>`
`font-size`	Задает *размер шрифта* для отображения содержимого элемента. Размеры могут быть абсолютными и относительными величинами, либо одним из значений `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large` (от наименьшего к наибольшему), либо одним из относительных указаний `smaller` (меньше шрифта родительского элемента) и `larger` (больше шрифта родительского элемента)
`font`	Является сокращением для свойств `font-style`, `font-variant`, `font-weight`, `font-size`, `font-family`, `line-height` и позволяет задать все свойства шрифта одновременно, например `P { font: bolder 12pt Arial, serif }`

В стандартах существуют также следующие свойства:

- `font-stretch` — для определения *выключки* шрифта, т. е. интервала между символами при отображении содержимого;
- `font-size-adjust` — для определения *аспекта* шрифта, т. е. соотношения размеров прописных и строчных букв.

В настоящее время они не поддерживаются.

Свойства, перечисленные в табл. 14.3, применимы ко всем элементам и наследуются дочерними элементами. Указание размеров в процентах допустимо только для свойства `font-size`. Приведем пример:

```
P { font-family: "Times New Roman", serif}.
```

Изучив в *разд. 5.2* тег `<font>` с атрибутом `face`, мы уже знакомы с приемом последовательного указания нескольких шрифтов, желаемых для отображения. Здесь в качестве приоритетного шрифта абзацев указан Times New Roman, при его отсутствии документ будет отображаться любым подходящим шрифтом *семейства* Serif. *Семейства шрифтов* или *родовые шрифты* были введены в стандарт на тот случай, если на компьютере-клиенте не установлен ни один из шрифтов, заданных автором. Тогда браузер использует родовой шрифт, начертание которого *напоминает* авторский. В CSS определены семейства шрифтов, описанные в табл. 14.4.

Таблица 14.4. *Семейства шрифтов в CSS*

Семейство	Описание
`serif`	Пропорциональные шрифты с засечками на буквах. Примеры: Times New Roman, Bodoni, Garamond
`sans-serif`	Пропорциональные шрифты без засечек на буквах. Примеры: Arial, Verdana, Helvetica, Tahoma
`cursive`	Каллиграфические шрифты, стилизованные под рукописный текст с типичными для него соединениями между буквами. Примеры: Script, Zapf-Chancery
`fantasy`	Шрифты декоративного характера. Примеры: Western, Comic
`monospace`	Моноширинные или телетайпные шрифты с одинаковой шириной печатаемых символов. Примеры: Courier New, Prestige, Everson Mono

Как и любые значения атрибутов, состоящие из нескольких слов, подобные имена шрифтов нужно заключать в кавычки.

Обратите внимание, что если такое определение, как `font-family: "Times New Roman"` будет сделано при указании шрифта внутри тега, то у атрибута `style` и свойства шрифта `font-family` должны быть *разные* типы кавычек:

```
<p style="font-family: 'Times New Roman'; color: black ">…</p>.
```

В противном случае, "запутавшись" во вложенных кавычках, браузер может отобразить документ с ошибками. Приведем еще один пример:

```
P { font-size: 12pt }
.little { font-size: 80% }.
```

Здесь для отображения текста абзацев установлен шрифт размером 12 пунктов, а также описан класс `little`, размер шрифта которого составляет 80% от базового.

В стандарте CSS имеется также директива `@font-face`, предназначенная для загрузки внешнего шрифта из Интернета. Например, указание ее в виде

```
@font-face { font-family: MyFont; src: http://www.fonts.com/MyFont.eot }
```

приведет к загрузке шрифта с указанного URL. При этом шрифт должен храниться на сервере в формате Embedded OpenType, являющемся расширением стандартного формата TrueType.

Рассмотрим набор свойств, определяющих различные параметры отображения текста (табл. 14.5).

