# CSS Grid

### Voorbeeld

### Te gebruiken bij:

-

### Eigenschappen:

### Info

CSS is ongeveer net zo oud als de eerste webpagina's op het web. We hebben het altijd gebruikt om onze pagina's van styling te voorzien, maar het heeft nooit echt handig gewerkt als het op layouts aankwam. Eerst gebruikten we tabellen, daarna *floats*, toen kwam positionering en daarna inline-block.

Flexbox was hierin het eerste systeem dat echt *werkte*, en is een fantastische tool om layouts mee te bouwen. Het is echter nog steeds ééndimensionaal: items staan naast- of onder elkaar, maar kunnen niet overlappen. Daarom is **CSS Grid Layout** in het leven geroepen: een tweedimensionaal raster-systeem.

Laten we beginnen met de HTML. We hebben, wederom, een container-element nodig dat vier kleinere items omwikkelt:

<div>

<section class="item item-a">A</section>

<section class="item item-b">B</section>

<section class="item item-c">C</section>

<section class="item item-d">D</section>

</div>

Omdat we alle items dezelfde achtergrond-kleur en font-kleur willen geven, krijgen ze allemaal dezelfde "item" class. Daarnaast krijgen ze ieder een eigen class-naam, omdat we ze later ook nog individueel moeten kunnen aanspreken in onze grid. Laten we direct wat styling toevoegen om de boel overzichtelijk te houden:

.item {

background-color: #5B052B;

padding: 20px;

}

Onze *soon-to-be-grid* ziet er nu uit als één blok, omdat alle elementen grenzen aan elkaar:

Afbeelding met schermopname, Rechthoek, ontwerp

Automatisch gegenereerde beschrijving

Om te beginnen zullen we de container moeten omtoveren tot grid-container. Dit gaat verrassend makkelijk, met de display-property:

div {

display: grid;

}

Afbeelding met schermopname, tekst, ontwerp

Automatisch gegenereerde beschrijving

Het lijkt nu wellicht alsof al onze elementen er vandoor zijn gegaan, maar in werkelijkheid staan ze allemaal op een hoopje in de rechterbovenhoek. Dat komt omdat wij hebben gezegd dat ze vanaf nu in een raster staan, zonder te zeggen wáár ze dan moeten staan.

Dit doen we met de grid-area-property:

.item-a {

grid-area: a;

}

.item-b {

grid-area: b;

}

.item-c {

grid-area: c;

}

.item-d {

grid-area: d;

}

Voordat we ze kunnen toewijzen, moeten we het daadwerkelijke raster opstellen. Zoals je kon zien wilden we drie kolommen, en twee rijen. Laten we voor het gemak zeggen dat iedere kolom 140 pixels breed is, en iedere rij 140 pixels hoog. Om dit te declareren, gebruiken we de grid-template-columns en grid-template-rows property voor. De property grid-gap zal zorgen voor ruimte *tussen* de items.

div {

display: grid;

grid-template-columns: 140px 140px 140px;

grid-template-rows: 140px 140px;

grid-gap: 16px;

}

Afbeelding met roze, schermopname, Lila, Magenta

Automatisch gegenereerde beschrijving

Gelukkig is alles wat er nu gebeurt heel logisch. We hebben onze items zojuist een naam gegeven, beschreven hoeveel kolommen en hoeveel rijen we hebben en wat hun afmetingen zijn.

Maar we hebben nog steeds niet gezegd wáár de items in die kolommen en rijen thuishoren.

We zullen grid-area-benamingen van de items gebruiken om ze toe te wijzen in ons raster. Hiermee gaan we beschrijven over hoeveel rijen en kolommen het item moet vallen en wat de volgorde zal zijn.

In onze eerste rij moet *A* 2/3 van de rij in beslag nemen, en *B* 1/3. Op de tweede rij moet *C* 1/3, *D* 1/3 en *B* opnieuw 1/3 van de kolommen in beslag nemen:

div {

display: grid;

grid-template-columns: 140px 140px 140px;

grid-template-rows: 140px 140px;

grid-gap: 16px;

grid-template-areas:

"a a b"

"c d b";

}

Nu gedraagt het raster zich zoals we verwachten. Zie je hoe de aanhalingstekens iedere rij omwikkelen.

Afbeelding met tekst, Lettertype, roze, schermopname

Automatisch gegenereerde beschrijving

#### Java

In de context van programmeren verwijst "grid" meestal naar een tweedimensionale datastructuur die bestaat uit rijen en kolommen. Het wordt vaak gebruikt om gegevens georganiseerd op te slaan en te verwerken, vooral in toepassingen zoals grafische weergave, spelontwikkeling en algoritmen voor routeplanning.

