CSS specs

Visual formatting model

贾正权

2017 / 8 / 11

Box sizing

Name:	box-sizing
Value:	content-box | border-box
Initial:	content-box
Applies to:	all elements that accept width or height
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	specified value
Canonical order:	per grammar
Animation type:	discrete

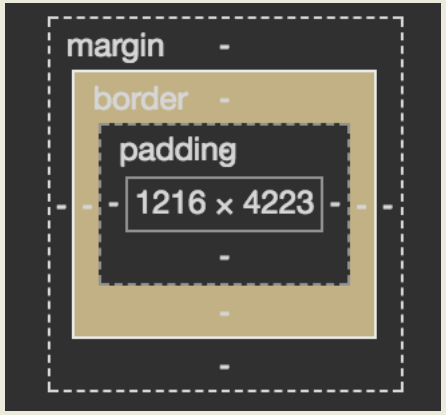
content-box

This is the behavior of width and height as specified by CSS2.1. The specified width and height (and respective min/max properties) apply to the width and height respectively of the content box of the element. The padding and border of the element are laid out and drawn outside the specified width and height.

Box sizing

border-box

Length and percentages values for width and height (and respective min/max properties) on this element determine the border box of the element. That is, any padding or border specified on the element is laid out and drawn inside this specified width and height. The content width and height are calculated by subtracting the border and padding widths of the respective sides from the specified width and height properties.



width

‘border-left’ + ‘padding-left’ + contentWidth + ‘border-left’ + ‘padding-left’

height

‘border-top’ + ‘padding-top’ + contentHeight + ‘border-bottom’ + ‘padding-bottom’

Notice

Bootstrap / Semantic / common.css -> border-box.

Internet Explorer quirks model -> border-box.

Containing block

The position and size of an element‘s box(es) are sometimes calculated relative to a certain rectangle.

- ✓ Initial containing block -> root element.
- ✓ Element‘s position is relative or static -> the nearest block container ancestor box.
- ✓ Element has position: fixed -> viewport .
- ✓ Element has position: absolute -> the nearest ancestor position is not static.

Notice

Ancestor is an inline element and split across multiple lines, the containing block is undefined.

If there is no such ancestor, the containing block is the initial containing block.

Containing block

e.g.

```
1  <html>
2    <body>
3      <p class="a">...</p>
4      <div class="b">
5        <em>text</em>
6      </div>
7    </body>
8  </html>
```

For box generated by	C.B. is established by
html	initial C.B. (UA-dependent)
body	html
p.a	body
div.b	body
em	div.b

If we position “div”:

```
div { position: absolute; }
```

its containing block is no longer "body"; it becomes the initial containing block (since there are no other positioned ancestor boxes).

Controlling box generation

Block-level elements and block boxes:

- ✓ Block-level elements -> ‘display’ property: ‘block’, ‘list-item’, and ‘table’.
- ✓ Block-level box -> each block-level element generates a principal block-level, that participate in a block formatting context.
- ✓ Block container box -> a block container box either contains only block-level boxes or establishes an IFC thus contains only inline-level boxes.

The three terms “block-level box,” “block container box,” and “block box” are sometimes abbreviated as “block” where unambiguous.

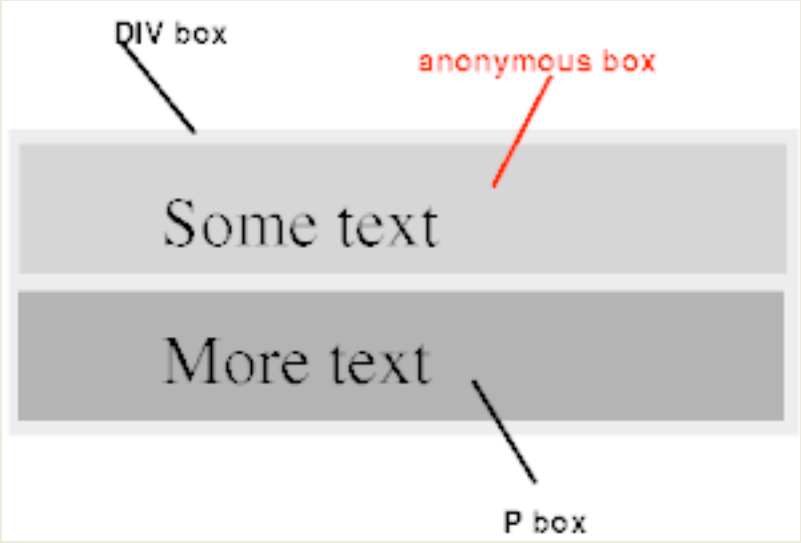
Anonymous block boxes:

