VZHŮRU DO (RESPONZIVNÍHO) WEBDESIGNU

Martin Michálek

Obsah

Kapitola 6: Obrázky a další média

Obsah vložený do prohlížeče je z podstaty responzivní. No, nedivte se – je to pravda. Jenže platí pro text, ne už další elementy. Obrázky, tabulky, video… Vsadím se, že takovou věc jste na webu už viděli.

1. Vrátíme se ke kořenům technického pojetí responzivního designu, abychom poznali, proč je potřeba mediálnímu obsahu věnovat tolik péče.
2. Pak se zaměříme na způsoby, jak přinutit mediální obsah k pružnému chování.
3. Zaostříme na obrázky a ukážeme si, jak různým zařízením posílat obrázky podle jejich zásluh.
4. Ukážu vám nejuniverzálnější metodu pro ošetření obrázků, nové atributy značky `<img>`.
5. Element `<picture>` možná také neznáte, že?
6. Vektorové SVG donutíme k pružnému chování ve všech prohlížečích, i když v Internet Exploreru to bude docela fuška. Uvařte si silné kafe.
7. A tabulky? Ach, ty responzivní tabulky. Těšte se na fakt hodně možných variant jejich návrhu a implementace.
8. Responzivní grafy vezmeme letem světem, protože tak časté nejsou.
9. Radostně také hlásím návrat ke kódování našeho fiktivního e-shopu. Naimplementujeme si do něj responzivní obrázky.

Marcotteho responzivní design

Původní definice responzivního designu pochází z roku 2010, hlavy a pera Ethana Marcotteho. A je to nadmíru jednoduchá, technicistní myšlenka.

1. Udělejte pružný layout.
2. Doplňte to pružnými obrázky.
3. Přidejte podmínky pro změny layoutu, Media Queries.

Článek z A List Apart nesoucí titul „Responsive Web Design“ je asi nejsdílenější článek v historii webdesignu a k téhle knížce vám asi nic moc nového neřekne. Odkázat na něj ale musím. Ethan Marcotte má můj respekt, protože to vymyslel a zformuloval tak skvěle, že si dnešní webdesign bez „responzivnosti“ už ani neumíme představit. alistapart.com/article/responsive-web-design

Tři principy responzivního designu: Pružný layout, pružné obrázky, Media Queries

1. Pružný layout

Ethan Marcotte psal článek (a pak ještě knihu) v době, kdy byl nebohý internet přeplněný weby připravenými pouze pro počítače. Skoro všechny měly fixní šířku layoutu, nastavenou v konkrétní pixelové hodnotě. No jen si to představte. Ach, bolí mě už jen to pomyšlení!

Pružný layout naproti tomu mění rozměry podle velikosti okna. Nejčastěji je definovaný v procentech ze šířky okna.

Dnes už bych nikomu nedoporučoval vymýšlet weby jen pro velké displeje. Tehdy to ale nikdo jinak nedělal. V roce 2010 tedy musel Marcotte a jeho tým zbourat něco jako Stalinův pomník tehdejšího webdesignu: Layout fixních rozměrů optimalizovaný pro velké displeje.

O layoutu píšu v samostatné kapitole.

2. Pružné obrázky

Když už máte pružný layout, musíte mu přizpůsobit elementy vevnitř. První věc, která vám ze zmenšeného layoutu vypadne, budou obrázky.

O obrázcích a jiných médiích dále píšu v samostatných textech.

3. Podmínky pro změny layoutu (Media Queries)

Představte si, že na tehdejší počítačový web nasadíte pružný layout a pružné obrázky. Máte? Teď mírně zmenšete okno prohlížeče. Vše funguje hezky, že? Co když ale okno zmenšíte opravdu hodně,

řekněme na velikost malých mobilů? Tam pružný layout nepomůže. Musíme tedy změnit layout nebo ho úplně zrušit. Pomohou nám k tomu Media Queries, které si spolu rozpitváme později, v kapitole o layoutu.

Tímto končíme prohlídku naší kuchyně. Nejzákladnější surovinu, vývar responzivního designu, máme. Další podkapitoly už budou detailně rozebírat responzivní přizpůsobování veškerého možného netextového obsahu.

Principy responzivního webdesignu by se daly ještě zjednodušit. Takhle jej vysvětluji na školeních:

> *Vložte obsah do prohlížeče. Zvětšujte nebo zmenšujte okno. A odstraňujte problémy, které vidíte.*

Ten bonmot je překvapivě pravdivý. Při zmenšování nebo zvětšování okna se musíte vypořádat s nepružnými médii, obsahu dát layoutový rámec, vyřešit rychlost načítání…

Vkládaná média se zachováním poměru stran

Jak zařídit, aby se obrázky, video a prvky vkládané přes `<iframe>` přizpůsobovaly šířce rodičovského elementu a ještě k tomu zachovávaly poměr stran?

Pružné obrázky

To nebude nic složitého. Vezmeme to rovnou od kódu, co říkáte?

```
.content img {
  max-width: 100%;
  height: auto;
}
```

`.content img`

Selektor by mohl být jednodušší (jen `img`), ale mám ve zvyku pružnost aplikovat jen na obrázky obsahové. Obvykle se nehodí takto ošetřovat například logotyp nebo ikony v hlavičce či patičce stránky.

`max-width: 100%`

Nastaví šířku obrázku podle šířky rodiče. Neudělá ale ten ošklivý trapas, že by jej roztahoval nad rámec jeho velikosti v pixelech a tím deformoval. To by udělala nezbednější kolegyně mezi vlastnostmi, `width`.

`height: auto`

Uvést musíme, aby si obrázek zároveň zachoval poměr stran.
V opačném případě by se poměrově deformoval.

Vyzkoušet si to můžete na CodePenu: cdpn.io/e/jWebge

Pružné vkládané elementy se zachovaním poměru stran

Vkládáte do stránky kód třetí strany pomocí `<iframe>`, máte tam `<video>` nebo nedejbože Flash v `<object>`?

Většina takto vkládaných elementů třetí strany má jakés-takés responzivní chování zajištěno díky výchově na straně dodavatele kódu. Vám se ale může stát, že i u nich potřebujete zachovat poměr stran.

```
.rwd-object {
  position: relative;
  height: 0;
  padding-bottom: 60%; /* Udává poměr stran */
}

.rwd-object-in {
  position: absolute;
  width: 100%;
  height: 100%;
}
```

V HTML to pak bude vypadat asi takto:

```
<div class="rwd-object">
  <iframe class="rwd-object-in" …></iframe>
</div>
```

A zase si to pojďme vysvětlit. To nejzajímavější se děje v kódu třídy `.rwd-object`:

`position: relative`
Vytvoří nový omezující blok, ve kterém lze absolutně pozicovat.

`height: 0`
Potřebujeme vynulovat proto, abychom výšku elementu mohli nastavit pomocí vnitřních okrajů prvku.

`padding-bottom: 60%`
Svislý vnitřní okraj se na rozdíl od vlastnosti `height` počítá ze šířky elementu, takže s jeho pomocí určíme poměr stran rodičovského bloku.

Poměr stran je zde tedy 100 ku 60, takže 5 : 3. Pro poměr 16 : 9 bychom do `padding-bottom` vložili hodnotu `56.25%`. Tady je ještě výpočet: `(9 / 16 * 100)`.

`.rwd-object-in`

Tuto třídu pak aplikujeme přímo na `<iframe>` nebo jiný vkládaný element. Ten už má za úkol se jen sobecky roztahovat na celou výšku i šířku rodiče.