***Таблица 14.5.** Свойства текста в CSS*

Свойство	Описание
`text-indent`	Определяет отступ первой строки при отображении блочных элементов. Отступ может быть отрицательным и задается либо абсолютным значением, либо в процентах от ширины вмещающего блока
`text-align`	Задает выравнивание текста при отображении блочных элементов. Может принимать значения `left`, `right`, `center`, `justify`, действие которых соответствует табл. 5.1
`text-decoration`	Задает украшение текста при отображении элементов. Может принимать значения `none` (обычный текст), `underline` (подчеркнутый), `overline` (надчеркнутый), `line-through` (перечеркнутый), `blink` (мерцающий). Значение `blink` не поддерживается Internet Explorer
`text-transform`	Определяет преобразование текста при отображении элементов. Может принимать значения `none` (без преобразования), `capitalize` (делать первую букву каждого слова прописной), `uppercase` (выводить прописными буквами), `lowercase` (выводить строчными буквами)
`letter-spacing`	Задает интервал между буквами при отображении текста. Значение `normal` означает стандартный интервал, абсолютный или относительный размер устанавливает интервал *в дополнение* к стандартному. Это значение может быть отрицательным
`word-spacing`	Задает интервал между словами при отображении текста. Значение `normal` означает стандартный интервал, абсолютный или относительный размер устанавливает интервал *в дополнение* к стандартному. Это значение может быть отрицательным

Таблица 14.5 *(окончание)*

Свойство	Описание
white-space	Определят правила обработки пробелов при отображении блочного элемента. Может принимать значения normal (пробелы и разрывы строк отображаются обычным образом), pre (отображение как в форматированном тексте), nowrap (пробелы отображаются обычным образом, но разрывы строк запрещены)
line-height	Задает высоту строки текста. Для блочного элемента устанавливает минимальную высоту входящих в него строчных блоков, для текстового элемента — точную высоту его строчного блока. Высота не может быть отрицательной и задается значением normal (по умолчанию) или в виде размера
vertical-align	Устанавливает положение элемента по вертикали относительно содержащего его строчного блока. Применимо к текстовым элементам и ячейкам таблиц. Принимает одно из значений: baseline (выравнивание по базовой линии строчного блока), sub (расположение в позиции нижних индексов), super (расположение в позиции верхних индексов), top (выравнивание по верху строчного блока), text-top (выравнивание по верхней линии шрифта строчного блока), middle (выравнивание по центру строчного блока), bottom (выравнивание по низу строчного блока), text-bottom (выравнивание по нижней линии шрифта строчного блока). Не меняет размер шрифта. Указание вместо одного из этих значений абсолютного или относительного размера означает увеличение (для положительного значения) или уменьшение (для отрицательного) отступа по вертикали на заданную величину. Величина в процентах вычисляется от значения line-height

Распространенные браузеры пока не поддерживают свойство text-shadow, создающее тень текста.

Свойства text-decoration и vertical-align не наследуются дочерними элементами, остальные свойства наследуемы. Приведем несколько примеров.

Свойство text-indent идеально подходит для создания "красной строки", принятой в русскоязычных текстах:

```
P { text-indent: 2em }
```

Во многих браузерах принято горизонтальное выравнивание текста по левому краю. Для приведения текста к более "читабельному" виду определим выравнивание по ширине в качестве основного способа для всего документа:

```
body { text-align: justify }
```

Свойства CSS, предназначенные для работы с цветом и фоном элементов, приведены в табл. 14.6.

Таблица 14.6. *Свойства цветов и фона*

Свойство	Описание
`color`	Задает цвет текста, содержащегося в элементе при отображении. Принимает значения наименования или кода цвета (см. *разд. 9.1*)
`background-color`	Устанавливает цвет фона элемента при отображении. Принимает значения наименования или кода цвета. Значение `transparent`, принятое по умолчанию, соответствует прозрачному фону
`background-image`	Определяет графический образ фона элемента при отображении, заданный относительным или абсолютным адресом в виде `url(URL-адрес)`. Значение `none`, принятое по умолчанию, означает отсутствие фонового образа
`background-repeat`	Если определено свойство `background-image`, то задает повтор фонового образа при отображении. Возможные значения: `repeat` (образ повторяется по горизонтали и вертикали), `repeat-x` (образ повторяется только по горизонтали), `repeat-y` (образ повторяется только по вертикали), `no-repeat` (образ не повторяется, отображается одна его копия)
`background-attachment`	Если задано свойство `background-image`, то устанавливает прокрутку фонового образа при отображении. Возможные значения: `scroll` (образ прокручивается вместе с документом), `fixed` (образ зафиксирован и не прокручивается)
`background-position`	Если задано свойство `background-image`, то определяет позицию фонового образа при отображении. Задается в виде `RX RY`, где `RX`, `RY` — положительные или отрицательные размеры, соответствующие горизонтальному и вертикальному смещению образа относительно области заполнения. Вместо значения `RX` может быть указано одно из значений `left`, `center`, `right`, вместо `RY` — одно из значений `top`, `center`, `bottom`
`background`	Является сокращением для остальных свойств фона, позволяя задавать их в одной декларации, например, `table{ background: url("img/bk1.gif") gray`

