Основы CSS

Лекция №3 (веб-версия: http://goo.gl/My4drO)

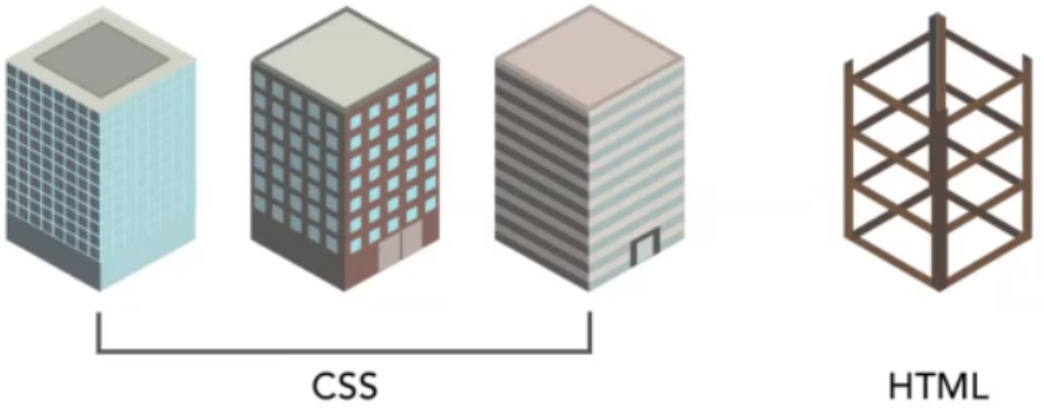
Сегодня будем разбираться с CSS, рассмотрим основные концепции, терминологию и инструменты. Узнаем как оно все работает в браузере. Куда обращаться за информацию, касательно CSS.

Содержание

Общие сведения

CSS — каскадные таблицы стилей (Cascading Style Sheets), технология созданная для того, чтобы контролировать визуальное представление документов, созданных с помощью языков разметки, таких как HTML.

Другими словами можно сказать, что с помощью HTML мы создаем структуру веб-страниц, а с помощью CSS — задаем их внешний вид.



Схематичное представление роли HTML и CSS для веб-страницы

Способы включения в документ

CSS можно ассоциировать с документом двумя способами

- подключив внешний CSS файл (применяется наиболее часто)
- поместив правила в между тегами <style>

еще можно влиять на внешний вид некоторых HTML элементов через их стилевые атрибуты, заметит внимательный студент, но этот способ никак не относится к каскадным таблицам стилей.

Синтаксис

С точки зрения синтаксиса CSS — набор правил, определяющих внешний вид конкретных элементов на странице. Синтаксис CSS чрезвычайно прост:

```
/* CSS комментарий */
div {
    color  : red;
    border : 1px solid #000000;
}
```

Представленный выше фрагмент кода (не считая комментария) представляет способой CSS-правило, с помощью правил и описываются все визуальные свойства элементов. Каждое правило состоит из двух частей селектора и декларации (одной или более), заключенной в фигурные скобки.

Каждая декларация, в свою очередь, состоит из свойства и значения, разделенных `:`, в конце каждой декларации должна стоять `;`. Для различных свойства значения могут иметь разный вид.

Селекторы

Селеторы в CSS нужны для того, чтобы описать некий набор DOM узлов к которым нужно задать определенный набор свойств. Очень подробно обо всех возможных типах селекторов написано тут.

Виды селекторов

селектор	пример	значение
*	*	все элементы
element	div	элементы конкретного типа
.className	.super-duper	элементы обладающие указанным классом
#id	#superDuper	элементы с указанным id
element[attr]	img[alt]	элементы обладающие указанным атрибутом

Комбинации селекторов

связка	пример	значение
	div .super	все элементы с классом .super внутри всех div
,	div, span	все элементы div и все элементы span
>	#super > span	все дочерние элементы 1го уровня внутри элмента с id=”super”
+	#super + div	элемент div, который является следующим сестринским элементом для #super

Псевдоклассы

селектор	пример	значение
:first-child	p:first-child	все элементы p, которые являются первым дочерним элементом своего предка
:hover	a:hover	ссылки в состоянии наведенного курсора мыши
:visited	a:visited	посещенные ссылки

Вес селекторов

Вес селекторов — свойство, на основе которого браузер принимает решение в какой последовательности применить свойства в правилах, если рассматриваемый элемент попадает сразу под несколько из них.

