Тема 3. Встроенные классы и объекты. DOM

(3.15) Некоторые предописанные классы и предсозданные объекты

Класс String содержит свойства и методы для работы со строками.

Класс Array содержит свойства и методы для работы с массивами.

Класс Date содержит свойства и методы для работы с датами и временными отметками.

Объект Math — предсозданный объект, содержит константы и функции для выполнения математических расчётов.

Класс HTMLElement содержит свойства и методы для работы с содержимым веб-страницы.

(3.25) Предсозданный объект Math

Объект Math — предсозданный объект, содержит методы и константы для выполнения математических расчётов.

Название свойства, метода	Возвращаемое значение, выполняемое действие	Пример использования
свойство **PI**	возвращает значение константы ПИ	`console.log( Math.PI );` **`3.141592653589793`**

метод **abs(значение)**	возвращает абсолютную величину (модуль) *значения*	`console.log( Math.abs(-5.3) );`
		`5.3`
		`console.log( Math.abs(7.2) );`
		`7.2`

метод **max(знач1,знач2)**	возвращает наибольшее из значений *знач1*, *знач2*... (значения должны быть числовыми; можно передавать более двух значений)	`var a=5;`
		`var b=7;`
		`console.log( Math.max(a,b) );`
		`7`

метод **min(знач1,знач2)**	возвращает наименьшее из значений *знач1*, *знач2*... (значения должны быть числовыми; может получать более двух значений)	`var a=5;`
		`console.log( Math.min(a,2,8) );`
		`2`

метод **round(значение)**	возвращает *значение*, округлённое до ближайшего целого числа	`console.log( Math.round(4.7) );`
		`5`
		`console.log( Math.round(4.2) );`
		`4`
		`console.log( Math.round(4) );`
		`4`

метод **ceil(значение)**	возвращает *значение*, округлённое в большую сторону до целого числа	`console.log( Math.ceil(4.7) );`
		`5`
		`console.log( Math.ceil(4.2) );`
		`5`
		`console.log( Math.ceil(-3.2) );`
		`-3`
		`console.log( Math.ceil(4) );`
		`4`

метод **floor(значение)**	возвращает *значение*, округлённое в меньшую сторону до целого числа	`console.log( Math.floor(4.7) );`
		`4`
		`console.log( Math.floor(4.2) );`
		`4`
		`console.log( Math.floor(-3.2) );`
		`-4`
		`console.log( Math.floor(4) );`
		`4`

метод **sqrt(значение)**	возвращает квадратный корень из *значения*	`console.log( Math.sqrt(9) );`
		`3`
		`console.log( Math.sqrt(5) );`
		`2.23606797749979`

метод **random()**	возвращает случайное число от 0 до 1 (не включая 1), при каждом обращении — новое	`console.log( Math.random() );` **`0.7871471339101703`** `console.log( Math.random() );` **`0.9052668876957104`** `console.log( Math.random() );` **`0.2578959968969101`**

Название свойства, метода	Возвращаемое значение, выполняемое действие	Пример использования
метод **sin(угол)**	возвращает синус *угла* (угол указывается в радианах)	`console.log( Math.sin(1) );` **`0.8414709848078965`** `console.log( Math.sin(Math.PI/2) );` **`1`**
метод **cos(угол)**	возвращает косинус *угла* (угол указывается в радианах)	`console.log( Math.cos(1) );` **`0.5403023058681398`** `console.log( Math.cos(Math.PI/2) );` **`6.123233995736766e-17`**
метод **tan(угол)**	возвращает тангенс *угла* (угол указывается в радианах)	`console.log( Math.tan(1) );` **`1.5574077246549023`** `console.log( Math.tan(Math.PI/2) );` **`16331239353195370`**

Полезные коды

Для округления числа не к ближайшей единице, а например к ближайшей сотне или к ближайшей десятой доле, можно применять формулу:

```
Math.round(n/m)*m
```

где n — округляемое число, а m — модуль округления, т.е. то число, на которое должно делиться нацело округляемое n (т.е. например 100 или 0.1).