	50% 50% repeat fixed }

Свойство `color` применимо ко всем элементам и наследуется. Свойства фона также применимы ко всем блочным и текстовым элементам, но не наследуются элементами-потомками.

На рис. 14.3 и в листинге 14.4 приведен пример использования изученных в этом разделе свойств CSS. Открыть этот пример в своем браузере вы можете, запустив документ Glava_14\FontProperties.html с компакт-диска.

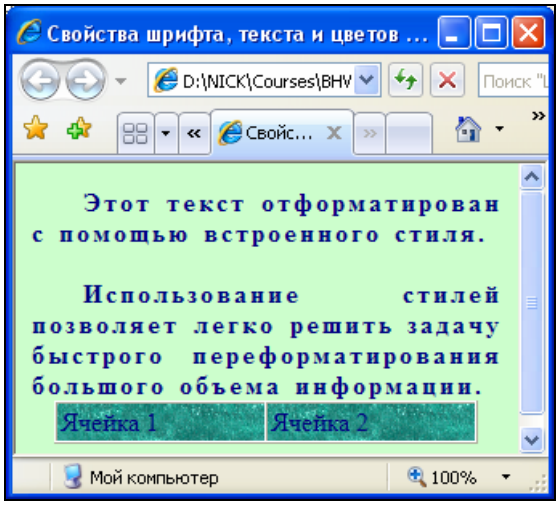

Рис. 14.3. Свойства шрифта, текста и цветов

Листинг 14.4. Свойства шрифта, текста и цветов

```
<html><head>
 <title>Свойства шрифта, текста и цветов</title>
<style type="text/css">
 body { /* основные свойства документа */
  text-indent: 2em;
  text-align: justify;
  letter-spacing: 0.1em;
  word-spacing: 0.1em;
  color: #000080;
  font-weight: 800;
  background-color: #CCFFCC
 }
```

```
 table{ /* свойства фона для таблиц */
  background: url("img/bk1.gif") #CCFFCC left top repeat fixed }
</style></head>
<body>
 <p>Этот текст отформатирован с помощью встроенного стиля.
 <p>Использование стилей позволяет легко решить задачу
 быстрого переформатирования большого объема информации.
<table width=90% align=center border=1 cellpadding=2 cellspacing=0>
<tr><td> Ячейка 1</td><td>
 Ячейка 2 </td></tr></table>
</body></html>
```

14.5. Свойства заполнителей, границ и рамок

Для любого элемента HTML в CSS создается *объемлющий его прямоугольник*, "устройство" которого схематично представлено на рис. 14.4, откуда видно, что помимо содержимого каждый элемент содержит *заполнитель* или *вмещающий блок*, *рамку* и *прозрачную границу*, образующие его внешние слои.

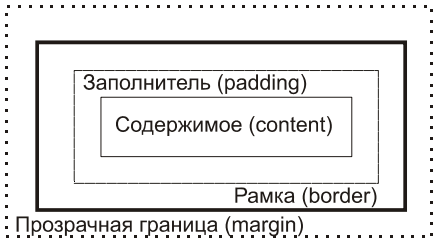

Рис. 14.4. Общая схема элемента HTML в CSS

Каждый из этих четырех вложенных прямоугольников распадается еще на четыре части: левую (`left`), правую (`right`), верхнюю (`top`) и нижнюю (`bottom`). Периметр прямоугольника называется *краем*, так что любой элемент может иметь до четырех краев:

- край содержимого — ограничивает отображаемую область элемента, в которой выводится его содержимое;
- край заполнителя — ограничивает свободное пространство между рамкой и содержимым. Если элемент имеет непрозрачный фон, внешней границей

его нанесения будет внешний край заполнителя. Если ширина заполнителя равна нулю, его край совпадает с краем содержимого;

- край рамки — ограничивает видимую рамку элемента. Если ее ширина равна нулю, то ее край совпадает с краем заполнителя;
- край границы — определяет прозрачные поля объемлющего прямоугольника, расположенные вне рамки. Если ширина границы равна нулю, то ее край совпадает с краем рамки.