Каждый селектор можно привести к набору из восьми десятичных значений, который однозначно определяет его вес.

Пример:

```
div p .super → 0,0,0,0 0,0,1,2
.logo.big    → 0,0,0,0 0,0,2,0
#head        → 0,0,0,0 0,1,0,0

div{ color: red !important; } → 0,0,0,1 0,0,0,0 /* при определении color */
```

Значение в каждой из позиций (справа налево) соответствуют указанным в селекторе:

1. тегам
2. классам, псевдоклассам, селекторам атрибутов
3. ID
4. inline стили

вторая секция значений отвечает за те же самые виды селекторов, но значения свойства для которых указаны с флагом `!important`.

Отличная, очень подробная статья про расчет веса селекторов:
http://habrahabr.ru/post/137588/

Единицы измерения

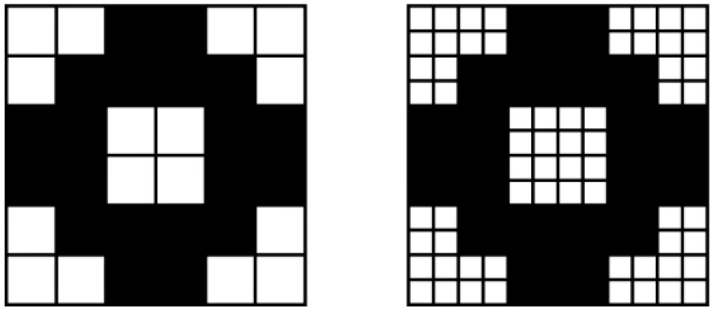
В CSS у нас есть просто гигантский выбор единиц измерения, в которых мы можем указывать значения наших свойств. Речь сейчас пойдет в свойствах, которые означают размер чего-либо.

Делятся они на 2 группы:

- абсолютные (`in, cm, mm, pt`)
  хорошо подходят для работы с утсройствами, где рабочей областью является нечто физическое или явно измеряемое, например принтер
- относительные (`px, em, rem, ex, gd, vw, vh, vm, ch`)
  а вот это уже то, чем вы чаще всего будете пользоваться, т.к. основная часть работы происходит на экранах, для которых физические величины весьма условны

PX

Весьма удивителен факт того, что пиксели принадлежат к относительным величинам, но ответ тут прост — размер пикселя относителен монитора.



1024 x 768 1440 x 900

EM

Эмы это единицы измерения, привязанные к размеру шрифта. По умолчанию это всегда 16px, но нужно иметь в виду, что пользователь может поменять значение в настройках браузера. Поэтому не стоит забывать о том, что неплохо бы задать дефолтный размер шрифта самостоятельно.

Кроме того, если вы задаете в em значения размеров для каких-либо блоков нужно помнить, что эти значения будут вычислены *относительно размера текста этого конкретного блока*.

Пример:

```
div{
    font-size     : 2 em;
    margin-bottom : 1 em;
}
```

В приведенном выше примере размер текста (при стандартных исходных) внутри блока будет равен 32px, а нижний внешний отступ — тоже 32px.

REM (root em)

Данная единица измерения появилась только в CSS3 и позволяет нам избежать неприятных и неочевидных последствий использования обычных em, приведенных в примере выше.