Máte svrbění si to hned zkoušet? Neváhejte, CodePen se na vás těší. cdpn.io/e/BdniC

Tímto jsme ošetřili *pružnost* obrázků a vkládaných médií v rámci layoutu. *Responzivnost* se ale definuje šířeji než jen prostou flexibilní reakcí na změnu šířky layoutu. Jdeme na ni. Zaměříme se nejprve na obrázky, což je bezpochyby nejčastější mediální obsah.

Responzivní obrázky

V responzivním designu máme na výběr poměrně hodně řešení možných problémů s obrázky.

Pojďme si nejprve udělat mapu těch problémů:

- *Rychlost načítání*
  Velký obrázek pro počítač je zbytečné posílat do mobilu. Datový objem je jeden z největších problémů bránících rychlému načtení.
- *Retina displeje*
  Pokud má displej dvojnásobnou a vyšší hustotu hardwarových pixelů, bude tam obyčejná bitmapa vypadat špatně. Znáte to z textu o CSS pixelu.
- *Art direction*
  Občas chceme na různá zařízení poslat různé výřezy obrazovky. Celou fotku na počítač a výřez obličeje na mobil například.
- *Velikost okna*
  Pro různě velká okna prohlížeče bychom rádi servírovali různé

varianty obrázků.

- *Layout*
  Úplně nejraději bychom, aby obrázky znaly layout stránky, protože ten je v responzivním designu velmi variabilní.

Nejčastěji chceme ušetřit datový objem stránky na mobilech nebo poskytnout kvalitní zobrazení pro vysokokapacitní displeje typu Retina.

Ukažme si proto kompletní přehled všech možných řešení, jejich výhod a nevýhod.

Řešení	Rychlost	Retina	AD	Okno	Layout	Vhodné pro
1. SVG	+	+	-	-	-	vektory
2. `<img src>`	-	-	-	-	-	cokoliv
3. `<img src>` 2×	-	+	-	-	-	cokoliv
4. `<img src>` kompr.	+	+	-	-	-	fotky
5. `<img src>`	-	+	+	+	-	cokoliv
6. `<img srcset>`	+	+	-	+	-	cokoliv
7. `<img srcset sizes>`	+	+	-	-	+	cokoliv
8. `<picture>`	+	+	+	+	-	cokoliv

Srovnání řešení pro responzivní obrázky. Rychlost – zohledňují rychlost načítání? Retina – zohledňují vysokokapacitní displeje? AD (Art Direction) – dokážou poslat různé ořezy obrázků na různá zařízení? Okno – umí vybírat obrázky podle velikosti okna prohlížeče? Layout – zohledňují layout webu?

Ve srovnání jsem leccos zjednodušil. Nevidíte tam, že jednotlivá řešení lze kombinovat. Třeba komprimované obrázky s technikou `srcset/sizes`. Ale to vám určitě došlo nebo dojde z dalšího textu. A teď už na jednotlivá řešení.

1. Vektory? SVG

Tohle je jednoduché. Máte-li obrázek vyjádřitelný vektorem, prostě z něj udělejte SVG a pošlete jej ve stránce prohlížečům. Pokud to extra nezmrvíte, vektory jsou datově velmi úsporné a automaticky připravené.

O responzivních SVG píšu v jedné z dalších podkapitol.

2. Staré dobré `<img src>`

S bitmapami to bude složitější, ale jednu věc vím jistě. Jeden neoptimalizovaný obrázek vám pravděpodobně stačit nebude.

```
<!-- Bude datově neúsporný nebo
     ošklivý na Retina displejích: -->
<img src="image_100x100.jpg"
  width="100" height="100" alt="…">
```

Tahle (ne)technika patří do muzea webového vývoje. Podlaží Počítačová éra.

3. Dvojnásobná velikost obrázku v `<img src>`

Občas se ještě setkávám s řešením, které upřednostňuje Retina displeje. Autoři prostě obrázek vloží ve dvojnásobné fyzické velikosti oproti původnímu:

```
<!-- Bude datově neúsporný (a možná také
     ošklivý na Retina displejích): -->
<img src="image_200x200.jpg"
  width="100" height="100" alt="…">
```

Je to samozřejmě nevýhodné pro rychlost načtení na běžných (ne-

Retina) displejích. Raději vás upozorním, že obrázek nebude datově dvakrát tak velký, ale tři- nebo čtyřikrát. Obsahuje přece čtyřnásobný počet pixelů. Zajímavější to začne být, když obrázku uberete na zobrazovací kvalitě.

4. Razantně komprimované obrázky v `<img src>`

Datový objem i vysokokapacitní displeje můžete v některých situacích vyřešit naráz. Prostě zvětšíte pixelovou velikost obrázku a výrazně snížíte jeho kvalitu:

```
<img src="image_300x300_lowquality.jpg"
  width="100" height="100" alt="…">
```

Jak vypadá výroba takového obrázku ve třech krocích?

1. Původní obrázek uložíte ve výrazně větší pixelové velikosti.
2. Snížíte kvalitu exportu někam výrazně pod polovinu.
3. Prohlížeč necháte obrázek převzorkovat na původní velikost.

Komprimované obrázky jsme zkoušeli nasadit na jednom starším projektu. Udělali jsme si testy pro různé kombinace komprese a pixelové velikosti. Nakonec jsme došli k tomu, že obrázky ve dvojnásobné pixelové velikosti a kvalitě komprese nastavené na 30 % měly nejlepší poměr kvality a datového objemu. Ten byl poloviční oproti původní verzi s 80% kvalitou a velikostí stejnou, jako se používá ve stránce. U různých typů obrázků to ale bude různé.

Autoři nápadu, Filament Group, svůj zkušební obrázek vkládali dvaapůlkrát větší a kvalitu JPG snížili na 0 %. Výsledný obrázek se pyšnil opět méně než polovinou datového objemu toho původního. vrdl.in/z7k34

Asi sami vidíte, že řešení je vhodné jen pro JPG nebo WebP obrázky, kde je možné nastavit ztrátovou kompresi různých úrovní. Typově je pak použití metody vhodné spíše pro fotografie než třeba obrázky s textem, kde by v ostrých hranách mezi barvami byla ztráta kvality viditelná.

5. Vlastní řešení pomocí `<img src/data-src>`

Občas je pro responzivní obrázky možné vidět řešení s nahrazováním atributu `src`:

```
<img src="image_100x100.jpg"
  data-src="image_300x300.jpg"
  width="100" height="100" alt="…">
```

Na velkých displejích pak autoři těchto řešení usilují o zkopírování obsahu `data-src` do `src` pomocí Javascriptu. Ano, prohlížeč pak zobrazí správný obrázek. Takto pracuje například knihovna Response. responsejs.com

Na pohled elegantní, ale nevýhody to má. Neexistuje totiž způsob, jak prohlížeč odradit od stažení obrázku nalinkovaného v atributu `src`. Proto se v těchto řešeních obrázek sice vymění, ale předtím se už stáhl tento soubor. To není potěšující zpráva pro uživatele čekající na pomalém připojení.

Navíc je nutné naprogramovat i logiku pro další scénáře, které mají responzivní obrázky řešit. Například ony Retina displeje. Logiku, kterou už navíc prohlížeče mají v sobě. Hned k ní dojdeme, ale musíme se rozloučit se starým známým atributem `src`.

Iniciativa Responsive Images Community Group totiž před lety přišla s novými atributy – `srcset` a `sizes` – a také s úplně novým tagem `<picture>`. To jsou řešení, která dnes považuji za standardní,

a pokud je to možné, dávám jim přednost.