Каждый край состоит из четырех сторон-частей.

Свойства границы и заполнителя приведены в табл. 14.7. Все они могут быть заданы в любых размерных единицах или в процентах. Указание в процентах всегда понимается относительно ширины вмещающего блока. Перечисленные свойства не наследуются и применимы ко всем элементам. Размеры границ *могут* быть отрицательными, а размеры заполнителей — нет.

Таблица 14.7. *Свойства границы и заполнителя*

Свойства	Описание

`margin-top`
`margin-right`
`margin-bottom`
`margin-left`
	Размеры верхней, правой, нижней и левой границ объемлющего прямоугольника соответственно

`margin`	Размер всех границ объемлющего прямоугольника одновременно, является сокращением для свойств `margin-top`, `margin-right`, `margin-bottom` и `margin-left`

`padding-top`
`padding-right`
`padding-bottom`
`padding-left`
	Размеры верхнего, правого, нижнего и левого заполнителя объемлющего прямоугольника соответственно

`padding`	Размер всех заполнителей объемлющего прямоугольника одновременно, является сокращением для свойств `padding-top`, `padding-right`, `padding-bottom` и `padding-left`

Приведем примеры работы с этими свойствами.

Для того чтобы браузер при размещении абзацев текста оставлял свободными по 5% ширины окна слева и справа, достаточно написать в стилевом файле определение вида

```
P { margin-left: 5%; margin-right: 5% }
```

По умолчанию в браузерах интервалы между абзацами установлены равными двух-трехкратному интервалу между строками, что порой делает текст неудобочитаемым. Исправим вид текста с помощью стиля:

```
P { margin-top: 4px; margin-bottom: 4px }.
```

Использовать "сокращенные" свойства `margin` и `padding` можно следующим образом:

```
body { margin: 10 px }      /* все границы равны 10 px */
body { margin: 10 px 8 px } /* top, bottom = 10 px, right, left = 8 px */
body {padding: 0,5 em 0.75 em 1 em }
            /* top=0,5 em, right=0,75 em, bottom=1 em, left=0,75 em */
body { margin: 1 em 2 em 3 em 4 em }
            /* top=1 em, right=2 em, bottom=3 em, left=4 em */
```

Основные свойства рамок приведены в табл. 14.8. Они применимы ко всем элементам и никогда не наследуются. Значения свойств рамок нельзя указывать в процентах.

Таблица 14.8. *Свойства рамок*

Свойства	**Описание**

`border-top-width`
`border-right-width`
`border-bottom-width`
`border-left-width`
	Задают размеры верхней, правой, нижней и левой рамок объемлющего прямоугольника соответственно. Указываются в виде фиксированного положительного размера или одним из ключевых слов `thin` (тонкая), `medium` (средняя, принято по умолчанию) или `thick` (толстая)

`border-width`	Сокращение для свойств `border-top-width`, `border-bottom-width`, `border-left-width` и `border-right-width`, позволяет задавать размер всех рамок одновременно

`border-top-color`
`border-right-color`
`border-bottom-color`
`border-left-color`
	Определяют цвета рамки объемлющего прямоугольника, цвет можно указать наименованием или 16-ричным кодом (см. *разд. 9.1*). Значение по умолчанию — цвет генерирующего элемента

`border-color`	Сокращение для свойств `border-top-color`, `border-right-color`, `border-bottom-color` и `border-left-color`, задает цвет всех рамок объемлющего прямоугольника одновременно. Значение `transparent` означает, что рамка должна быть прозрачной, прозрачная рамка при этом может иметь ненулевую ширину

Таблица 14.8 *(окончание)*

Свойства	Описание

`border-top-style`
`border-right-style`
`border-bottom-style`
`border-left-style`
	Задают стиль верхней, правой, нижней и левой рамок объемлющего прямоугольника соответственно. Могут принимать одно из *значений стиля* `none` (нет рамки), `hidden` (рамка скрыта), `dotted` (пунктирная рамка), `dashed` (короткий штрих), `solid` (сплошная линия), `double` (двойная линия), `groove` (трехмерная выемка), `ridge` (трехмерный выступ), `inset` (трехмерная врезка), `outset` (трехмерная вырезка)