If a block container box has a block-level box inside , then we force it to have *only* block-level boxes inside it.

When an inline box contains an in-flow block-level box, the inline box (and its inline ancestors within the same line box) are broken around the block-level box. splitting the inline box into two boxes (even if either side is empty), one on each side of the block-level box(es).

Controlling box generation

e.g.




```
<div>
    Some text
    <p>More text</p>
</div>
```

text
This is the content of SPAN.

```
<p>
text
<span>This is the content of SPAN.</span>
</p>

p    { display: inline;  border:  red solid; }
span { display: block }
```

Controlling box generation

Inline-level elements and inline boxes

Inline-level elements are those elements of the source document that do not form new blocks of content; the content is distributed in lines (e.g., emphasized pieces of text within a paragraph, inline images, etc.).

The following values of the 'display' property make an element inline-level: 'inline', 'inline-table', and 'inline-block'. Inline-level elements generate inline-level boxes, which are boxes that participate in an inline formatting context.

Any text that is directly contained inside a block container element (not inside an inline element) must be treated as an anonymous inline element.

In a document with HTML markup like this:

```
<p>Some <em>emphasized</em> text</p>
```

If it is clear from the context which type of anonymous box is meant, both anonymous inline boxes and anonymous block boxes are simply called anonymous boxes in this specification.

Positioning schemes

In CSS 2.1, a box may be laid out according to three positioning schemes:

- ✓ Normal flow. In CSS 2.1, normal flow includes block formatting of block-level boxes, inline formatting of inline-level boxes, and relative positioningof block-level and inline-level boxes.
- ✓ Floats. In the float model, a box is first laid out according to the normal flow, then taken out of the flow and shifted to the left or right as far as possible. Content may flow along the side of a float.
- ✓ Absolute positioning. In the absolute positioning model, a box is removed from the normal flow entirely (it has no impact on later siblings) and assigned a position with respect to a containing block.

An element is called out of flow if it is floated, absolutely positioned, or is the root element. An element is called in-flow if it is not out-of-flow.

Positioning schemes

'position'

Value:	static | relative | absolute | fixed | inherit
Initial:	static
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	as specified

static -> laid out according to the normal flow.
relative -> box is offset relative to its normal position.
absolute -> offsets with respect to the box's containing block. though absolutely positioned boxes have margins, they do not collapse with any other margins.
fixed -> respect to the viewport

Normal flow

Boxes in the normal flow belong to a BFC or IFC, but not both simultaneously.

Block formatting contexts

Floats, absolutely positioned elements, block containers (such as inline-blocks, table-cells, and table-captions) that are not block boxes, and block boxes with ‘overflow’ other than ‘visible’ (except when that value has been propagated to the viewport) establish new block formatting contexts for their contents. In a block formatting context, each box‘s left outer edge touches the left edge of the containing block (for right-to-left formatting, right edges touch).

Inline formatting contexts

In an inline formatting context, boxes are laid out horizontally, one after the other, beginning at the top of a containing block. Horizontal margins, borders, and padding are respected between these boxes.

When an inline box exceeds the width of a line box, it is split into several boxes and these boxes are distributed across several line boxes. If an inline box cannot be split(disallow word break) then the inline box overflows the line box.

Relative positioning

A relatively positioned box keeps its normal flow size, including line breaks and the space originally reserved for it. A relatively positioned box establishes a new containing block.