6. Atribut `srcset` značky `<img>`

Hodí se pro scénář s výběrem varianty podle velikosti okna. Do atributu `srcset` uvedete velikostní varianty, které jste si předtím uložili na server:

```
<img src="image_300x300.jpg"
  srcset="
    image_100x100.jpg 100w,
    image_200x200.jpg 200w,
    image_300x300.jpg 300w"
  width="100" height="100"
  alt="…">
```

Všimněte si `w`, takzvaného *deskriptoru*, který nese informaci o šířce obrázku. Proč je tam potřeba? Dobrá otázka, zodpovíme si ji v textu o atributech `srcset` a `sizes`. Těší se na vás hned v další podkapitole.

7. Atribut `sizes` značky `<img>`

Řešení s atributem `srcset` je fajn, ale zajistí výměnu obrázků jen podle velikosti okna. Obrázky se ale obvykle vyskytují v nějakém prostředí layoutu webu. Proto potřebujeme ještě atribut `sizes`, kterým prohlížeči předáváme onu informaci o layoutu:

```
<img src="image_300x300.jpg"
  srcset="
    image_100x100.jpg 100w,
    image_200x200.jpg 200w,
    image_300x300.jpg 300w"
  sizes="980px"
  width="100" height="100"
  alt="…">
```

Pokud chcete více informací, odkážu vás opět na podrobně rozepsaný materiál o atributech `srcset` a `sizes`.

8. Nová značka `<picture>`

Nový tag `<picture>` vymysleli pro méně časté scénáře – třeba když potřebujete mít na konkrétních velikostech layoutu jinak oříznuté obrázky:

```
<picture>
    <source media="(min-width: 600px)"
srcset="image_100x100.jpg">
    <source media="(min-width: 1024px)"
srcset="image_300x300.jpg">
    <img src="image_100x100.jpg" alt="…">
</picture>
```

Na první pohled méně zkušených očí vypadá užitečněji než atributy `srcset` a `sizes`, ale není to pravda. Hodí se opravdu hlavně jen na ty speciální ořezové verze a další méně časté scénáře. Více si přečtete v samostatném textu.

Co ale ještě zmínit chci, je podpora nových atributů a značky `<picture>` v prohlížečích. Je výborná, nebojte se.

Podpora `srcset`, `sizes` a `<picture>` v prohlížečích

Podporují je všechny moderní prohlížeče. Responzivní obrázky nám chybí hlavně ve všech verzích Exploreru (a taky Android Browseru do čtyřkových verzí Androidu, ale to už nás asi nemusí trápit). caniuse.com/srcset

Obzvlášť IE ve verzi 11 je ke dni psaní textu ještě velmi silně zastoupený. Je však dobré si uvědomit, jaké je v tomto případě chování „nepodporujících prohlížečů“.

První náhradní řešení: přirozené

Použijete parametr `src`, který moderní prohlížeče ignorují, pokud je přítomný `srcset`:

```
<img
  src="starsi_prohlizece.jpg"
  srcset="nove_prohlizece.jpg …">
```

Druhé náhradní řešení: Picturefill

Javascriptová knihovna, která zařídí fungování atributů `srcset`, `sizes` a značky `<picture>` i ve starších prohlížečích. Jmenuje se Picturefill a považuji ji za dobré řešení, které mám odzkoušené na několika webech. scottjehl.github.io/picturefill

Atributy responzivních obrázků: srcset a sizes

Nové atributy řeší potřebu autorů stránek zobrazovat v různých kontextech designu různé varianty obrázků.

Změna kontextu v tomto případě nejčastěji vypadá jako změna layoutu pro jinou velikost obrazovky. Může ale jít také o zobrazení stránky na zařízeních s displeji typu Retina (různými poměry `device-pixel-ratio`). Do budoucna třeba ještě o změnu úrovně zoomu na stránce nebo uživatele na pomalém připojení.

Na atributech `srcset` a `sizes` je hezké, že poměrně složité rozhodování, který obrázek ve které situaci použít, necháváme na prohlížeči.

Jako autoři stránky mu jen řekneme, jaké varianty obrázku má k dispozici (`srcset`) a jak jsou veliké na jednotlivých breakpointech layoutu (`sizes`).

`srcset`: Sada zdrojů obrázku a jejich vlastností

```
<img src="medium_1024.png"
  srcset="
    small_600.png 600w,
    medium_1024.png 1024w,
    large_1600.png 1600w"
  width="200" height="200" alt="…">
```

Prohlížeči tímhle kódem sdělujeme, že jsme předgenerovali obrázek `small_600.png`. Jeho fyzická šířka (při exportu z grafického editoru) je 600 pixelů, což říká zápis `600w`, k němuž se ještě dostaneme. Dále zde máme obrázek `medium_1024.png` v šířce 1024 pixelů a `large_1600.png` v šířce 1600 pixelů. V atributu `src` pak uvádíme náhradní řešení pro prohlížeče, které atribut `srcset` neumí.

Prohlížeč se zde při rozhodování o tom, který obrázek stáhnout a zobrazit, dívá na aktuální šířku okna.

Některé prohlížeče na to půjdou chytřeji a například `small_600.png` budou zobrazovat ještě kousek nad hranicí šestisetpixelového okna, protože na vizuální kvalitě obrázku to neubere.

Prohlížeč vezme v potaz i aktuální `device-pixel-ratio`. Například na zařízení s původním Retina displejem (`device-pixel-ratio=2`) pak v případě, že okno je široké 600 pixelů, stáhne a použije už největší obrázek, `large_1600.png`. Potřebuje obrázek velikosti šířky okna krát `device-pixel-ratio`. Takže kolem 1200 pixelů široký. Obrázek `medium_1024.png` by mu tedy nestačil.

V potenciálu chytrého rozhodování prohlížeče vězí krása atributu `srcset`. Prohlížeč zváží všechny informace, které má o stavu stránky k dispozici a podle toho vybere nejvhodnější obrázek. Vy jako autoři jen vygenerujete dost variant a správně je popíšete.

Demo výše uvedeného kódu mám také na CodePenu. Nejlépe jej vyzkoušíte, když si zmenšíte okno ukázky, obnovíte stránku a pak budete okno postupně zvětšovat. cdpn.io/e/WboGgE

Kolik variant obrázků vygenerovat?

V ukázce mám obrázky tři, to by ale v praxi nestačilo. Obvykle si vypočtu šířku nejmenší a největší možné varianty. Mezi nimi pak nechám vygenerovat obrázky odstupňované po 200 až 400 pixelech. Je to samozřejmě někdy limitováno i úložným místem na serveru, takže tuto mou radu prosím berte s rezervou.

Detailně si postup hledání variant ukážeme už za chvíli na příkladu.

Deskriptory vlastností obrázků v `srcset`

Zatím jsem zmínil jen šířku obrázku, tedy deskriptor `w`. Ten budete používat v naprosté většině případů. Deskriptor `x` jej doplňuje pro specifické scénáře použití.

deskriptor `w`

Udává, jakou pixelovou šířku má ve skutečnosti soubor s obrázkem.

```
<img srcset="
      small_600.png 600w,
      medium_1024.png 1024w"
    width="200" height="200" alt="…">
```

Pozor, neříká nic o šířce obrázku v rámci layoutu stránky. K tomu dále slouží atribut `width` nebo případně CSS.

deskriptor `x`

Druhý deskriptor určuje připravenost souboru s obrázkem pro různé poměry `device-pixel-ratio`, například:

```
<img srcset="
      image.png,
      image@2x.png 2x"
    width="200" height="200" alt="…">
```

Tímto zápisem říkám, že `image@2x.png` má prohlížeč použít při `device-pixel-ratio` alespoň 2 a `image.png` ve všech hodnotách menších než 2. Když deskriptor neuvedete, výchozí je `1x`.