`border-style`	Сокращение для свойств `border-top-style`, `border-right-style`, `border-bottom-style` и `border-left-style`, задает стиль всех рамок одновременно. Значением свойства может быть от одного до четырех стилей, стили те же, что для исходных свойств. Один заданный стиль применяется ко всем рамкам, четыре — последовательно к верхней, правой, нижней и левой сторонам, если заданы два или три значения, недостающий стиль принимается равным стилю противоположной рамки

`border-top`
`border-right`
`border-bottom`
`border-left`
	Сокращенные свойства рамки, позволяющие определить размер, стиль и цвет соответствующей рамки одновременно. Если какое-то из свойств не задано, используется его начальное значение

`border`	Устанавливает размер, стиль и цвет всех рамок одновременно. Если какое-то из свойств не задано, используется его начальное значение

Приведем примеры описанных свойств.

```
P {
  border-width: thin thick medium;
  border-color: red green blue;
  border-style: dashed dotted;
 }
```

Здесь верхняя граница абзаца будет тонкой, правая и левая — толстыми, а нижняя — средней по толщине. Верхняя граница выведется красным цветом, правая и левая — зеленым, а нижняя — синим. При этом верхняя и нижняя границы будут выведены штриховыми линиями, а левая и правая — пунктирными.

Посмотреть этот пример в своем браузере и поэкспериментировать с типами линий вы можете, открыв документ Glava_14\Borders1.html с компакт-диска.

Необязательно придавать элементам столь экзотичный вид. Вполне симпатичный блок текста можно получить, определив, например, следующее стилевое оформление абзацев:

```
P {
  border: thin dotted #333333;
  background-color: #CCCCCC; color: black;
  margin: 0.5em; padding: 0.1em
 }
```

Здесь абзац обрамляется тонкой темной пунктирной рамкой, черный текст выводится на светло-сером фоне, дополнительно настроены размеры областей заполнителя и прозрачной границы. Этот пример записан на компакт-диске под именем Glava_14\Borders2.html.

14.6. Позиционирование элементов

CSS позволяет не только задать свойства объемлющих прямоугольников, но и управлять их *взаимным расположением*. Рассмотрим основные средства позиционирования объектов.

Свойство `display` задает *тип объемлющего прямоугольника* для элемента и непосредственно влияет на его отображение. Это свойство применимо ко всем элементам и имеет фиксированный набор значений (табл. 14.9).

***Таблица 14.9.** Основные значения свойства display*

Значение	Описание
`display="block"`	Блочный элемент, отображаемый как отдельный абзац. При его отображении генерируется главный прямоугольник блока, в котором располагаются объемлющие прямоугольники потомков элемента
`display="inline"`	Текстовый элемент, отображаемый как строки внутри текущего абзаца
`display="run-in"`	Присоединяемый элемент. Если следующий за ним элемент блочный, то форматируется как его первый текстовый элемент, иначе как блочный
`display="compact"`	Компактный элемент. Если следующий за ним элемент блочный, то форматируется как однострочный текстовый элемент, по возможности отображаемый на левой или правой границе последующего блока. В противном случае отображается как обычный блочный элемент
`display="none"`	Элемент и все его потомки игнорируются при отображении

Существуют также значения свойства `display` для маркировки элементов таблиц и списков.

Например, задав в документе Glava_14\Borders2.html свойство абзаца `display: inline`, вы получите отображение абзацев в виде обычных текстовых элементов, без разрыва строки.

Свойство `position` задает *схему позиционирования* элемента. Его определенные в CSS значения описаны в табл. 14.10.