%

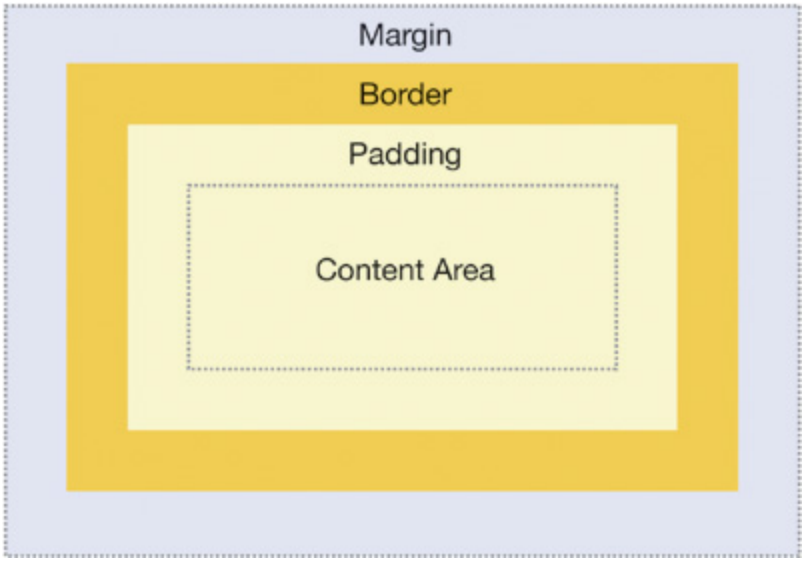
Тут все просто. Процент — это величина равная заданной доли от соответствующего свойства контейнера. Например блок, чья ширина задана в 50% будет занимать половину ширины родителя. Ну и, естественно, при изменении свойств родительского элемента будут меняться и значения у его потомков.

Блочная модель (box model)

Блочная модель — термин, используемый для описания правил, по которым высчитываются геометрические свойства всех элементов дерева отображения на этапах компановки и отрисовки (вспоминаем первую лекцию).

Для каждого узла создается один или более блоков, каждый из которых состоит из:

- области содержимого (content area)
- внутреннего отступа (padding)
- границы (border)
- внешний отступ (margin)



визуализация блочной модели элемента

Для того, чтобы скомпоновать блок, нужно определить 4 его характеристики:

- тип
- схему позиционирования
- размеры
- внешняя информация (размеры изображения, размер экрана)

Тип элемента

Типов элементов бывает всего два:

- блочные
- строчные

Главное их отличие — это их поведение при форматировании: блочные элементы размещаются друг под другом, при этом пытаясь занять собой всю ширину контейнера, а

строчные – друг рядом с другом, занимая только область, которая соответствует расположенному в них контенту.

Схема позиционирования

Схема позиционирования определяется свойствами `position` и `float`. Представлена в трех вариантах:

- **стандартная**
  объект размещается в документе согласно своему положению в дереве отображения и в дереве DOM, а также своему типу и размерам окна.
- **плавающая**
  объект сначала компонуется по стандартной схеме, затем смещается в крайнее правое или крайнее левое положение.
- **абсолютная**
  положение объекта в дереве отображения отличается от его положения в дереве DOM.

Если значение свойства `position` установить в `static` или `relative`, то блок будет соответствовать стандартной схеме, а при значениях `absolute` или `fixed` — абсолютной. Плавающей схеме позиционирования будут соответствовать элементы, для который задано свойство `float`. При стандартной схеме браузером будет использовано положение, рассчитанное по общим правилам, в остальных схемах автор может указать положение с помощью значений `top`, `bottom`, `left` и `right`.

Позиционирование элементов

В зависимости от выбранной схемы позиционирования и конкретного значения свойства `position`. Расположение элемента может быть рассчитано по-разному.

При **относительном позиционировании** (`position:relative`) положение элемента рассчитывается стандартным способом для элемента его типа, а затем сдвигается относительно полученных значений, на величины, указанные в свойствах `top`, `bottom`, `left` и `right`.