Pojďme se teď ještě podívat na atribut `sizes`, který prohlížeči umožní vybírat nejen podle fyzických parametrů souborů s obrázky, ale i podle layoutu vaší stránky.

`sizes`: Velikost obrázku ve stránce

V praxi totiž tak často nepotřebujeme, aby prohlížeč vybral obrázek podle šířky okna. Spíše podle šířky prostoru pro obrázek v rámci aktuálního layoutu stránky. A právě od toho máme atribut `sizes`:

```
<img src="small.png"
  srcset="
    small_600.png 600w,
    medium_1024.png 1024w,
    large_1600.png 1600w"
  sizes="
    (min-width: 768px) 300px,
    100vw"
  width="200" height="200" alt="…">
```

Sledujte hodnoty v `sizes`. Responzivní layout v ukázce je vymyšlený takto:

- V oknech šířky 768 pixelů a více se obrázek vykresluje do plochy o šířce 300 pixelů (`(min-width: 768px) 300px`).
- Ve všech ostatních velikostech okna, tedy do 767 pixelů, zabere 100 procent šířky okna (`100vw`).

„Volkswageny“, tedy jednotku `vw`, jsme probírali v textu o jednotkách, vzpomínáte?

První vyhovující varianta v `sizes` vyhrává, takže na pořadí záleží. Pozor na to.

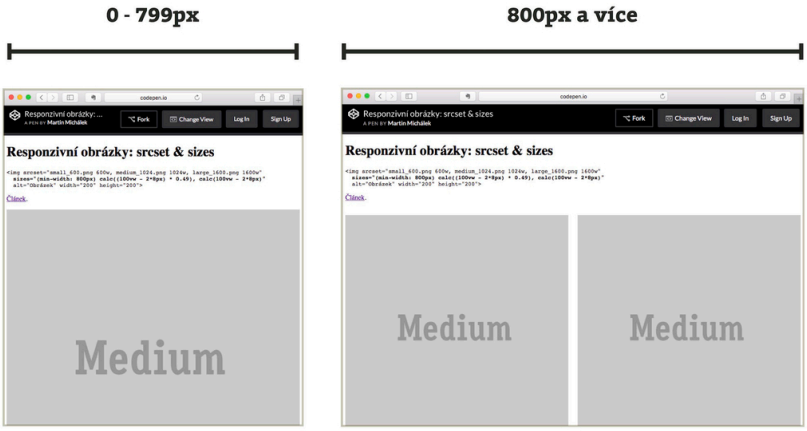
Proč si informaci o layoutu stránky prohlížeč nevezme z CSS?

Dobrá otázka. Rozhodnutí, který z obrázků stáhnout a zobrazit, prohlížeče dělají ještě předtím, než znají CSS. Ony v tu chvíli jen rychle parsují HTML a o stylech zatím „nevědí“. Když by na styly čekaly, zpozdí se stažení a zobrazení obrázků. Je to takhle výhodnější pro uživatele. Zkracuje to načtení obrázků a vlastně celé stránky.

Velikosti obrázků podle layoutu

V responzivním layoutu obvykle přesně nevíme, jaké rozměry budou mít obrázky v rámci konkrétní šířky okna. A právě tady se projeví síla kombinace `sizes` s CSS funkcí `calc()`. Definovat velikost obrázku můžeme relativně k layoutu mezi konkrétními breakpointy.

Pojďme si nejprve vizuálně přiblížit layout pro další ukázku:



Layout příkladu pro demonstraci srcset/sizes

Do `800px` breakpointu je to jednoduché: Obrázek zabírá celou šířku layoutu. Nikoliv ovšem šířku okna, a tak musíme odečíst výchozí `margin` u `<body>`, který mají prohlížeče nastavený na `8px`:

```
calc(100vw - 2 * 8px)
```

Od breakpointu `800px` pak musíme vyjít z CSS layoutu, který je zapsaný takto:

```
@media only screen and (min-width: 800px) {
  .image {
    width: 49%;
  }
}
```

Přepsáno do funkce `calc()` to vypadá takto:

```
calc((100vw - 2 * 8px) * 0.49)
```

Ještě v jazyce našeho kmene:

```
(100 procent šířky viewportu - výchozí margin u <body>)
  * 49% šířka obrázku
```

Takže celý zápis tagu `<img>` bude vypadat takto:

```
<img src="small_600.png"
  srcset="
    small_600.png 600w,
    medium_1024.png 1024w,
    large_1600.png 1600w"
  sizes="
    (min-width: 800px) calc((100vw - 2*8px) * 0.49),
    calc(100vw - 2*8px)"
  width="200" height="200" alt="…"  >
```

Pojďme si pro jistotu ještě shrnout celý zápis:

1. V `src` máme obrázek sloužící jako náhradní řešení pro starší prohlížeče.
2. V `srcset` máme seznam variant obrázku, které jsme předpřipravili a uložili na server.
3. Atribut `sizes` říká: na šířkách okna od 800 pixelů výše bude mít obrázek velikost `calc((100vw - 2 * 8px) * 0.49)`. Ve všech ostatních případech – to znamená do 799 pixelů – pak `calc(100vw - 2 * 8px)`.

Demo na CodePenu: cdpn.io/e/azBmaX

Nezapomínejte prosím na povinný atribut `alt`, který ocení vyhledávače a odečítače obrazovky pro zrakově hendikepované uživatele.

Doplňte i atribut `height`, který vylepší vykreslování stránky tím, že ještě před stažením obrázku sdělí prohlížeči, jak velký prostor pro něj má v layoutu vynechat. Nevztahujte jej tedy k pixelové výšce obrázku, ale k prostoru stránky, který si přejete pro obrázek rezervovat do chvíle, než se stáhne a zobrazí.

„Hernajs, a proč je to tak složité?“

Nedivím se samozřejmě žádným námitkám vůči estetice a zdánlivě zbytečné složitosti autorské práce s responzivními obrázky. Moje první reakce byly stejné. Když si ale zopakujeme, že informace z CSS prohlížeči v momentě parsování HTML nijak nepomohou, asi bychom došli k řešení stejnému nebo velmi podobnému. Pokud bychom jej tedy chtěli vymýšlet znovu, což nikomu nedoporučuji.

V textu o responzivních obrázcích jsem zmiňoval i další alternativy. `<img srcset sizes>` ale považuji za výchozí řešení. Ta ostatní se hodí pro konkrétní a méně časté scénáře.

Pojďme si rozpitvat jednu z metod pro specifické situace – novou značku `<picture>`.

Nová značka Picture

`<picture>` umožňuje definovat varianty obrázku pro různé stavy v responzivním webdesignu.

Na rozdíl od atributů `srcset` a `sizes` nenecháváme rozhodování na prohlížeči. Vedení tady přebíráme my autoři. Ukážu tady pár scénářů, kdy je to výhodné.

Ukázka zápisu

```
<picture>
  <source media="(min-width: 1024px)" srcset="large.jpg">
  <source media="(min-width: 600px)" srcset="medium.jpg">
  <img src="small.jpg" alt="…">
</picture>
```

V elementech `<source>` uvádím alternativy k výchozímu obrázku, který je v `<img>`.