Таблица 14.10. *Схемы позиционирования в CSS*

Значение	Описание
`position="static"`	*Статический* элемент, не имеющий специального позиционирования. Значение по умолчанию
`position="relative"`	*Относительно позиционируемый* элемент, для которого объемлющий прямоугольник сначала генерируется в обычном порядке, а затем смещается на величины, заданные свойствами `left` и `top`
`position="absolute"`	*Абсолютно позиционируемый* элемент, положение которого вычисляется относительно позиции ближайшего позиционированного предка на основании свойств `left` и `top`. Кроме позиции элемента, можно задать его размер свойствами `right` и `bottom`. Если все предки не позиционированы, положение вычисляется относительно элемента `body`
`position="fixed"`	*Фиксированный элемент*, который позиционирован абсолютно и дополнительно привязан к некоторой базовой точке. Его позиция зафиксирована относительно окна браузера, т. е. он остается неподвижным при прокрутке окна

Ко всем позиционированным элементам применимы свойства `left`, `top`, `right` и `bottom`, которые задают *относительное смещение* элемента. Смещение может быть задано как фиксированный размер либо указываться в процентах, что приводит к его вычислению относительно высоты или ширины вмещающего блока.

Позиционирование элементов может привести к тому, что на экране они накладываются друг на друга. При этом считается, что элементы могут располагаться в несколько *слоев* относительно оси Z, направленной от монитора на пользователя. Слои образуют *стек*, расположенный по оси Z. Управлять этим стеком легко: чем больше *индекс элемента* в стеке, тем выше он расположен, т. е. лучше видим пользователем. Элементы с одинаковым индексом распола-

гаются по правилу наполнения стека в программировании: чем раньше элемент расположен в тексте документа, тем глубже он находится.

Свойство `z-index` позволяет явно задавать индекс элемента в стеке, т. е. управлять наложением элементов друг на друга. Оно может принимать значения целого положительного числа или `auto` (индекс элемента автоматически принимается равным индексу его родителя).

Пример позиционирования объектов демонстрирует рис. 14.5. Код соответствующего документа содержится в листинге 14.5. На компакт-диске этот пример записан под именем Glava_14\Position1.html.

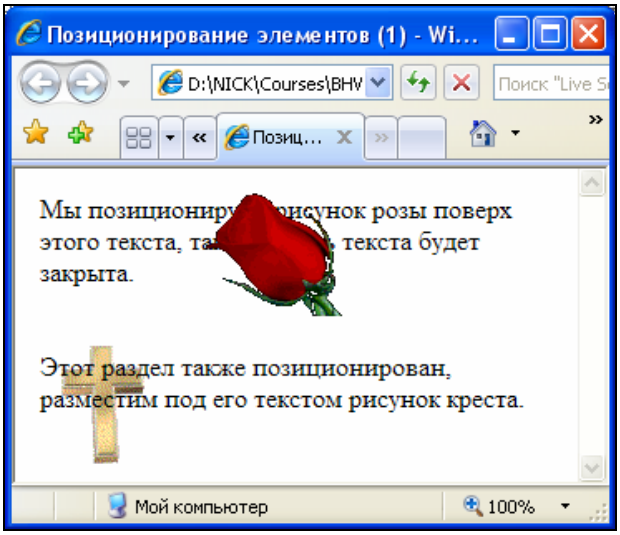

Рис. 14.5. Позиционирование элементов

Листинг 14.5. Позиционирование элементов

```
<html>
<head>
 <title>Позиционирование элементов (1)</title>
<style type="text/css">
 div.before {
  position: relative; left:5px; top: 0;
 }
 .abs1 {
```

```
  position: absolute; left: 100px; top: 0
 }
 .abs2 {
  position: absolute; z-index: -1; left: 10px; top: -5px
 }
</style>
</head><body>
<DIV class="before">Мы позиционируем рисунок розы поверх этого текста, так
что часть текста будет закрыта.
<img src="img/rose.gif" width="82" height="74" class="abs1">
</DIV>
<br><br>
<DIV class="before">Этот раздел также позиционирован, разместим под его
текстом рисунок креста.
<img src="img/cross.gif" width="55" height="75" class="abs2">
</div></body></html>
```

Свойство `float` со значением `left` или `right` указывает, что элемент *плавающий*, и задает его выравнивание влево или вправо. По умолчанию свойство имеет значение `none` (элемент не является плавающим). Оно применимо ко всем элементам, кроме позиционированных.