Для удобства можно написать функцию для такого округления:

```
function roundMod(n,m) {
  return Math.round(n/m)*m;
}
```

Пример использования:

```
console.log( roundMod(15.2,10) );
20
console.log( roundMod(14.8,10) );
10
console.log( roundMod(55.55555,0.01) );
55.56
```

Для получения случайного целого числа в заданном диапазоне (например от 1 до 10), можно применять формулу:

```
Math.floor(Math.random()*(m-n+1))+n
```

где n и m — границы диапазона (т.е. например 1 и 10).

Для удобства можно написать функцию для получения целого случайного числа в заданном диапазоне:

```
function randomDiap(n,m) {
  return Math.floor(
    Math.random()*(m-n+1)
    )+n;
}
```

Пример использования:

```
console.log( randomDiap(1,10) );
8
console.log( randomDiap(1,10) );
8
```

(3.40) Предописанный класс String

Класс String содержит свойства и методы для работы со строками.

Любая строка и любая переменная, содержащая строку, является объектом класса String.

Название свойства, метода	Возвращаемое значение, выполняемое действие	Пример использования

свойство **length**	возвращает длину строки	`var s='Утром деньги, вечером стулья!';`
		`console.log( s.length );`
		`29`

метод **charAt(индекс)**	возвращает символ по указанному *индексу* (т.е. из указанной позиции) в строке (символы в строке индексируются с нуля)	`var s='Утром деньги, вечером стулья!';`
		`console.log( s.charAt(2) );`
		`р`
		`console.log( s.charAt(100) );`

[индекс]	возвращает символ по указанному *индексу* (т.е. из указанной позиции) в строке	`var s='Утром деньги, вечером стулья!';`
		`console.log( s[2] );`
		`р`
		`console.log( s[100] );`
		`undefined`

метод **substr(начало)** **substr(начало,длина)**	возвращает подстроку от указанного индекса *начала* до конца строки, или от указанного индекса указанной *длины*	`var s='Утром деньги, вечером стулья!';`
		`console.log( s.substr(14) );`
		`вечером стулья!`
		`console.log( s.substr(14,7) );`
		`вечером`

метод **slice(начало)** **slice(начало,конец)**	возвращает подстроку от указанного индекса *начала* до конца строки, или от указанного индекса до указанного индекса *конца*, не включая его	`var s='Утром деньги, вечером стулья!';`
		`console.log( s.slice(14) );`
		`вечером стулья!`
		`console.log( s.slice(14,21) );`
		`вечером`

метод **split(разделитель)**	разбивает строку на подстроки, используя указанный *разделитель*; возвращает массив из найденных подстрок; если *разделитель* равен пустой строке, строка разбивается на отдельные буквы	`var s='Утром деньги, вечером стулья!';`
		`console.log( s.split(' ') );`
		`[ 'Утром', 'деньги,', 'вечером', 'стулья!' ]`
		`var s2='Бендер';`
		`console.log( s2.split('') );`
		`[ 'Б', 'е', 'н', 'д', 'е', 'р' ]`

метод **toLowerCase()**	возвращает строку, преобразованную к нижнему регистру	`var s='Утром деньги, вечером стулья!';`
		`console.log( s.toLowerCase() );`
		`утром деньги, вечером стулья!`

метод **toUpperCase()**	возвращает строку, преобразованную к верхнему регистру	`var s='Утром деньги, вечером стулья!';`
		`console.log( s.toUpperCase() );`
		`УТРОМ ДЕНЬГИ, ВЕЧЕРОМ СТУЛЬЯ!`

метод
indexOf(подстр)
indexOf(подстр,нач)
	находит в строке указанную *подстроку*
	и возвращает её позицию; если не найдена — возвращает -1;
	во втором варианте вызова поиск подстроки начинается не с самого начала строки, а с указанной аргументом *нач* позиции.
		`var s='Утром деньги, вечером стулья!';`
		`console.log( s.indexOf('o') );`
		3
		`console.log( s.indexOf('o',4) );`
		19
		`console.log( s.indexOf('o',20) );`
		-1