- ✓ If both 'left' and 'right' are 'auto' (their initial values), the used values are '0' (i.e., the boxes stay in their original position).
- ✓ If 'left' is 'auto', its used value is minus the value of 'right' (i.e., the boxes move to the left by the value of 'right').
- ✓ If 'right' is specified as 'auto', its used value is minus the value of 'left'.
- ✓ If neither 'left' nor 'right' is 'auto', the position is over-constrained, and one of them has to be ignored. If the 'direction' property of the containing block is 'ltr', the value of 'left' wins and 'right' becomes -'left'. If 'direction' of the containing block is 'rtl', 'right' wins and 'left' is ignored.

Example. The following three rules are equivalent:

```
div.a8 { position: relative; direction: ltr; left: -1em; right: auto }
div.a8 { position: relative; direction: ltr; left: auto; right: 1em }
div.a8 { position: relative; direction: ltr; left: -1em; right: 5em }
```

Floats

A float is a box that is shifted to the left or right on the current line.

'float'

Value:	left | right | none | inherit
Initial:	none
Applies to:	all, but see 9.7
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	as specified

left -> The element generates a block box that is floated to the left. Content flows on the right side of the box, starting at the top (subject to the 'clear' property).
right -> Similar to 'left', except the box is floated to the right, and content flows on the left side of the box, starting at the top.
none -> The box is not floated.

Floats

Here are the precise rules that govern the behavior of floats:

1. The left outer edge of a left-floating box may not be to the left of the left edge of its containing block. An analogous rule holds for right-floating elements.
2. If the current box is left-floating, and there are any left-floating boxes generated by elements earlier in the source document, then for each such earlier box, either the left outer edge of the current box must be to the right of the right outer edge of the earlier box, or its top must be lower than the bottom of the earlier box. Analogous rules hold for right-floating boxes.
3. The right outer edge of a left-floating box may not be to the right of the left outer edge of any right-floating box that is next to it. Analogous rules hold for right-floating elements.
4. A floating box's outer top may not be higher than the top of its containing block. When the float occurs between two collapsing margins, the float is positioned as if it had an otherwise empty anonymous block parent taking part in the flow. The position of such a parent is defined by the rules in the section on margin collapsing.

Floats

Here are the precise rules that govern the behavior of floats:

5. The outer top of a floating box may not be higher than the outer top of any block or floated box generated by an element earlier in the source document.

6. The outer top of an element's floating box may not be higher than the top of any line-box containing a box generated by an element earlier in the source document.

7. A left-floating box that has another left-floating box to its left may not have its right outer edge to the right of its containing block‘s right edge. (Loosely: a left float may not stick out at the right edge, unless it is already as far to the left as possible.) An analogous rule holds for right-floating elements.

8. A floating box must be placed as high as possible.

9. A left-floating box must be put as far to the left as possible, a right-floating box as far to the right as possible. A higher position is preferred over one that is further to the left/right.

Clear float

'clear'

Value:	none | left | right | both | inherit
Initial:	none
Applies to:	block–level elements
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	as specified

left
Requires that the top border edge of the box be below the bottom outer edge of any left-floating boxes that resulted from elements earlier in the source document.

right
Requires that the top border edge of the box be below the bottom outer edge of any right-floating boxes that resulted from elements earlier in the source document.

both
Requires that the top border edge of the box be below the bottom outer edge of any right-floating and left-floating boxes that resulted from elements earlier in the source document.

none
No constraint on the box's position with respect to floats.

Layered presentation

'z-index'

Value:	auto | <integer> | inherit
Initial:	auto
Applies to:	positioned elements
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	as specified

For a positioned box, the 'z-index' property specifies:

1.The stack level of the box in the current stacking context.

2.Whether the box establishes a stacking context.

Within each stacking context, the following layers are painted in back-to-front order:

1.the background and borders of the element forming the stacking context.

2.the child stacking contexts with negative stack levels (most negative first).

3.the in-flow, non-inline-level, non-positioned descendants.

4.the non-positioned floats.

5.the in-flow, inline-level, non-positioned descendants, including inline tables and inline blocks.

6.the child stacking contexts with stack level 0 and the positioned descendants with stack level 0.

7.the child stacking contexts with positive stack levels (least positive first).

