Валидация тегов

<!-- -->

<!DOCTYPE>

<a>

<abbr>

<acronym>

<address>

<applet>

<area>

<article>

<aside>

<audio>

<b>

<base>

<basefont>

<bdi>

<bdo>

<bgsound>

<big>

<blink>

<blockquote>

<body>

<br>

<button>

<canvas>

<caption>

<center>

<cite>

<code>

<col>

<colgroup>

<command>

<comment>

<datalist>

<dd>

<del>

<details>

<dfn>

<dir>

<div>

<dl>

<dt>

<em>

<embed>

<fieldset>

<figcaption>

<figure>

<font>

<footer>

<form>

<frame>

<frameset>

<h1>

<h2>

<h3>

<h4>

<h5>

<h6>

<head>

<header>

<hgroup>

<hr>

<html>

<i>

<iframe>

<img>

<input>

<ins>

<isindex>

<kbd>

<keygen>

<label>

<legend>

<li>

<link>

<listing>

<main>

<map>

<mark>

<marquee>

<menu>

<meta>

<meter>

<multicol>

<nav>

<nobr>

<noembed>

<noframes>

<noscript>

<object>

<ol>

<optgroup>

<option>

<output>

<p>

<param>

<plaintext>

<pre>

<progress>

<q>

<rp>

<rt>

<ruby>

<s>

<samp>

<script>

<section>

<select>

<small>

<source>

<spacer>

<span>

<strike>

<strong>

<style>

<sub>

<summary>

<sup>

<table>

<tbody>

<td>

<textarea>

<tfoot>

<th>

<thead>

<time>

<title>

<tr>

<track>

<tt>

<u>

<ul>

<var>

<video>

<wbr>

<xmp>

Атрибуты тегов

Универсальные

События

Значения

Скрипт

Цвет

Названия цветов

Коды языков

MIME-типы

www.krediidikeskus.ee

Типы тегов

HTML5

Блочные элементы

Строчные элементы

Универсальные элементы

Нестандартные теги

Осуждаемые теги

Видео

Документ

Звук

Изображения

Объекты

Скрипты

Списки

Ссылки

Таблицы

Текст

Форматирование

Формы

Фреймы

**Тег <canvas>**

**﻿**

Описание

Создает область, в которой при помощи JavaScript можно рисовать разные объекты, выводить изображения, трансформировать их и менять свойства. При помощи тега <canvas> можно создавать рисунки, анимацию, игры и др.

Синтаксис

<canvas id="идентификатор">

</canvas>

Атрибуты

height

Задает высоту холста. По умолчанию 300 пикселов.

width

Задает ширину холста. По умолчанию 150 пикселов.

Также для этого тега доступны универсальные атрибуты и события.

Закрывающий тег

Обязателен.

Пример

HTML5IE 8IE 9CrOpSaFx

<!DOCTYPE html>

<html>

<head>

<title>canvas</title>

<meta charset="utf-8">

<script>

window.onload = function() {

var drawingCanvas = document.getElementById('smile');

if(drawingCanvas && drawingCanvas.getContext) {

var context = drawingCanvas.getContext('2d');

// Рисуем окружность

context.strokeStyle = "#000";

context.fillStyle = "#fc0";

context.beginPath();

context.arc(100,100,50,0,Math.PI\*2,true);

context.closePath();

context.stroke();

context.fill();

// Рисуем левый глаз

context.fillStyle = "#fff";

context.beginPath();

context.arc(84,90,8,0,Math.PI\*2,true);

context.closePath();

context.stroke();

context.fill();

// Рисуем правый глаз

context.beginPath();

context.arc(116,90,8,0,Math.PI\*2,true);

context.closePath();

context.stroke();

context.fill();

// Рисуем рот

context.beginPath();

context.moveTo(70,115);

context.quadraticCurveTo(100,130,130,115);

context.quadraticCurveTo(100,150,70,115);

context.closePath();

context.stroke();

context.fill();

}

}

</script>

</head>

<body>

<canvas id="smile" width="200" height="200">

<p>Ваш браузер не поддерживает рисование.</p>

</canvas>

</body>

</html>

Результат примера в браузере Opera показан на рис. 1.

Рис. 1. Вывод рисунка с помощью тега <canvas>

HTML5

TM Feed

Хабрахабр

Geektimes

Тостер

Мой круг

Фрилансим

Мегапосты: Uremont за 2 млн $

Конкурс SAP-кодеров

Публикации

Пользователи

Хабы

Компании

Песочница

**CANVAS шаг за шагом: Основы из песочницы**

Если верить англо-русскому словарю, то можно узнать что canvas переводится как холст, а если верить википедии, то можно узнать что тег canvas, это элемент HTML 5, который предназначен для создания растрового изображения при помощи JavaScript. Тому как создать это растровое изображение и будет посвящен мой небольшой текст. Прежде чем начинать пробовать свои силы в этом не легком деле рекомендуется уже иметь базовые знания о том что такое HTML и с чем едят JavaScript.