Een grid kan worden voorgesteld als een matrix, waarbij elk element in de matrix overeenkomt met een specifieke locatie in het grid. Elk element kan worden geïdentificeerd door zijn rij- en kolomindex.

Bijvoorbeeld, in een 3x3 grid kunnen de elementen worden geïdentificeerd als (rij, kolom):

css

rijen

(0,0) (0,1) (0,2) k

(1,0) (1,1) (1,2) o

(2,0) (2,1) (2,2) l

o

m

Elk element in het grid kan een bepaalde waarde bevatten, zoals een getal, een karakter, een object, enzovoort, afhankelijk van het specifieke gebruiksscenario.

Het gebruik van een grid biedt verschillende mogelijkheden. Hier zijn enkele veelvoorkomende operaties die kunnen worden uitgevoerd op een grid:

Waarde toewijzen: U kunt een specifieke waarde toewijzen aan een bepaald element in het grid door de rij- en kolomindex te gebruiken. Bijvoorbeeld, grid[1][2] = 5; wijst de waarde 5 toe aan het element op rij 1, kolom 2.

Waarde ophalen: U kunt de waarde van een specifiek element in het grid ophalen door de rij- en kolomindex te gebruiken. Bijvoorbeeld,

int value = grid[0][1]; haalt de waarde op van het element op rij 0, kolom 1 en slaat het op in de variabele value.

Itereren over het grid: U kunt door alle elementen in het grid itereren met behulp van lussen zoals for of while. Dit stelt u in staat om bewerkingen uit te voeren op elk individueel element of om het grid te doorzoeken op specifieke patronen.

java

public class GridExample {

private int rows;

private int columns;

private int[][] grid;

public GridExample(int rows, int columns) {

this.rows = rows;

this.columns = columns;

grid = new int[rows][columns];

}

public void setValue(int row, int column, int value) {

grid[row][column] = value;

}

public int getValue(int row, int column) {

return grid[row][column];

}

public void printGrid() {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

System.out.print(grid[i][j] + " ");

}

System.out.println();

}

}

public static void main(String[] args) {

GridExample gridExample = new GridExample(3, 3);

// Waarden toewijzen aan het grid

gridExample.setValue(0, 0, 1);

gridExample.setValue(0, 1, 2);

gridExample.setValue(0, 2, 3);

gridExample.setValue(1, 0, 4);

gridExample.setValue(1, 1, 5);

gridExample.setValue(1, 2, 6);

gridExample.setValue(2, 0, 7);

gridExample.setValue(2, 1, 8);

gridExample.setValue(2, 2, 9);

// Grid afdrukken

gridExample.printGrid();

}

}

Dit voorbeeld maakt een GridExample-klasse die een tweedimensionaal grid implementeert met behulp van een tweedimensionale array van gehele getallen (int). De GridExample-klasse heeft methoden om waarden aan het grid toe te wijzen, waarden op te halen en het grid af te drukken. In de main-methode wordt een GridExample-object gemaakt en worden enkele waarden toegewezen aan het grid. Vervolgens wordt het grid afgedrukt. Het resultaat zou er als volgt uitzien:

Copy code

1 2 3

4 5 6

7 8 9

Dit is slechts een eenvoudig voorbeeld om het concept van een grid te illustreren. Je kunt de GridExample-klasse verder uitbreiden en aanpassen aan je specifieke behoeften, afhankelijk van de vereisten van je programma.

Regenerate response

Gridmanipulatie: U kunt het grid wijzigen door waarden in specifieke elementen te veranderen, elementen toe te voegen of te verwijderen, rijen of kolommen toe te voegen of te verwijderen, enzovoort. Deze bewerkingen kunnen worden uitgevoerd op basis van

import java.util.Arrays;

public class GridExample {

private int rows;

private int columns;

private String[][] grid;

public GridExample(int rows, int columns) {

this.rows = rows;

this.columns = columns;

grid = new String[rows][columns];

// Vul het grid met lege cellen

for (String[] row : grid) {

Arrays.fill(row, "");

}

}

public void setCellContent(int row, int column, String content) {

grid[row][column] = content;

}

public String getCellContent(int row, int column) {

return grid[row][column];

}

public void printGrid() {

for (String[] row : grid) {

for (String cell : row) {

System.out.print(cell + " ");