Положение элементов, для которых свойство `position` установлено в `absolute` или `fixed`, будет рассчитано особым образом. Положение узлов с таким позиционированием в DOM не имеет значения, оно будет обусловлено свойствами top, `bottom`, `left`, `right` и высчитываться будет относительно контейнера. Для **абсолютно позиционированных** элементов это их контейнер, если он сам позиционирован абсолютно или относительно или область просмотра браузера. Для **фиксировано позиционированных** элементов это всегда область просмотра браузера.

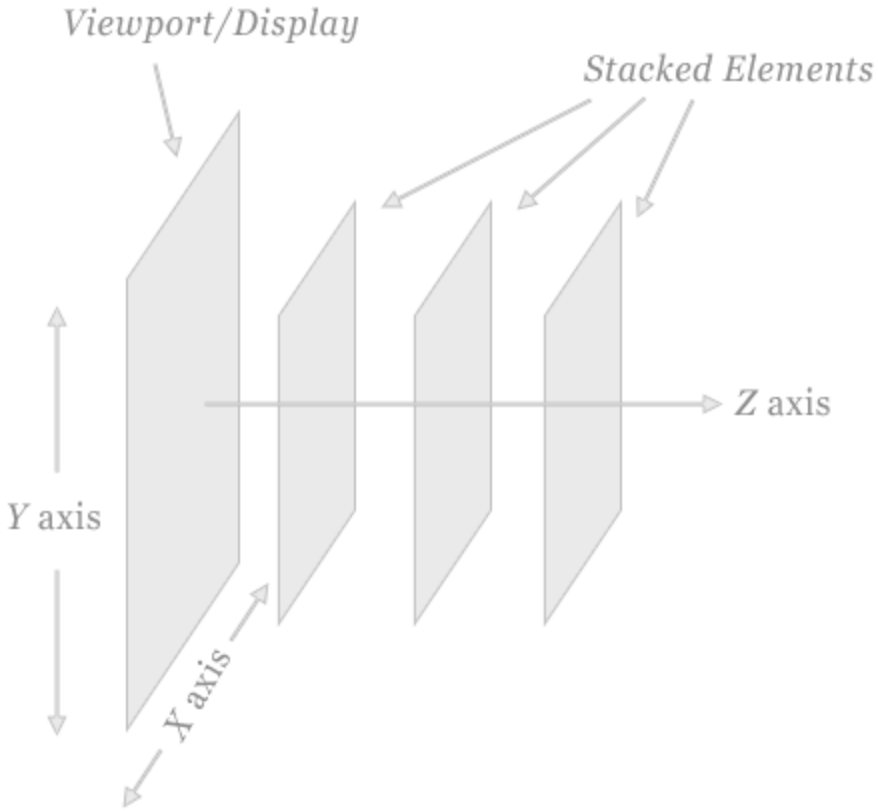
Плавающие элементы будут смещены внутри строки в сторону, указанную в значении float, остальной контент блока будет его обтекать.

Слоевое представление документа

Реализуется эта возможность CSS с помощью свойства `z-index`. По сути оно отражает третье измерение, объем документа. Как работает данное свойство можно заметить только в случае, когда элементы визуально пересекаются, т.е. смещены относительно позиции, определенной нормаальным потоком документа.

Это возможно в 2х случаях:

- для элемента свойтво `position` отлично от `static;`
- для элемента свойство `float` отлично от `none;`



Каждый блочный элемент на странице формирует стек отображения, который выглядит следующим образом:

1. Фон и границы того элемента, в контексте которого и формируется стек. Этот элемент образует так называемый **стековый контекст**. Согласно CSS 2.1 спецификации, такими являются позиционированные элементы, с вычисляемым значением свойства `z-index`, отличным от `auto`. То есть если для позиционированного элемента не установлено свойство `z-index` или если оно установлено явно со значением `auto` (что делать бессмысленно), то этот

элемент не образует новый стековый контекст. Во всех остальных случаях, когда для позиционированного элемента свойство `z-index` задано со значением отличным от auto, то он образует новый стековый контекст. Следующий уровень спецификации — CSS3, допускает образование стекового контекста, путем определения и других свойств элемента, к примеру, `opacity`.