Prohlížeč vezme vždy první vyhovující obrázek. Je možné tedy obrázky řadit jak od největšího, tak od nejmenšího obrázku. V prvé případě použijte Media Query `min-width`, v druhém `max-width`. cdpn.io/e/qDmar

Značka `<picture>` přitom tvoří jen obal, zatímco prvky `<source>` jsou jakési molitanové vycpávky nesoucí informaci o alternativách. Veškeré stylování nebo věšení událostí v Javascriptu je nutné dělat přímo na `<img>` elementu. V každém `<picture>` musí být právě jeden `<img>`.

Kdy se vám může `<picture>` hodit? Hlavně ve dvou situacích:

1. Připravili jste obrázky v různých ořezech. Třeba na mobily chcete poslat čtverce a jinde obdélníky. Zároveň chcete mít pod kontrolou hranice, kdy prohlížeč použije jednu, či druhou ořezovou verzi. Jde o „art direction“, tedy autorské řízení formy a obsahu obrázků.
2. Prohlížečům jste obrázky připravili v různých souborových formátech.

V naprosté většině případů vám bude stačit stará dobrá značka `<img>` s atributy `srcset` a `sizes`.

Art direction: obrázky pro různá rozlišení mají také různý obsah

Máme tři varianty obrázků a prohlížeči chceme přesně stanovit hranice přepínání mezi nimi:

```
<picture>
  <source
    srcset="large_1600.png"
    media="(min-width: 1024px)">
  <source
    srcset="medium_1024.png"
    media="(min-width: 800px)">
  <img
    src="small_600.png"
    alt="Obrázek" width="200" height="200">
</picture>
```

Pro okna 1024 pixelů a větší se stáhne a použije obrázek `large_1600.png`, od 800 do 1023 pixelů `medium_1024.png` a pro okna šířky 799 a méně pixelů pak `small_600.png`.

I tady jsem pro vás připravil demo na CodePen. cdpn.io/e/VYPPQQ

V čem se to liší od `<img srcset sizes>`? Příklad, který uvádím výše, je velmi zjednodušený. Museli byste v něm ještě ošetřit displeje typu Retina, tedy různé hodnoty `device-pixel-ratio`. To máte u `srcset` a `sizes` „v ceně“ řešení: prohlížeč to udělá sám. Na druhou stranu tady pomocí jakýchkoliv Media Queries určíte sami hranice mezi variantami. V metodě `srcset` vybírá prohlížeč sám podle layoutu nastaveného v `sizes`.

Jinými slovy: Pokud byste se rozhodli používat `<picture>` pro běžné obrázky, byly by vaše Media Queries v nich uvedené dost složité. Kromě šířky okna by musely zohledňovat velikost obrázku v layoutu a také displeje typu Retina. Pojďme se ale zaměřit na ty scénáře, kdy se nová značka opravdu hodí.

Podle formátu obrázku

Vybírat obrázky prohlížeče umí i podle formátu. Použijte atribut `type`. Hodí se hlavně pro detekci prohlížečů, které zvládají nový

formát WebP. Ten je mimochodem ještě výrazně datově úspornější než JPG, ale ke dni psaní jej podporuje jen Chrome a Opera. caniuse.com/webp

```
<picture>
    <source media="(min-width: 1024px)"
    srcset="large.webp" type="image/webp">
    <source media="(min-width: 1024px)"
    srcset="large.jpg">
    <img src="small.jpg" alt="…">
</picture>`
```

Prohlížeč, který umí formát WebP a běží v okně velikosti alespoň 1024 pixelů, stáhne a zobrazí soubor `large.webp`.

Tímto způsobem je také možné udělat pěkné náhradní řešení pro formát SVG:

```
<picture>
  <source type="image/svg+xml" srcset="logo.svg">
  <img src="logo.png" alt="…">
</picture>
```

Šmytec. O bitmapových obrázcích jsme si toho řekli už dost. Teď vzhůru do vektorů!

Responzivní SVG

Vektorové obrázky jsou fajn. Asi tady není potřeba zahltit vás celou řadou argumentů pro využívání SVG. Ty si případně nastudujte na Vzhůru dolů. vrdl.cz/p/svg

Používání SVG pro ikony nebo logotypy namísto PNG či GIF obrázků hájím kam vkročím, ale v jedné věci jsou bitmapy zlaté: V přizpůsobování šířce layoutu a zachování poměru stran. To, čeho jsme tak snadno dosáhli v kapitole o pružných obrázcích, u SVG

budeme dělat poměrně složitě.

Abychom ale pochopili, proč není dosažení pružnosti SVG jednoduché jako facka, musíme na světlo Boží vytáhnout jednu překvapivou a možná i nepříjemnou pravdu. Nádech…

SVG není obrázek.

… výdech! No jasně, SVG je vektorový dokument, který vkládáme do stránky. Právě proto jej do HTML můžeme dát nejen ve formě obrázku (`<img>`), ale také jako externí zdroj (`<iframe>`, `<object>`) nebo vektory vykreslit přímo (`<svg>`).

Bitmapy mají jasně definovanou výšku i šířku. Je proto velmi snadné jejich poměr stran zachovat. Jaká je ale výška nebo poměr stran dokumentu?

Nastavení šířky a výšky je k ničemu, použijte `viewbox`

Mnoho kodérů se domnívá, že u značky `<svg>` nastaví parametry `width` a `height` – a SVG začne poslouchat. Je to tak ale jen v některých prohlížečích a některých typech vložení do stránky. Celá pravda ovšem je, že nastavením výšky a šířky si mnoho věcí zkomplikujeme.

Vysvětlení je složité, takže vás odkážu na článek, který to rozebírá lépe, než bych to dokázal udělat já. Než se ale do čtení textu „How to Scale SVG“ na CSS Tricks pustíte, ujistěte se, že doma máte dostatečnou zásobu brufenů. css-tricks.com/scale-svg

S brufeny nebo bez, výsledek je stejný. Prostě použijeme parametr `viewbox`:

```
<svg viewbox="0 0 100 50">
```

Kód říká, že výchozí velikost SVG dokumentu je 100 na 50 pixelů, že souřadnicový systém začíná klasicky na bodě nula nula a že si má dokument držet poměr stran.

Jak teď zajistit pružné chování SVG při různých typech vložení do stránky?

Pružné SVG vložené do HTML dokumentu pomocí `<svg>`

V moderních prohlížečích je to v případě dodržení výše uvedeného opravdu jednoduché. Jen kvůli Internet Exploreru musím přidat obalující značku:

```
<p class="svg-container">
  <svg viewbox="0 0 900 400" class="svg-content"> … </svg>
</p>
```

Třídu `.svg-container` pak kvůli Explorerům nastylujeme metodou pro zachování poměru stran, stejně jako to děláme u vkládaných elementů v textu o obrázcích:

```
.svg-container {
  position: relative;
  width: 100%;
  height: 0;
  padding-top: 100%; /* Poměr stran 1:1 */
}

.svg-content {
  position: absolute;
  top: 0;
  left: 0;
}
```

Podívejte na výsledný CodePen. cdpn.io/e/oYOZwz

Pružné SVG vkládané externě pomocí `<img>`

Opět do zdrojového SVG přidáme viewbox a zrušíme případně přítomné atributy `width` a `height`. Pak stačí klasicky vložit do stránky:

```
<img src="logo.svg" class="svg"
  alt="Logo firmy">
```

Pokud nebudeme nastavovat výšku obrázku, v CSS nemusíme pro moderní prohlížeče nastavovat vůbec nic. Opět jen musíme napravit chování Internet Explorerů:

```
.svg {
  width: 100%;
}
```

CodePen s příkladem se na vás těší i tady. cdpn.io/e/VmNbPx

Pružné SVG externě v CSS

Tady je to jednoduché – stačí prostě obrázek umístit na pozadí, zakázat mu opakování a pomocí `background-size: contain` jej rozprostřít do celé šířky rodiče.

```
.svg-icon {
  background-image: url('icon.svg');
  background-repeat: no-repeat;
  background-size: contain;
}
```

U většiny vektorových obrázků pak chceme definovat poměr stran, který si mají zachovat. Opět si pomůžeme trikem pro zachování poměru stran.

```
.svg-icon
  …
  height: 0;
  padding-bottom: 100%; /* Poměr stran 1:1 */
}
```

Tady už není ani žádné speciální nastavení pro Internet Explorer. Hurá!