метод
lastIndexOf(подстр)
lastIndexOf(подстр,нач)
	находит в строке, начиная с конца, указанную *подстроку* и возвращает
	её позицию; если не найдена — возвращает -1;
	во втором варианте вызова поиск подстроки начинается не с самого конца строки,
	а с указанной аргументом *нач* позиции.
		`var s='Утром деньги, вечером стулья!';`
		`console.log( s.lastIndexOf('o') );`
		19
		`console.log( s.lastIndexOf('o',18) );`
		3
		`console.log( s.lastIndexOf('o',2) );`
		-1

метод
replace(старая,новая)
	находит в строке *старую* подстроку и заменяет её на *новую* (если *старая* подстрока встречается несколько раз, будет заменено только самое первое вхождение); возвращает получившуюся строку	`var s='Утром деньги, вечером стулья!';`
		`console.log( s.replace('ром','RO') );`
		УтRO деньги, вечером стулья!

метод
trim()
	удаляет в начале и в конце строки все пробельные символы (пробелы, табуляции, переводы строк); возвращает получившуюся строку	`var s=' \n\r\t\v hello \n\r\t\v ';`
		`console.log( '*' + s.trim() + '*' );`
		hello

Полезные коды

Для замены в строке одной подстроки на другую можно применять
метод строки split(разделитель) в комбинации с методом массива join(разделитель):

```
s.split(s1).join(s2)
```

где s1 — заменяемая подстрока, а s2 — подстрока, на которую нужно заменить.

Пример использования:

```
var r='Изучаем JAVASCRIPT, забываем JAVA!';
console.log( r.split('JAVA') );
[ 'Изучаем ', 'SCRIPT, забываем ', '!' ]
console.log( r.split('JAVA').join('Java') );
Изучаем JavaSCRIPT, забываем Java!
console.log( r.split('JAVA').join('Java').split('SCRIPT').join('Script') );
Изучаем JavaScript, забываем Java!
```

(3.55) Предописанный класс Array

Класс Array содержит свойства и методы для работы с массивами.

Название свойства, метода	Возвращаемое значение, выполняемое действие	Пример использования

свойство **length**	возвращает количество элементов в массиве	`var a=['Opel','BMW'];`
		`console.log( a.length );`
		2

метод **concat(массив)**	возвращает новый массив, полученный путём соединения массива, для которого вызывается метод concat, и *массива*, переданного как аргумент метода concat	`var a=['Opel','BMW'];`
		`var a2=['Lada','Zapor'];`
		`console.log( a.concat(a2) );`
		['Opel', 'BMW', 'Lada', 'Zapor']
		`console.log( a2.concat(a) );`
		['Lada', 'Zapor', 'Opel', 'BMW']

метод **join(разделитель)**	возвращает строку, полученную путём соединения всех элементов массива с *разделителем*	`var a=['Opel','BMW','Lada'];`
		`console.log( a.join(' / ') );`
		Opel / BMW / Lada
		`console.log( a.join('') );`
		OpelBMWLada

метод **push(элемент)**	добавляет *элемент* в конец массива и возвращает новую длину массива; может получать несколько элементов **(этот метод изменяет массив!)**	`var a=['Opel','BMW'];`
		`a.push('Seat','Ford');`
		`console.log( a );`
		['Opel', 'BMW', 'Seat', 'Ford']

метод **pop()**	удаляет последний элемент массива и возвращает удалённый элемент **(этот метод изменяет массив!)**	`var a=['Opel','BMW','Skoda'];`
		`console.log( a.pop() );`
		Skoda
		`console.log( a );`
		['Opel','BMW']

метод **unshift(элемент)**	добавляет *элемент* в начало массива и возвращает новую длину массива; может получать несколько элементов **(этот метод изменяет массив!)**	`var a=['Opel','BMW'];`
		`a.unshift('Seat','Ford');`
		`console.log( a );`
		['Seat', 'Ford', 'Opel', 'BMW']

метод **shift()**	удаляет первый элемент массива и возвращает удалённый элемент **(этот метод изменяет массив!)**	`var a=['Opel','BMW','Skoda'];`
		`console.log( a.shift() );`
		Opel
		`console.log( a );`
		['BMW','Skoda']