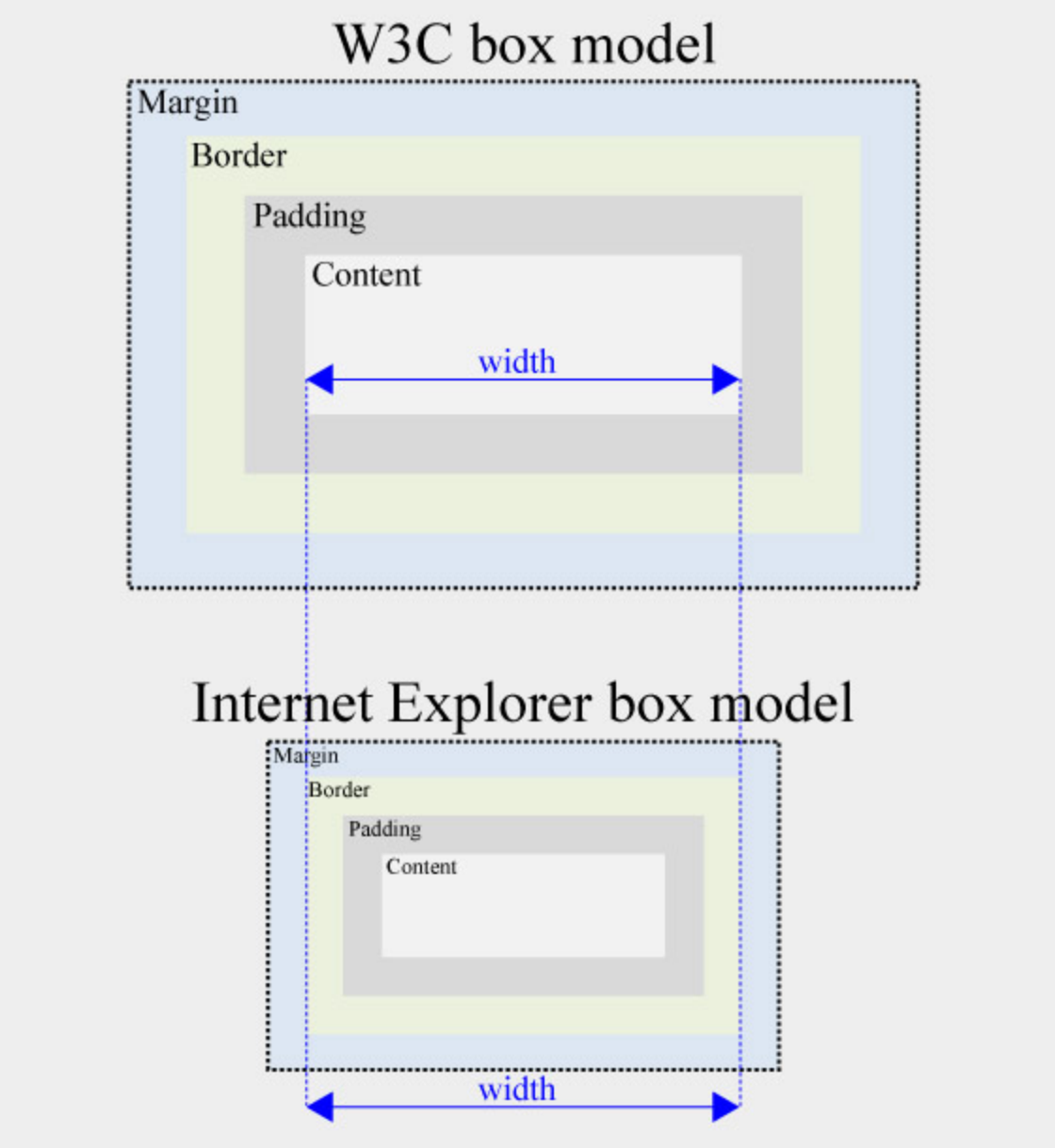
2. Потомки контекстного элемента, образующие новые стековые контексты (условия, которым должны отвечать такие элементы, указаны в предыдущем пункте), имеющие отрицательные значения свойства `z-index`.
3. Не позиционированные поточные потомки контекстного элемента, которые не относятся к элементам строчного уровня.
4. Элементы с установленным свойством `float` и их содержимое.
5. Не позиционированные строчные потомки контекстного элемента.
6. Образующие новые стековые контексты дочерние элементы контекстного контейнера (условия, которым должны удовлетворять такие элементы, указаны в первом пункте) и его позиционированные потомки, значение свойства z-index которых равно нулю.
7. Потомки родительского контекстного элемента, образующие дочерние стековые контексты (удовлетворяющие условиям из первого пункта), имеющие положительные значения свойства `z-index`.