Плавающий элемент всегда считается блочным, т. е. его свойство `display` игнорируется. Такой элемент смещается влево или вправо до тех пор, пока не встретит рамку, заполнитель или границу другого блочного элемента. Элементы, следующие за плавающим элементом, сдвигаются относительно него и обтекают его справа или слева. Например, использование "плавающих" абзацев — это еще один способ вывести их рядом:

```
<p style="float: left; width: 50%">Абзац 1 - прижат к левому краю.</p>
<p style="float: right; width:50%">Абзац 2 - прижат к правому краю.</p>
```

Пример с плавающими абзацами записан на компакт-диске под именем Glava_14\Float1.html.

Свойство `clear` указывает, как текущий элемент *не может* обтекать предыдущий плавающий объект. Принятое по умолчанию значение `none` разрешает обтекание плавающего элемента с обеих сторон, `left` *запрещает* обтекание слева, `right` — справа, а `both` — с обеих сторон. Запрет обтекания означает, что данный элемент должен отображаться ниже предыдущего плавающего элемента. Корректная поддержка свойства пока реализована не во всех обо-

зревателях. Проверить, как обстоит дело с вашим браузером, вы можете, открыв пример Glava_14\Float2.html с компакт-диска.

В завершение раздела кратко опишем свойства, позволяющие управлять размерами и видимостью элементов на экране.

- `width` — задает ширину содержимого для блочных элементов и элементов, *внешних* по отношению к CSS (например, рисунков). Ширина не может быть отрицательной и задается в виде размера или в процентах от ширины вмещающего блока.
- `height` — аналогичным образом задает высоту содержимого. Напомним, что высота текстовых элементов устанавливается свойством `line-height`.

Свойства ширины и высоты применимы ко всем элементам, кроме текстовых, строк таблиц и групп строк. Эти свойства не наследуются.

- `clip` — задает область *обрезки* содержимого при переполнении. Оно может принимать значение `auto` (область обрезки совпадает с границами элемента) или `rect (<top> <right> <bottom> <left>)`, где `<top>`, `<bottom>`, `<right>` и `<left>` показывают величину отступов от соответствующих сторон блока.

Отрицательные смещения допускаются, например `P { clip: rect(-2px, -2px, 4px, 4px); }`.

- `overflow` — определяет *правила обрезки* содержимого элемента при переполнении. Принимает одно из значений `visible` (содержимое не обрезается), `hidden` (содержимое обрезается, размер и форма области обрезки задается свойством `clip`), `scroll` (содержимое обрезается, браузер должен обеспечить механизм прокрутки), `auto` (правила обрезки определяются браузером и при необходимости обеспечивается прокрутка).

Последние два свойства применимы к блочным, но не текстовым, элементам, не используют размеров в процентах и не наследуются.

- `visibility` — устанавливает *видимость* элемента при отображении. Принимает одно из значений `visible` (элемент видим), `hidden` (невидим).

Применяя и комбинируя изученные в этой главе свойства CSS, можно добиваться самых разнообразных эффектов. Например, несложный код из документа Glava_14\Shadow.html демонстрирует создание заголовка с тенью, полученного наложением друг на друга двух элементов `<h1>`, относящихся к классам с различными свойствами.

Еще одно популярное применение стилей — улучшение внешнего вида форм, не слишком удачно выглядящих в большинстве браузеров.

14.7. DHTML и создание визуальных эффектов

DHTML (Dynamic HTML) — не отдельный язык или формат. По сути дела, это термин для обозначения Web-страниц с динамически изменяющимся содержимым. DHTML построен на трех основных технологиях:

- HTML — средство разметки содержимого;
- CSS — средство стилевого оформления документа;
- язык программирования сценариев JavaScript — средство динамического управления содержимым и обработки событий.

Объектная модель документа (Document Object Model, DOM) связывает эти три технологии воедино, представляя собой *интерфейс прикладного программирования* Web-документов.

Суть модели DOM в том, что она делает все элементы страницы *программируемыми объектами*. С помощью языка сценариев разработчик может получить доступ к любому элементу страницы и поменять любые из его разрешенных к изменению свойств. Таким образом, значение любого атрибута любого тега можно изменять программно, что делает документ действительно динамическим. Кроме того, любое действие пользователя, связанное с навигацией по документу и выполняемое с помощью клавиатуры или мыши, трактуется как *событие*, которое может быть перехвачено и обработано сценарием страницы.