метод **slice(начало)**
slice(начало,конец)
	возвращает новый массив, содержащий часть исходного массива от указанного *начального* элемента и до конца массива; или, во втором варианте, до указанного *конечного* элемента; если указанный *конечный* элемент отрицателен — он означает смещение конечного элемента от конца массива, а не от начала	`var a=['A','B','C','D','E','F','G'];`
		`console.log( a.slice(3) );`
		['D', 'E', 'F', 'G']
		`console.log( a.slice(3,5) );`
		['D', 'E']
		`console.log( a.slice(3,-1) );`
		['D', 'E', 'F']

метод **splice(нач,колво,эл1,...)**	удаляет, начиная с указанной *начальной* позиции, указанное *количество* элементов, и возвращает их;
	вставляет, начиная с указанной позиции, новые *элементы* **(этот метод изменяет массив!)**
		`var a=['A','B','C','D','E','F','G'];`
		`console.log( a.splice(2,3) );`
		['C', 'D', 'E']
		`console.log( a );`
		['A', 'B', 'F', 'G']
		`console.log( a.splice(2,0,'q','r') );`
		[ ]
		`console.log( a );`
		['A', 'B', 'q', 'r', 'F', 'G']
		`console.log( a.splice(3,1,'r1','r2') );`
		['r']
		`console.log( a );`
		['A', 'B', 'q', 'r1', 'r2', 'F', 'G']

Название метода	Возвращаемое значение, выполняемое действие	Пример использования

метод **reverse()**	изменяет порядок элементов массива на противоположный **(этот метод изменяет массив!)**	`var a=['Opel','BMW','Skoda'];`
		`a.reverse();`
		`console.log( a );`
		`['Skoda','BMW','Opel']`

метод **sort()** **sort(функция)**	сортирует элементы массива **как строки**;
	во втором варианте — с помощью *функции* можно задать произвольный порядок сортировки **(этот метод изменяет массив!)**
		`var a=['Opel','BMW','Skoda'];`
		`a.sort();`
		`console.log( a );`
		`[ 'BMW', 'Opel', 'Skoda' ]`
		`var n=[7,55,9,22];`
		`n.sort();`
		`console.log( n );`
		`[ 22, 55, 7, 9 ]`

Пример сортировки элементов массива в произвольном порядке

Должна быть описана **функция сравнения**, которая умеет сравнивать два переданных ей элемента массива (например a и b), и возвращать:

- любое отрицательное число — если после сортировки a должно идти раньше b;
- любое положительное число — если после сортировки b должно идти раньше a;
- 0 — если a и b одинаковы.

Пусть у нас есть массив:

```
var m=[
  { fio:'Иванов', zp:1000 },
  { fio:'Петров', zp:800 },
  { fio:'Сидоров', zp:1100 },
  { fio:'Егоров', zp:850 }
];
```

Если мы его хотим отсортировать по возрастанию фамилии, функция сравнения должна быть такой:

```
function compareFIO(a,b) {
  if ( a.fio<b.fio ) return -1;
  if ( a.fio>b.fio ) return 1;
  return 0;
}
m.sort(compareFIO);
console.log( m );
[ {fio:'Егоров', zp:850}, {fio:'Иванов', zp:1000},
{fio:'Петров', zp:800}, {fio:'Сидоров', zp:1100} ]
```

Если мы хотим отсортировать массив по убыванию зарплаты, функция сравнения должна быть такой:

```
function compareZP(a,b) {
  return b.zp-a.zp;
}
m.sort(compareZP);
console.log( m );
[ {fio:'Сидоров', zp:1100}, {fio:'Иванов', zp:1000},
{fio:'Егоров', zp:850}, {fio:'Петров', zp:800} ]
```

(3.62) Document Object Model

DOM — Document Object Model (объектная модель документа) — программный интерфейс, позволяющий программам получить доступ к содержимому HTML-документов (XHTML-документов, XML-документов), а также изменять содержимое, структуру и оформление таких документов.