Mrkněme se spolu na CodePen. cdpn.io/e/NbmgPr

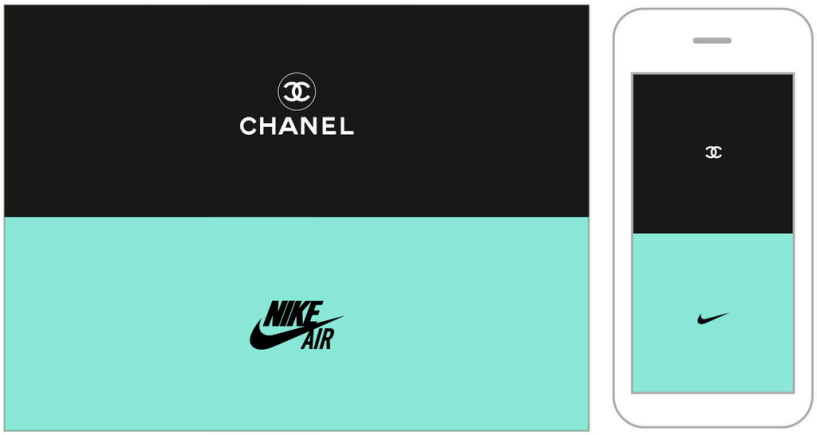
Tím bychom mohli zajištění pružnosti SVG v responzivním layoutu uzavřít pro nejčastěji používané způsoby vložení.

Formát je ale možné do stránky vkládat i dalšími, méně používanými způsoby. Jak zajistit pružnost při vložení do `<object>`, `<iframe>`? Poradí vám vrchní odbornice na SVG, Sara Soueidan, v článku „Making SVGs Responsive with CSS“. vrdl.in/yixth

Ale čtěte dál. To nejlepší teprve přijde. Slíbil jsem povídání o *responzivních* SVG. Zatím jsme se ale bavili jen o těch *pružných*. O těch, které se přizpůsobují šířce rozvržení stránky. Kromě pružného layoutu ale responzivitu definují ještě změny stylování v určitých velikostech obrazovky. Prostě Media Queries.

(Opravdu) responzivní SVG

Ano, uvnitř SVG můžeme Media Queries úplně v pohodě použít. A ano, někdy se podmínky nevztahují k rozměrům celé stránky, ale k rodiči SVG dokumentu.



Využití Media Queries v responzivních SVG najdete na webu „Responsive Logos“. Podívejte se, stojí to za to. responsivelogos.co.uk

K čemu se vztahují Media Queries v SVG?

Podívejme se na jednoduchý příklad, kdy na menších velikostech okna invertujeme barvy loga Vzhůru dolů.

Media Queries prostě napíšeme dovnitř kódu vektorového dokumentu:

```
<svg>
  <style>
    @media all and (max-width: 500px) {
      #layer-background { display: none; }
      #layer-text path { fill: url(#Gradient_1); }
    }
  </style>
  …
</svg>
```

Tady se podmínky v `@media` vcelku logicky vztahují k šířce okna prohlížeče. cdpn.io/e/vyMRPL

Co když ale vložíme SVG soubor obsahující Media Queries externě?

```
<img src="vd_logo_mq.svg"
  width="100" height="100" alt="Logo Vzhůru dolů">
```

Tušíte správně, podmínky v `@media` se pak budou vztahovat k šířce obrázku samotného. cdnp.io/e/zZKzRe

Podmínky budou pracovat jako Container Queries, které bychom ve webdesignu potřebovali jako sůl. Ale nemáme je. Zatím tedy jen u externích SVG. Zmíním je ještě v kapitole o Media Queries.

Tak či tak, mechanismus responzivních SVG má velkou budoucnost: pro ikony, grafy, interaktivní elementy, mapy (!) a další prvky s potřebou měnit hustotu informací nebo formy podle velikosti okna prohlížeče.

A teď už pryč od obrázků a hurá do tabulek a grafů. Slibuji, že to nebude taková nuda, jak to zní.

Responzivní tabulky

Chuck Norris toho zvládne hodně, třeba i rozbrečí cibuli, ale tabulky na webu by mu daly zabrat. No vážně. Však čtěte.

Zejména ty rozsáhlejší mají nehezkou vlastnost, že na menších displejích jsou rozměrově poněkud nezkrotitelné. Pojďme si představit všechny způsoby, jak lze s tabulkami v dnešním webdesignu zacházet, a vy si jistě vyberete. Tedy pokud nejste Chuck Norris. Ten si vybral, ještě než jsem začal psát.

Posun do stran

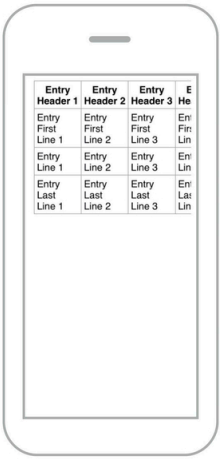
Nejjednodušší varianta. Prostě tabulce přikážete, ať se roluje do

strany, pokud překročí aktuální šířku okna:

```
.scrollable-table {
  overflow-x: auto;
}
```

Uživatel si pak onen posun obstará palcem. Důležité je, aby byla možnost posunu indikována useknutím obsahu zprava. „Scrollbar“, indikátor možnosti posunu, totiž sám o sobě nestačí. Na mobilech nebývá vidět, dokud uživatel na tabulku nezaútočí prstem.

Entry Header 1	Entry Header 2	Entry Header 3	Entry Header 4
Entry First Line 1	Entry First Line 2	Entry First Line 3	Entry First Line 4
Entry Line 1	Entry Line 2	Entry Line 3	Entry Line 4
Entry Last Line 1	Entry Last Line 2	Entry Last Line 3	Entry Last Line 4



Obsah tabulky se na malém displeji posouvá do stran

Vyzkoušejte si zmenšit okno v ukázce. cdpn.io/e/ENMezZ

Řešení se hodí hlavně pro tabulky s menším počtem řádků i sloupců a s popisem dat nahoře. Nebo také pro tabulky vkládané přes redakční systémy, u kterých nevíte, jak složité budou. Anebo když prostě chcete ušetřit čas na vývoj.

Než si ukážeme propracovanější způsoby práce s responzivními tabulkami, dovolte mi jeden tip na nástroj.

Chytrý plugin: Tablesaw

Tablesaw je jQuery plugin, který zvládá téměř všechny zde popsané možnosti chování responzivních tabulek. Zkrátka švýcarský tabulkový nůž Chuck Norrise. github.com/filamentgroup/tablesaw

Posun do stran s fixním sloupcem

Varianta pro tabulky s popisem dat ve svislém směru a klidně i příšerně moc sloupci s daty samotnými. Na malém displeji prstem posunujete do stran jen sloupce s daty. Popis zůstává na místě.