}

System.out.println();

}

}

public static void main(String[] args) {

GridExample gridExample = new GridExample(3, 3);

// Inhoud toewijzen aan cellen

gridExample.setCellContent(0, 0, "<h1>Title 1</h1>");

gridExample.setCellContent(0, 1, "<p>Paragraph 1</p>");

gridExample.setCellContent(0, 2, "<img src=\"image1.jpg\">");

gridExample.setCellContent(1, 0, "<h1>Title 2</h1>");

gridExample.setCellContent(1, 1, "<p>Paragraph 2</p>");

gridExample.setCellContent(1, 2, "<img src=\"image2.jpg\">");

gridExample.setCellContent(2, 0, "<h1>Title 3</h1>");

gridExample.setCellContent(2, 1, "<p>Paragraph 3</p>");

gridExample.setCellContent(2, 2, "<img src=\"image3.jpg\">");

// Grid afdrukken

gridExample.printGrid();

}

}specifieke vereisten en logica van uw toepassing.

Om de breedte en hoogte van het grid aan te passen, moet je enkele wijzigingen aanbrengen in de GridExample-klasse. Hier is een bijgewerkte versie van de klasse met de mogelijkheid om de breedte en hoogte dynamisch in te stellen:

java

import java.util.Arrays;

public class GridExample {

private int rows;

private int columns;

private String[][] grid;

public GridExample(int rows, int columns) {

this.rows = rows;

this.columns = columns;

grid = new String[rows][columns];

// Vul het grid met lege cellen

for (String[] row : grid) {

Arrays.fill(row, "");

}

}

public void setCellContent(int row, int column, String content) {

grid[row][column] = content;

}

public String getCellContent(int row, int column) {

return grid[row][column];

}

public void printGrid() {

for (String[] row : grid) {

for (String cell : row) {

System.out.print(cell + " ");

}

System.out.println();

}

}

public void resizeGrid(int newRows, int newColumns) {

String[][] newGrid = new String[newRows][newColumns];

// Kopieer de inhoud van de oude grid naar de nieuwe grid

for (int i = 0; i < Math.min(rows, newRows); i++) {

System.arraycopy(grid[i], 0, newGrid[i], 0, Math.min(columns, newColumns));

}

grid = newGrid;

rows = newRows;

columns = newColumns;

}

public static void main(String[] args) {

GridExample gridExample = new GridExample(3, 3);

// Inhoud toewijzen aan cellen

gridExample.setCellContent(0, 0, "<h1>Title 1</h1>");

gridExample.setCellContent(0, 1, "<p>Paragraph 1</p>");

gridExample.setCellContent(0, 2, "<img src=\"image1.jpg\">");

gridExample.setCellContent(1, 0, "<h1>Title 2</h1>");

gridExample.setCellContent(1, 1, "<p>Paragraph 2</p>");

gridExample.setCellContent(1, 2, "<img src=\"image2.jpg\">");

gridExample.setCellContent(2, 0, "<h1>Title 3</h1>");

gridExample.setCellContent(2, 1, "<p>Paragraph 3</p>");

gridExample.setCellContent(2, 2, "<img src=\"image3.jpg\">");

// Grid afdrukken

gridExample.printGrid();

// Grid formaat wijzigen

gridExample.resizeGrid(4, 4);

// Inhoud toewijzen aan nieuwe cellen

gridExample.setCellContent(3, 0, "<h1>Title 4</h1>");

gridExample.setCellContent(3, 1, "<p>Paragraph 4</p>");

gridExample.setCellContent(3, 2, "<img src=\"image4.jpg\">");

gridExample.setCellContent(3, 3, "<h1>Title 5</h1>");

// Grid opnieuw afdrukken na formaatwijziging

gridExample.printGrid();

}

}

##### 2 benaderingen

In Java is een "grid" geen specifieke collectie zoals een List of Set. Het is eerder een concept dat wordt gebruikt om een tweedimensionale structuur van elementen of waarden weer te geven. Je kunt een grid in Java echter implementeren met behulp van een geschikte datastructuur.

Er zijn verschillende benaderingen om een grid te implementeren in Java, afhankelijk van je specifieke vereisten. Hier zijn twee veelvoorkomende benaderingen:

Array: Je kunt een grid implementeren met behulp van een tweedimensionale array. Bijvoorbeeld:

java

Copy code

int rows = 3;

int columns = 3;

int[][] grid = new int[rows][columns];

// Elementen toewijzen aan het grid

grid[0][0] = 1;

grid[0][1] = 2;

grid[0][2] = 3;

// ...