DHTML — Dynamic HTML — это способ создания интерактивного веб-сайта, при котором JavaScript используется для управления веб-страницей через интерфейс DOM.

```
<html>
  <head>
    <title>Объединение Германии</title>
  </head>
  <body>
    <p>
      Объединение <span class='Country'>Германии</span>
      &mdash; присоединение <span class='Country'>ГДР</span>
      и <span class='Country'>Западного Берлина</span>
      к <span class='Country'>ФРГ</span>.<br>
      Оно состоялось <b>3 октября 1990 года</b>.
    </p>
    <p>
      Первый реальный шаг в сторону объединения
      <span class='Country'>Германии</span> был сделан
      в <b>октябре 1988 года</b>, когда в Москву приехал
      с визитом канцлер <span class='Country'>ФРГ</span> Гельмут Коль.
    </p>
  </body>
</html>
```

(3.64) Поиск элементов в дереве

В дереве элементов можно выполнять поиск элементов.
Элементы могут быть найдены по их идентификаторам (т.е. по значению атрибута id), по классу (т.е. по значению атрибута class), по имени тега, а также по произвольному CSS-селектору.

```
document.getElementById(идентификатор)
```

Ищет в документе элемент с указанным идентификатором.
Возвращает найденный элемент (объект класса HTMLElement), или null, если такой элемент не найден.

```
document.getElementsByClassName(имя_класса)
```

Ищет во всём документе элементы с указанным классом.
Возвращает **массив**, содержащий найденные элементы.

```
document.getElementsByTagName(имя_тега)
```

Ищет во всём документе элементы с указанным именем тега.
Возвращает **массив**, содержащий найденные элементы.

```
document.querySelectorAll(CSS-селектор)
```

Ищет во всём документе элементы, соответствующие указанному CSS-селектору.
Возвращает **массив**, содержащий найденные элементы.

```
document.querySelector(CSS-селектор)
```

Ищет во всём документе первый попавшийся элемент, соответствующий указанному CSS-селектору.
Возвращает найденный элемент, или null, если такой элемент не найден.

Методы getElementsByClassName, getElementsByTagName, querySelectorAll, querySelector могут быть также вызваны не для объекта document, а для любого элемента (т.е. объекта класса HTMLElement). В этом случае поиск элементов выполняется не во всём документе, а среди дочерних элементов данного объекта.

```
элемент.getElementsByClassName(имя_класса)
элемент.getElementsByTagName(имя_тега)
элемент.querySelectorAll(CSS-селектор)
элемент.querySelector(CSS-селектор)
```

(3.66) Установка стилевых свойств элемента

Для элемента можно устанавливать (но не читать) любые стилевые свойства CSS через свойство style:

```
элемент.style.имя_стилевого_свойства=значение
```

Например, установим у элемента с идентификатором EEE красный цвет текста:

```
var elem=document.getElementById('EEE');
elem.style.color='red';
```

Например, найдём все изображения и установим для них рамку синего цвета:

```
var elems=document.getElementsByTagName('img');
for ( var e=0; e<elems.length; e++ ) {
  var elem=elems[e];
  elem.style.borderStyle='solid';
  elem.style.borderColor='blue';
  elem.style.borderWidth='1px';
}
```

Названия стилевых свойств CSS не всегда совпадают с теми что устанавливаются через DOM (это стилевые свойства из CSS2, в CSS3 появляется множество новых):

свойство CSS	свойство DOM
background	background
background-color	backgroundColor
background-image	backgroundImage
background-position	backgroundPosition
background-repeat	backgroundRepeat
border	border
border-bottom	borderBottom
border-bottom-color	borderBottomColor
border-bottom-style	borderBottomStyle
border-bottom-width	borderBottomWidth
border-collapse	borderCollapse
border-color	borderColor
border-left	borderLeft
border-left-color	borderLeftColor
border-left-style	borderLeftStyle
border-left-width	borderLeftWidth
border-right	borderRight
border-right-color	borderRightColor
border-right-style	borderRightStyle
border-right-width	borderRightWidth
border-style	borderStyle
border-top	borderTop
border-top-color	borderTopColor
border-top-style	borderTopStyle
border-top-width	borderTopWidth