Movie Title	Rank	Year	Rating	Gross
Avatar	1	2009	83%	$2.7B
Titanic	2	1997	88%	$2.1B
The Avengers	3	2012	92%	$1.5B
Harry Potter and the Deathly Hallows—Part 2	4	2011	96%	$1.3B
Frozen	5	2013	89%	$1.2B
Iron Man 3	6	2013	78%	$1.2B
Transformers: Dark of the Moon	7	2011	36%	$1.1B
The Lord of the Rings: The Return of the King	8	2003	95%	$1.1B
Skyfall	9	2012	92%	$1.1B
Transformers: Age of Extinction	10	2014	18%	$1.0B

Tabulka s pevně ukotveným prvním sloupcem na mobilu a možností posouvat gestem „swipe“. github.com/filamentgroup/tablesaw

Vyzkoušejte si naživo v CodePenu. cdpn.io/e/qqvJdV

S propracovanějším řešením využívajícím flexbox a další moderní CSS vlastnosti přišel David Bushell v textu „CSS only Responsive

Tables“. vrdl.in/xlpbn

Fixně-posuvné řešení je pak možné doplnit detekcí gesta švihnutí (swipe) pro snadnější a přesnější posouvání sloupečků.

Řešení má mnoho užití. Podmínkou ale je, aby tabulka měla přijatelně nízký počet řádků.

No jo, ale co když máte tabulku toho typu, kterému programátoři říkají „datagrid“? Ta má, potvora, hodně sloupečků, ale také řádků.

Stohování

Datagrid není žádná vzácnost. Každá webová aplikace pro interní systémy je datagridů plná. Je to případ i vašeho projektu? Pak bych vám doporučil přestylovat tabulku na mobilech do podoby netabulkového, kartičkového zobrazení. Říkám tomu *stohování*.

Movie Title	Rank	Year	Rating	Gross
Avatar	1	2009	83%	$2.7B
Titanic	2	1997	88%	$2.1B
The Avengers	3	2012	92%	$1.5B
Harry Potter and the Deathly Hallows—Part 2	4	2011	96%	$1.3B
Frozen	5	2013	89%	$1.2B
Iron Man 3	6	2013	78%	$1.2B
Transformers: Dark of the Moon	7	2011	36%	$1.1B
The Lord of the Rings: The Return of the King	8	2003	95%	$1.1B
Skyfall	9	2012	92%	$1.1B
Transformers: Age of Extinction	10	2014	18%	$1.0B

Stohování tabulky na menších displejích. github.com/filamentgroup/tablesaw

V nejjednodušší možné CSS implementaci tabulce na menších displejích zrušíme „tabulkovost“:

```
@media only screen and (max-width: 600px) {
  table, th, td, tr, thead, tbody {
    display: block;
  }
}
```

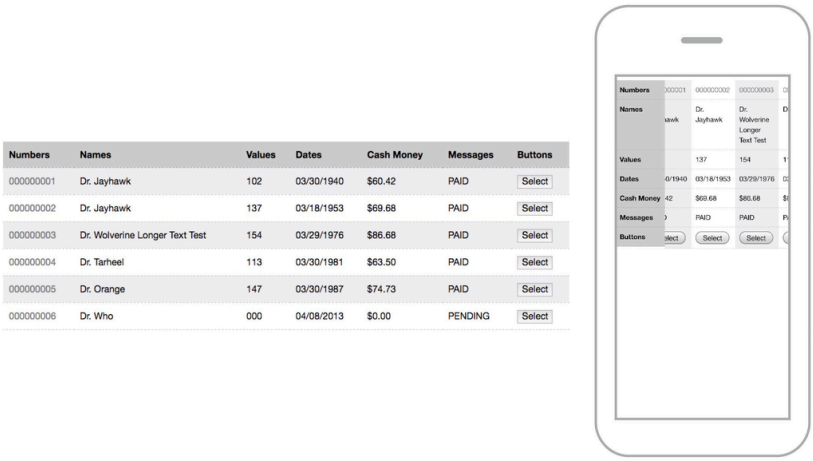
Na CodePenu je možné zkusit si to i s dalším stylováním. cdpn.io/e/bBZmxE

I tak ale čisté CSS řešení nebude dokonalé. Pro tento typ práce s tabulkami budete potřebovat kousek Javascriptu nebo zmíněný plugin.

Stohování se hodí i pro tabulky se složitějším obsahem v buňkách: odstavcový text, formulářové prvky a tak dále.

Změna směru tabulky

Často se stává, že se pro malé displeje hodí jiný směr zobrazení tabulky než u velkých. Podívejte se na obrázek, hned pochopíte.



Numbers	Names	Values	Dates	Cash Money	Messages	Buttons
000000001	Dr. Jayhawk	102	03/30/1940	$60.42	PAID	Select
000000002	Dr. Jayhawk	137	03/18/1953	$69.68	PAID	Select
000000003	Dr. Wolverine Longer Text Test	154	03/29/1976	$86.68	PAID	Select
000000004	Dr. Tarheel	113	03/30/1981	$63.50	PAID	Select
000000005	Dr. Orange	147	03/30/1987	$74.73	PAID	Select
000000006	Dr. Who	000	04/08/2013	$0.00	PENDING	Select

Změna směru tabulky. Vyzkoušejte si to na CodePenu. cdpn.io/rjmyx

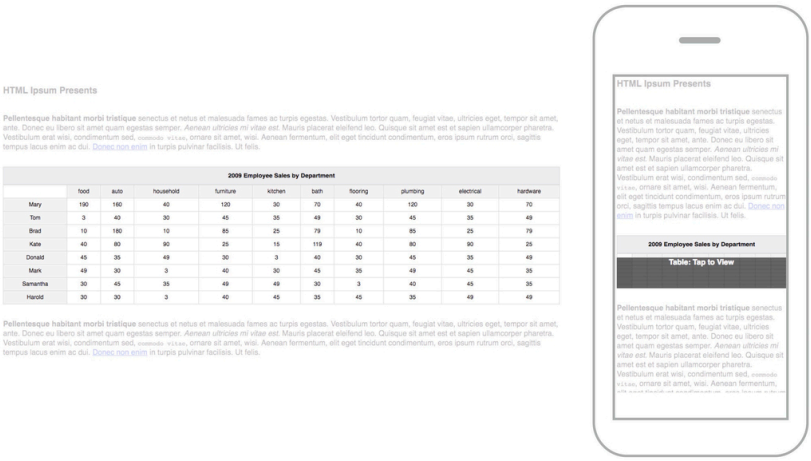
Nasazení doporučuji u tabulek, které mají velký počet řádků, ale málo sloupců.

Tak. Řekněme, že nejčastější scénáře jsme vyčerpali. Pojďme se ale podívat i na exotičtější situace.

Odkaz na plnou tabulku

Na mobilech můžete samozřejmě tabulku hodně zjednodušit a přiložit odkaz na plnou verzi.

Za fajn nápad také považuji vložit do stránky namísto tabulky jen jakýsi zástupný symbol. Vidíte to na obrázku a zkoumat můžete v přiložené ukázce. jsbin.com/apane6/14



Tabulku na mobilu uprostřed obsahu nahradíme zástupným symbolem. Kliknutím se zobrazí plná verze

Kdy se hodí? Pro složité tabulky uprostřed jiného obsahu, kde ostatní scénáře (stohování, fixní sloupec) selhávají.

Varianta s reprezentací obsahu na mobilech grafem

Z tabulky prostě na mobilu uděláte zjednodušený graf. Doporučuji nasazovat v kombinaci s odkazem na plnou verzi tabulky. Hodí se opět pro situace se strašlivě komplikovanými daty bez výrazné obsahové hierarchie.

Na mobilech něco vynechat. Ale opravdu?