Один из основных принципов DOM — представление документа в виде логической древовидной структуры, о которой мы говорили в *разд. 14.1*, другой важный принцип — представление любых элементов документа в виде объектов, а не просто наборов данных. Соответственно, эти объекты могут обладать изменяемыми *свойствами* и программируемыми *методами* обработки связанных с ними событий.

К сожалению, объем и задачи книги не позволяют нам детально познакомиться с интереснейшими технологиями DHTML, но о самом главном следует сказать несколько слов.

Все объекты страницы образуют иерархию, на вершине которой находится объект `document`. Этот объект, как и все остальные, использует для доступа к нижележащим объектам два набора:

- `all` — содержит ссылки на все объекты, расположенные в иерархии ниже данного;
- `children` — содержит ссылки на все объекты, непосредственно порождаемые данным.

Каждый набор имеет ряд доступных *свойств*, например, свойство `length` определяет общее число элементов в наборе.

В записи имени каждого конкретного элемента имена объектов, наборов и свойств разделяются символом точки.

Если некоторый абзац текста является *вторым* на странице и имеет атрибут `id="t2"`, обратиться к нему можно следующими способами:

```
document.all.t2
document.all['t2']
document.all[1]
```

В последнем случае мы учли, что номер абзаца в тексте нам известен точно (*второй*), а вся нумерация в языке сценариев начинается *с нуля*.

Задать имя элемента HTML можно с помощью не только атрибута `id`, но и `name` там, где он допустим.

В программируемом сценарии можно обращаться к *любым свойствам CSS*, следуя простому правилу: из названий свойств CSS удаляются дефисы, а части имен после дефисов присоединяются к предшествующим частям с прописной буквы. Так, свойство CSS `background-color` доступно из сценария JavaScript как `backgroundColor`, а `border-top-width` как `borderTopWidth`.

Кроме того, язык сценариев позволяет писать подпрограммы-функции, которые, становясь обработчиками событий от элементов форм или тега `<body>`, выполняют любые требуемые расчеты и представляют пользователю их результаты.

Несложный пример, записанный на компакт-диске под именем Glava_14\Position2.html, демонстрирует динамическое позиционирование рисунка, начинающего двигаться сразу же после загрузки страницы в окно браузера.

В версии DHTML, разработанной Microsoft для браузера Internet Explorer, есть элементы, использование которых не требует изучения программирования, но позволяет легко добиться впечатляющих эффектов. Речь идет о *фильтрах*, представляющих собой готовые алгоритмы преобразования содержимого в окне браузера. Фильтры применимы к блочным элементам HTML, не являющимся отдельными окнами, например, `<body>`, `<img>`, `<button>`, `<input>`, `<table>` и `<textarea>`. Обращение к фильтру во всем подобно записи декларации в CSS и имеет общий вид

```
filter: название (параметры).
```

Отдельные параметры, разделенные запятыми, определены в виде

```
название=значение.
```

Если фильтр не имеет параметров, после названия все равно указываются пустые круглые скобки. Допускается перечислять сразу несколько фильтров, в этом случае они задаются в виде списка с пробелом в качестве разделителя. В табл. 14.11 приведены названия и краткие описания основных фильтров.

Таблица 14.11. *Основные фильтры для создания динамических эффектов*

Наименование	Описание
alpha	Управляет уровнем непрозрачности объекта
blendTrans	Изменяет контрастность изображения объекта
blur	Создает эффект размытия изображения
chroma	Делает прозрачными пикселы заданного цвета
dropShadow, shadow	Создают эффект наложения тени
flipH	Отражает объект зеркально по горизонтали
flipV	Отражает объект зеркально по вертикали
glow	Добавляет свечение внешних границ объекта
gray	Отображает объект в оттенках серого цвета
invert	Меняет оттенок, яркость и насыщенность цветов объекта на противоположные
light	Подсвечивает объект заданным цветом
mask	Накладывает на объект маску из пискелов заданного цвета
revealTrans	Показывает или скрывает объект, используя набор эффектов перехода
wave	Создает волнообразный эффект, наложенный на объект
xray	Имитирует рентгеновский снимок объекта

Пример использования фильтра приведен в файле Glava_14\Filter.html на компакт-диске.

Из популярных браузеров лишь последние версии Internet Explorer обеспечивают поддержку фильтров, поэтому применять их следует с осторожностью.