Content width

'width'

Value:	<length> | <percentage> | auto | inherit
Initial:	auto
Applies to:	all elements but non-replaced inline elements, table rows, and row groups
Inherited:	no
Percentages:	refer to width of containing block
Media:	visual
Computed value:	the percentage or 'auto' as specified or the absolute length

This property does not apply to non-replaced inline elements, and negative values for 'width' are illegal.

✓ <length>

Specifies the width of the content area using a length unit.

✓ <percentage>

Specifies a percentage width. The percentage is calculated with respect to the width of the generated box's containing block. If the containing block's width depends on this element's width, then the resulting layout is undefined in CSS 2.1.

✓ auto

depends on the values of other properties.

Shrink-to-fit width

Calculation of the shrink-to-fit width is similar to calculating the width of a table cell using the automatic table layout algorithm. Roughly: calculate the preferred width by formatting the content without breaking lines other than where explicit line breaks occur, and also calculate the preferred minimum width, e.g., by trying all possible line breaks. CSS 2.1 does not define the exact algorithm. Thirdly, find the available width: in this case, this is the width of the containing block minus the used values of 'margin-left', 'border-left-width', 'padding-left', 'padding-right', 'border-right-width', 'margin-right', and the widths of any relevant scroll bars.

Then the shrink-to-fit width is: min(max(preferred minimum width, available width), preferred width).

Floating non-replaced elements, absolute positioned, inline-block. If 'width' is computed as 'auto', the used value is the "shrink-to-fit" width.

Line height calculations

The height of a line box is determined as follows:

- ✓ The height of each inline-level box in the line box is calculated. For replaced elements, inline-block elements, and inline-table elements, this is the height of their margin box; for inline boxes, this is their ‘line-height’.
- ✓ The inline-level boxes are aligned vertically according to their 'vertical-align' property. In case they are aligned 'top' or 'bottom', they must be aligned so as to minimize the line box height. If such boxes are tall enough, there are multiple solutions and CSS 2.1 does not define the position of the line box's baseline .
- ✓ The line box height is the distance between the uppermost box top and the lowermost box bottom.

Line height calculations

'line-height'

Value:	normal | <number> | <length> | <percentage> | inherit
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	refer to the font size of the element itself
Media:	visual
Computed value:	for <length> and <percentage> the absolute value; otherwise as specified

normal -> Tells user agents to set the used value to a "reasonable" value based on the font of the element. The value has the same meaning as <number>. We recommend a used value for 'normal' between 1.0 to 1.2. The computed value is 'normal'.

<length> -> The specified length is used in the calculation of the line box height. Negative values are illegal.

<number> -> The used value of the property is this number multiplied by the element's font size. Negative values are illegal. The computed value is the same as the specified value.

<percentage> -> The computed value of the property is this percentage multiplied by the element's computed font size. Negative values are illegal.

Visual effects

'overflow'

Value:	visible | hidden | scroll | auto | inherit
Initial:	visible
Applies to:	block containers
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	as specified

This property specifies whether content of a block container element is clipped when it overflows the element's box.

text-overflow

```
.title {
    display: inline-block;
    overflow: hidden;
    white-space: nowrap;
    text-overflow: ellipsis;
}
```

Visibility

'visibility'

Value:	visible | hidden | collapse | inherit
Initial:	visible
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

The 'visibility' property specifies whether the boxes generated by an element are rendered. Invisible boxes still affect layout (set the 'display' property to 'none' to suppress box generation altogether). Values have the following meanings:

visible
The generated box is visible.
hidden
The generated box is invisible (fully transparent, nothing is drawn), but still affects layout. Furthermore, descendants of the element will be visible if they have 'visibility: visible'.
collapse
If used on elements other than rows, row groups, columns, or column groups, 'collapse' has the same meaning as 'hidden'.

References

- ✓ https://www.w3.org/TR/CSS2/visuren.html
- ✓ https://www.w3.org/TR/CSS2/visudet.html
- ✓ https://www.w3.org/TR/CSS2/visufx.html
- ✓ https://www.w3.org/TR/css-ui-3/#box-sizing

Thanks for watching