Типы блочной модели

Существует 2 типа блочной моделей:

- W3C
- традиционная (IE)

Отличаются они методом расчета ширины и высоты элемента, на рисунке наглядна показана разница. Дело в том, что в модели W3C, если вы задаете через CSS ширину элемента, то фактический размер элемента будет отличаться (будет больше указанной) на сумму значений padding и border. Таким образом ваша верстка может выглядеть не так, как вы предполагали. Чтобы реальные размеры элемента соответствовали указанным включая отступы и границы нужно рассчитать их по традиционной модели, для этого достаточно установить CSS свойство `box-sizing` в значение `border-box`.



разница блочной модели W3C и IE box model

Стили по умолчанию

Каждый браузер имеет по умолчанию некоторый набор стилей для различных видов элементов. Это нужно для того, чтобы простейшие документы, на которых нет никакого стайлинга тоже можно было нормально читать.

Но с этим есть одна проблема: дефолтные стили разные для каждого браузера. От этого сверстанный вами документ может немного отличаться от браузера к браузеру.

Чтобы избежать такого эффекта нужно обнулить эти дефолтные стили либо самостоятельно, что не лучший вариант, т.к. за вас это уже сделали:

- CSS reset http://meyerweb.com/eric/tools/css/reset/
- CSS normalize http://necolas.github.io/normalize.css/

Идентификация устройств

В CSS у нас есть возсожность применить различные наборы стилей для различных типов устройств, чтобы содержимое наших страниц выглядело более совместемым с конкретным устройством.

Media types

Позволяют выбрать тип устройства, для которого применить тот или иной стиль, поддерживаемых типов устройств довольно много:

- `all`
- `aural`
- `braille`
- `embossed`
- `handheld`
- `print`
- `projection`
- `screen`
- `tty`
- `tv`

Существует три способа как можно указать media type:

@media

```
@media screen{
      // тут правила
}
```

@import

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

link media attribute

```
<link rel="stylesheet" href="print.css" media="print">
```

Media queries

Это такое, скажем, расширение media types (кусок CSS3). Теперь у нас есть возможность писать целые условия вместо обычного типа устройства.

Синтаксис (может быть использован с любым из трех описанных выше способов)

```
[not | only] screen [and] (expression)
```

Примеры

```
@media only screen and (-webkit-min-device-pixel-ratio: 1.5),
       only screen and (min-resolution: 144dpi) { ... }

@media only screen and (min-device-width : 768px)
                   and (max-device-width : 1024px)
                   and (orientation : portrait) {
                       /* Styles or @import */
                   }

@import url(small.css)  (min-width : 400px);

<link rel="stylesheet" media="screen and (min-device-width: 768px) and
(max-device-width: 1024px)" href="css/ipad.css" />
```

Он-лайн инструменты

1. validator.w3.org
2. csslint.net
3. caniuse.com
4. css3generator.com
5. css3please.com
6. css3.me
7. css3maker.com
8. colorzilla.com/gradient-editor/

Использованные материалы, рекомендуемые источники

1. html5rocks.com
2. w3schools.com
3. habrahabr.ru
4. webknowledge.ru