// Element ophalen uit het grid

int element = grid[1][2];

In dit voorbeeld wordt een tweedimensionale int-array gebruikt om een grid van 3x3 elementen voor te stellen. Je kunt waarden toewijzen aan specifieke cellen in het grid en de waarden uit het grid ophalen op basis van de rij- en kolomindices.

Collectie van Collecties: Je kunt ook een grid implementeren met behulp van een collectie van collecties, zoals een List van List.

java

int rows = 3;

int columns = 3;

List<List<Integer>> grid = new ArrayList<>();

// Grid initialiseren

for (int i = 0; i < rows; i++) {

List<Integer> row = new ArrayList<>();

for (int j = 0; j < columns; j++) {

row.add(0); // Voeg een leeg element toe aan elke kolom

}

grid.add(row); // Voeg de rij toe aan het grid

}

// Elementen toewijzen aan het grid

grid.get(0).set(0, 1);

grid.get(0).set(1, 2);

grid.get(0).set(2, 3);

// ...

// Element ophalen uit het grid

int element = grid.get(1).get(2);

In dit voorbeeld wordt een List van List gebruikt om een grid van 3x3 elementen voor te stellen. Elke binnenste List vertegenwoordigt een rij in het grid, en elementen worden toegewezen aan specifieke cellen in het grid door de juiste binnenste List op te halen op basis van de rij- en kolomindices.

De keuze tussen een array en een collectie van collecties hangt af van de vereisten van je

#### CSS

Het CSS display: grid property wordt gebruikt om een element te transformeren naar een grid-container. Hierdoor kun je eenvoudig een tweedimensionale grid-layout maken voor de inhoud van het element.

De belangrijkste eigenschappen van display: grid zijn:

Grid Container: Het element waarop display: grid wordt toegepast, wordt de grid-container. Dit kan elk willekeurig HTML-element zijn, zoals een <div>. De grid-container fungeert als een container voor de grid-items.

Grid Items: De directe kinderen van de grid-container worden grid-items. Dit kunnen HTML-elementen zijn zoals <div>, <p>, <img>, enz. Elke grid-item wordt geplaatst binnen het grid volgens de regels die zijn gedefinieerd voor het grid.

Grid Lines: Het grid bestaat uit horizontale en verticale lijnen, genaamd grid lines. Grid lines worden genummerd vanaf 1. Je kunt deze lijnen gebruiken om de positie van grid-items te bepalen.

Grid Tracks: De ruimte tussen twee aangrenzende grid lines wordt een grid track genoemd. Grid tracks kunnen zowel rijen als kolommen zijn.

Grid Cell: Het gebied tussen vier aangrenzende grid lines wordt een grid cell genoemd. Een grid cell kan één of meerdere grid-items bevatten.

Om display: grid te gebruiken, volg je deze stappen:

Definieer de grid-container door het gewenste element te selecteren in je CSS-bestand.

Gebruik de eigenschap display: grid om het element te transformeren naar een grid-container.

Definieer de gewenste lay-out van de rijen en kolommen met behulp van de eigenschappen grid-template-rows en grid-template-columns. Je kunt de grootte van de rijen en kolommen instellen met specifieke waarden zoals pixels, percentages of fr units (een fr unit verdeelt de beschikbare ruimte in gelijke delen).

Plaats de grid-items binnen het grid met behulp van de eigenschap grid-row en grid-column. Je kunt specifieke grid lines of grid track namen gebruiken om de positie van de items te bepalen.

Hier is een voorbeeld van hoe je display: grid kunt gebruiken:

HTML:

html

<div class="grid-container">

<div class="grid-item">Item 1</div>

<div class="grid-item">Item 2</div>

<div class="grid-item">Item 3</div>

</div>

CSS:

css

Copy code

.grid-container {

display: grid;

grid-template-rows: 50px 100px; /\* Twee rijen met verschillende hoogtes \*/

grid-template-columns: 1fr 2fr; /\* Twee kolommen met verhouding 1:2 \*/

}

.grid-item {

border: 1px solid black;

padding: 10px;

}

#### PDF met overzicht

Dit zijn slechts enkele van de tientallen mogelijkheden van CSS Grid. ["The complete Guide to Grid"](https://css-tricks.com/snippets/css/complete-guide-grid/)

Afbeelding met tekst, schermopname, ontwerp

Automatisch gegenereerde beschrijving