свойство CSS	свойство DOM
border-width	borderWidth
clear	clear
color	color
cursor	cursor
display	display
float	**cssFloat / styleFloat**
font	font
font-family	fontFamily
font-size	fontSize
font-style	fontStyle
font-variant	fontVariant
font-weight	fontWeight
height	height
left	left
letter-spacing	letterSpacing
line-height	lineHeight
list-style	listStyle
list-style-image	listStyleImage
list-style-position	listStylePosition
list-style-type	listStyleType
margin	margin
margin-bottom	marginBottom
margin-left	marginLeft
margin-right	marginRight
margin-top	marginTop

свойство CSS	свойство DOM
max-height	maxHeight
max-width	maxWidth
min-height	minHeight
min-width	minWidth
overflow	overflow
padding	padding
padding-bottom	paddingBottom
padding-left	paddingLeft
padding-right	paddingRight
padding-top	paddingTop
position	position
right	right
table-layout	tableLayout
text-align	textAlign
text-decoration	textDecoration
text-indent	textIndent
text-transform	textTransform
top	top
vertical-align	verticalAlign
visibility	visibility
white-space	whiteSpace
width	width
word-spacing	wordSpacing
z-index	zIndex

Для элемента можно устанавливать имя CSS-класса через свойство className:

```
элемент.className=класс
```

Например, установим для элемента с идентификатором EEE CSS-классы SLeft и SGood:

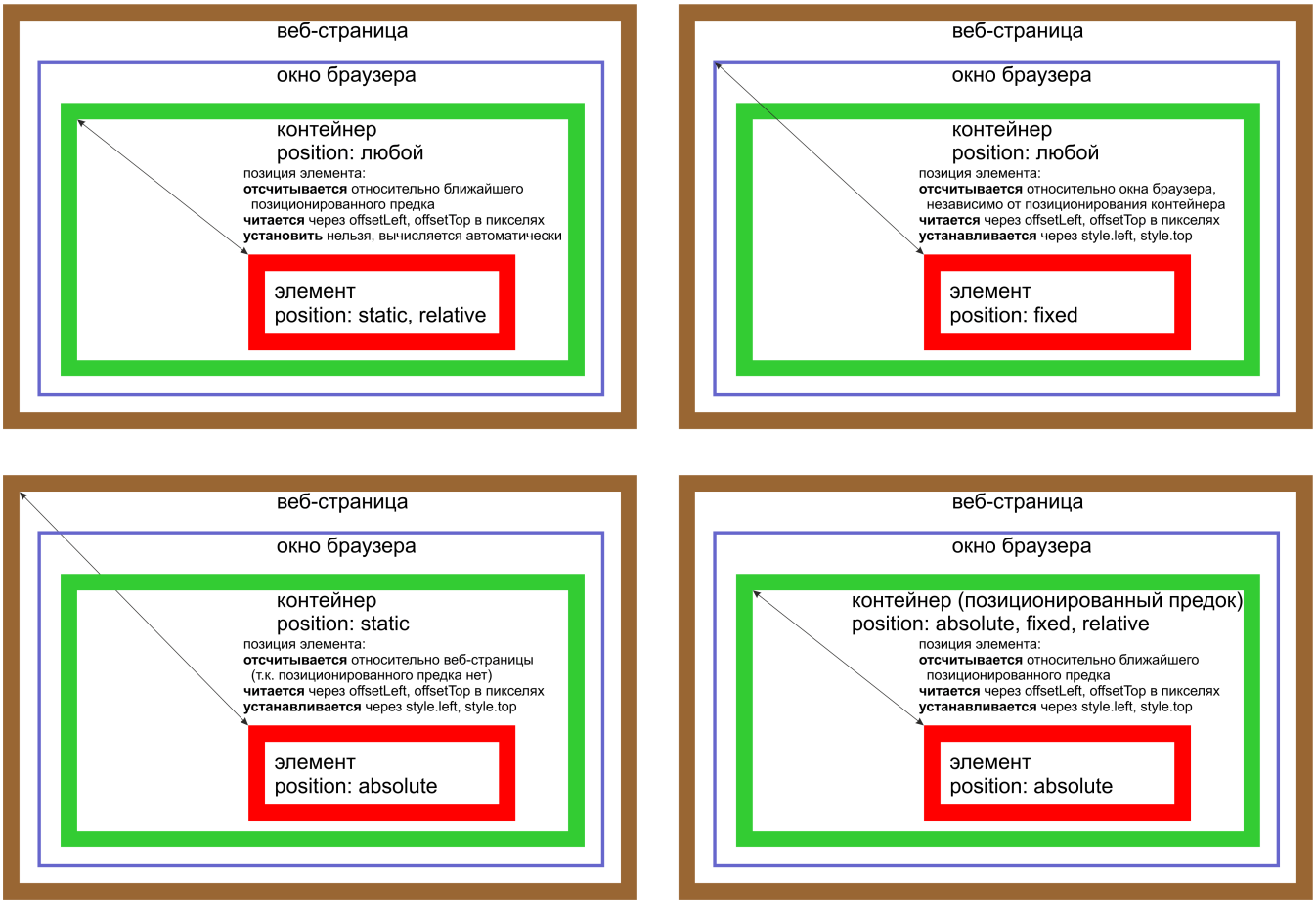
```
var elem=document.getElementById('EEE');
elem.className='SLeft SGood';
```