Предварительная «настройка» нашего холста

У нашего подопытного тега есть всего два атрибута — height и width, высота и ширина соответственно, по умолчанию размер холста 150х300 пикселей.

Стоит отметить что canvas создает область фиксированного размера содержимым которого управляют контексты.

Элементарный пример:

<!doctype html>

<html>

<head>

<title>canvasExample</title>

<meta charset='utf-8' />

</head>

<body>

<canvas height='320' width='480' id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById("example"),

ctx = example.getContext('2d');

ctx.fillRect(0, 0, example.width, example.height);

</script>

</body>

</html>

Если сохранить эти несчастные 13 строк в файл и открыть его браузером, то можно будет увидеть область с чёрным прямоугольником, так вот это и есть тот самый холст, на котором нарисован прямоугольник размеры которого равны размерам canvas'а.

Прямоугольники

Самой элементарной фигурой которую можно рисовать является прямоугольник. Предусмотрено три функции для отрисовки прямоугольников.

strokeRect(x, y, ширина, высота) // Рисует прямоугольник

fillRect(x, y, ширина, высота) // Рисует закрашенный прямоугольник

clearRect(x, y, ширина, высота) // Очищает область на холсте размер с прямоугольник заданного размера

Пример иллюстрирующий работу этих функций:

<!doctype html>

<html>

<head>

<title>rectExample</title>

<meta charset='utf-8' />

</head>

<body>

<canvas id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById("example"),

ctx = example.getContext('2d');

example.width = 640;

example.height = 480;

ctx.strokeRect(15, 15, 266, 266);

ctx.strokeRect(18, 18, 260, 260);

ctx.fillRect(20, 20, 256, 256);

for (i = 0; i < 8; i += 2)

for (j = 0; j < 8; j += 2) {

ctx.clearRect(20 + i \* 32, 20 + j \* 32, 32, 32);

ctx.clearRect(20 + (i + 1) \* 32, 20 + (j + 1) \* 32, 32, 32);

}

</script>

</body>

</html>

А теперь краткий построчный разбор:

в строках 10 и 11 мы изменили размер холста — чтоб бы задуманное нами изображение полностью отобразилось,

в строках 12 и 13 мы нарисовали два не закрашенных прямоугольника которые будут символизировать своеобразную рамку нашей «шахматной доски»,

в строке 14 отрисовываем закрашенный прямоугольник размеры которого бы позволил вместить в себе 64 квадрата с шириной стороны 32 пикселя,

в строках с 15 по 19 у нас работает два цикла которые очищают на чёрном прямоугольнике квадратные области в таком порядке что бы в итоге полученное изображение было похоже на шахматную доску

Линии и дуги

Рисование фигур составленных из линий выполняется последовательно в несколько шагов:

beginPath()

closePath()

stroke()

fill()

beginPath используется что бы «начать» серию действий описывающих отрисовку фигуры. Каждый новый вызов этого метода сбрасывает все действия предыдущего и начинает «рисовать» занова.

closePath является не обязательным действием и по сути оно пытается завершить рисование проведя линию от текущей позиции к позиции с которой начали рисовать.

Завершающий шаг это вызовом метода stroke или fill. Собственно первый обводит фигуру линиями, а второй заливает фигуру сплошным цветом.

Те кто когда-то на школьных 486х в былые годы рисовал в бейсике домик, забор и деревце по задумке учителя тот сразу поймёт часть ниже. Итак, существуют такие методы как,

moveTo(x, y) // перемещает "курсор" в позицию x, y и делает её текущей

lineTo(x, y) // ведёт линию из текущей позиции в указанную, и делает в последствии указанную текущей

arc(x, y, radius, startAngle, endAngle, anticlockwise) // рисование дуги, где x и y центр окружности, далее начальный и конечный угол, последний параметр указывает направление

Пример ниже показывает действие всего описанного выше:

<!doctype html>

<html>

<head>

<title>pathExample</title>

<meta charset='utf-8' />

</head>

<body>

<canvas id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById("example"),

ctx = example.getContext('2d');

example.height = 480;

example.width = 640;

ctx.beginPath();

ctx.arc(80, 100, 56, 3/4 \* Math.PI, 1/4 \* Math.PI, true);

ctx.fill(); // \*14

ctx.moveTo(40, 140);

ctx.lineTo(20, 40);

ctx.lineTo(60, 100);

ctx.lineTo(80, 20);

ctx.lineTo(100, 100);

ctx.lineTo(140, 40);

ctx.lineTo(120, 140);

ctx.stroke(); // \*22

</script>

</body>

</html>

В строке 14 заливается цветом дуга, в строке 22 обводится контур нашей короны.