(3.67) Работа с положением и размером элементов

Каждый элемент имеет свойства, позволяющие узнавать его положение и размер на странице.
Положение элемента есть координаты левого верхнего угла border-box элемента относительно левого верхнего угла content-box его родителя (или относительно документа, окна браузера, ближайшего позиционированного предка — в зависимости от значения стилевого свойства position). Border-box — это прямоугольник элемента, включающий padding и border элемента; content-box — это прямоугольник элемента, не включающий ни padding, ни border элемента.
Изменить положение или **размер** элемента в некоторых случаях можно, устанавливая стилевые свойства style.left, style.top, style.right, style.bottom, style.width, style.height, однако **читать положение** или **размер** элемента следует иначе:

Обращение	Возвращаемое значение
элемент.offsetWidth	Возвращает ширину элемента на экране, в пикселях.
элемент.offsetHeight	Возвращает высоту элемента на экране, в пикселях.
элемент.offsetLeft	Возвращает расстояние на экране от левого края элемента до левого края его контейнера/предка/страницы/окна, в пикселях.
элемент.offsetTop	Возвращает расстояние на экране от верхнего края элемента до верхнего края его контейнера/предка/страницы/окна, в пикселях.

Схема, указывающая точку отсчёта позиции элемента при различных вариантах позиционирования:



Для вычисления положения элемента относительно левого верхнего угла окна браузера (не относительно начала документа!) можно использовать метод элемента getBoundingClientRect, который возвращает хэш со свойствами top, left, right, bottom, а в современных браузерах — также и свойствами width и height.

```
var elem=document.getElementById('proba');
var pos=elem.getBoundingClientRect();
// теперь pos - хэш вида {left:XXX, top:XXX, right:XXX, bottom:XXX}
```

Позицию элемента относительно левого верхнего угла страницы можно получить, добавив к результату getBoundingClientRect текущую прокрутку окна браузера:

```
function getElementPos(elem) {
  var bbox=elem.getBoundingClientRect();
  return {
    left: bbox.left+window.pageXOffset,
    top: bbox.top+window.pageYOffset
  };
}
```

Пример использования:

```
var elem=document.getElementById('proba');
// теперь elem - span со словом 'использования' из текста чуть выше
console.log( elem.offsetLeft );
59
console.log( elem.offsetTop );
320
var pos=getElementPos(elem);
console.log( pos );
{left:77.99768829345703, top:11646.9453125}
```

(3.70) Управление содержимым элемента

Каждый элемент на странице имеет содержимое, которое можно представить в виде HTML-кода. Свойство элемента innerHTML позволяет прочитать имеющееся либо установить новое содержимое элемента, представленное в формате HTML.

```
элемент.innerHTML
```