Кривые Бернштейна-Безье

Что такое кривые Безье я думаю лучше объяснит Википедия.

Нам доступно две функции, для построения кубической кривой Бизье и квадратичной, соотвестствено:

quadraticCurveTo(Px, Py, x, y)

bezierCurveTo(P1x, P1y, P2x, P2y, x, y)

x и y это точки в которые необходимо перейти, а координаты P(Px, Py) в квадратичной кривой это дополнительные точки которые используются для построения кривой. В кубическо кривой соответственно две дополнительные точки.

Пример двух кривых:

<!doctype html>

<html>

<head>

<title>curveExample</title>

<meta charset='utf-8' />

</head>

<body>

<canvas id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById("example"),

ctx = example.getContext('2d');

example.height = 480;

example.width = 640;

ctx.beginPath();

ctx.moveTo(10, 15);

ctx.bezierCurveTo(75, 55, 175, 20, 250, 15);

ctx.moveTo(10, 15);

ctx.quadraticCurveTo(100, 100, 250, 15);

ctx.stroke();

</script>

</body>

</html>

Добавим цвета

Что бы наше изображение было не только двух цветов, а любого цвета предусмотрено, два свойства

fillStyle = color // определяет цвет заливки

strokeStyle = color // цвет линий цвет задается точно так же как и css, на примере все четыре способа задания цвета

Цвет задается точно так же как и css, на примере все четыре способа задания цвета

// все четыре строки задают оранжевый цвет заливки

ctx.fillStyle = "orange";

ctx.fillStyle = "#FFA500";

ctx.fillStyle = "rgb(255,165,0)";

ctx.fillStyle = "rgba(255,165,0,1)"

Аналогично задаётся и цвет для линий.

Возьмём пример с шахматной доской и добавим в него немного цвета:

<!doctype html>

<html>

<head>

<title>rectExample</title>

<meta charset='utf-8' />

</head>

<body>

<canvas id='example'>Обновите браузер</canvas>

<script>

var example = document.getElementById("example"),

ctx = example.getContext('2d');

example.height = 480;

example.width = 640;

ctx.strokeStyle = '#B70A02'; // меняем цвет рамки

ctx.strokeRect(15, 15, 266, 266);

ctx.strokeRect(18, 18, 260, 260);

ctx.fillStyle = '#AF5200'; // меняем цвет клеток

ctx.fillRect(20, 20, 256, 256);

for (i = 0; i < 8; i += 2)

for (j = 0; j < 8; j += 2) {

ctx.clearRect(20 + i \* 32, 20 + j \* 32, 32, 32);

ctx.clearRect(20 + (i + 1) \* 32, 20 + (j + 1) \* 32, 32, 32);

}

</script>

</body>

</html>

Задача

Что бы усвоить информацию и закрепить прочитанное на практике я всегда ставлю перед собой не большую цель которая бы одновременно охватывала всё прочитанное и одновременно процесс достижения которой было бы интересен мне самому. В данном случае я попытаюсь отрисовать уровень одной из моих самых любимых в детстве игр. Собственно за не имением времени — добавлять жизнь на него я не буду, а сделаю максимально понятный код охватывающий практически всё то что сегодня здесь описал.

Я воспроизвел один из уровней игры BattleCity известную у нас как Танчики, а вот и ссылка на pastebin на случай если не будет откликаться дропбокс.

На последок комментарий по примеру. В спецификациях картинки которую может выдавать Денди разрешение экрана должно быть 256×240 пикселей.

Поле боя в общеизвестнных Танчиках размером 13х13 больших блоков. Каждый из которых нарисован 4мя повторяющимися спрайтами (коих по общему подсчёту выходит на карте 26х26=676). Итак прикинем как было в оригинале по пикселам и как это правильно масштабировать. Если поделить 240 на 26 то выйдет что целая часть от деления будет 8. Получается что размерность текстуры была 8х8 пиксела т.е. размер поля боя 208х208, а большого блока 16х16. Ширина должна быть 256 пикселов. Сейчас вычислим размер правого столбца с дополнительной информацией и размер полей сверху/снизу. Справа если присмотреться ширина составляет размерность в два блока, итого 2\*16=32. У нас уже 32+208=240 слева поле 16, а снизу и сверху соответственно так же по 16 пикселов. Собственно в моём примере размерность большого блока заключена в переменной cellSize, собственно все вычисления делаются иходя из её размеров. Можете по экспериментировать и поменять её значение, настоятельно рекомендую делать его кратным степеням двойки (16, 32, 64, 128...), если хотите чтоб всё выглядело так как на старом добром денди то устанавливайте её значение равным 16. Хотя и при любых других значениях всё выглядит нормально. Если то как я пишу понравится кому-то кроме меня, то напишу продолжение, а то что в нём будет пока утаю

Смотрите в следующей серии: CANVAS шаг за шагом: изображения