„Prostě schováme pár sloupečků, a na mobil se to vejde.“ Tahle varianta vypadá, že se sama nabízí. Z technického pohledu budiž. Jenže designér ve mně zvedá obočí i ukazováček, aby vás upozornil na možné nevýhody.

Schovávání obsahu na konkrétních zařízeních je dost nebezpečné.

Jak už jsem argumentoval dříve, stejní lidé se na vaše rozhraní dívají z různých zařízení. Proč by určitý obsah měli na jednom zařízení vidět a jiném ne?

Připomenu to znovu v textu o častých chybách responzivních webů v rámci sedmé kapitoly.

Responzivní grafy

Responzivní grafy nacházejí využití hlavně v rukou demagogických politiků. Grafy, které používají, se *přizpůsobují* jejich vidění světa.

OK, nebudu vám kazit krásné chvíle s mými texty těmito rádoby vtipnými odbočkami.

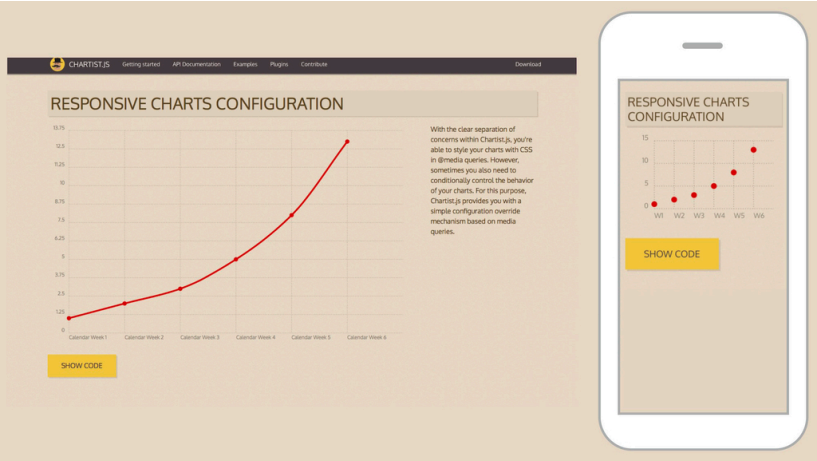
Budeme se bavit o opravdových grafech v opravdových responzivních stránkách. Zase tak často se nepoužívají, proto to vezmu letem světem.

Pokud s grafy pracujete často, pak to hlavní co byste si měli odnést je: V responzivním rozhraní nestačí grafy přizpůsobit šířkou layoutu stránky. Obvykle je potřeba změnit hustotu dat. Někdy dokonce připravit zcela jiné zobrazení pro malé a jiné pro velké displeje.

Poradím alespoň několik knihoven, které to s responzivními grafy umí.

Chartist.js

Z mého pohledu nejzajímavější knihovna pro jednodušší grafy. Protože používá formát SVG, umí kromě přizpůsobení velikosti okna také přizpůsobovat obsah grafů. gionkunz.github.io/chartist-js/



Chartist.js se umí menším displejům přizpůsobovat nejen velikostně, ale i hustotou dat a typem zobrazení

Highcharts

Nějaké responzivní možnosti má i tahle populární grafová knihovna. Spíše se ale jedná o přizpůsobení velikosti než o adekvátní změny v obsahu a hustotě formy či obsahu.
highcharts.com/demo/responsive

Chart.js

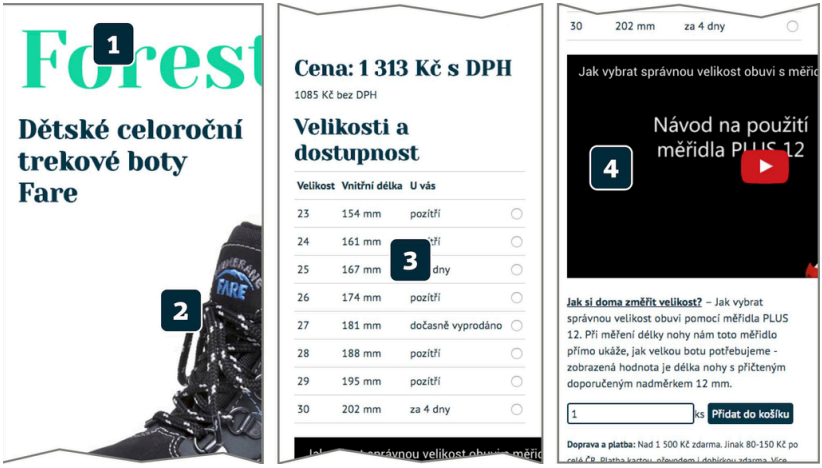
Velmi populární knihovna, ale grafy vykresluje do prvku `<canvas>`, takže s responzivitou to bude horší. Canvas totiž není elegantně vektorový jako SVG. Šířkou a výškou se ale grafy přizpůsobovat umí.
chartjs.org

Tím jsme se dostali k poslednímu textu kapitoly o mediálním obsahu ve stránkách. Pojďme se vrátit k příkladu. ForestKid.cz,

vzpomínáte?

Příklad: dokument s přizpůsobivými médii

Abychom použili vědomosti nasáté v téhle kapitole, vložil jsem do příkladu veškerý mediální obsah. Podívejme se teď na něj v rozlišení dnešních běžných mobilů.



Příklad před aplikováním přizpůsobivosti médií

Text se chová hezky, ale média nám vystrkují růžky, že ano? Žádný strach, nůžky na ně brát nebudeme. Prostě zařídíme, aby se začala chovat pružněji.

Pro nedočkavce je tady výsledek:

- Otevření v prohlížeči: vrdl.in/vdwdmed
- Stažení v ZIPu: vrdl.in/vdwdmedzip

Pojďme si teď lidsky popsat, co přesně se změnilo.

1. SVG logo

Na logo aplikujeme trik, který jsme se naučili v podkapitole o responzivním SVG. Odstraníme parametry `width` a `height`. Všechno necháme na jejich kolegovi: `viewBox`. Řešení hledejte přímo v `index.html`.

Pro správně pružné chování v Internet Exploreru ještě doplníme rodičovský kontejner a trik se zachováním poměru stran pomocí `padding-bottom`. To hledejte v souboru `style/ui/logo.css`.

2. Obrázky produktu

První krok je jednoduchý: pružné přizpůsobení velikosti okna. Toho dosáhneme kódem pro obrázky, který jsme se naučili na začátku kapitoly. Najdete jej v souboru `style/media/images.css`.

Druhý krok zajistí, aby se načítaly také správné varianty obrázků na různě velkých oknech a poměrech `device-pixel-ratio`.

Pojďme si tady projít celý proces, protože jinde v knize tuto část popisuji spíše obecně.

Vložíme obrázky v maximálním rozlišení

Obsahové obrázky webu se hodí ukládat v co největším rozlišení. Nikdy totiž nevíte, jak budou vypadat displeje budoucnosti. Nám se je podařilo ulovit v šířkách kolem dvou tisíc pixelů.

Najdeme nejmenší a největší velikost

Nejmenší velikost obrazovky aktuálních mobilů je 240 pixelů. Dnes už se moc nedělají, ale nějaký podíl na trhu ještě mají. V tomto rozlišení mají obrázky po odečtení okrajů šířku 192 pixelů. Zaokrouhlíme si to na 200 pixelů a to bude naše nejmenší varianta.

Maximální šířka layoutu je nastavená na `30em`, což je 540 pixelů. Kvůli „Retina“ displejům budeme počítat s dvojnásobkem, tedy 1080 pixelů. Obrázky je vhodné testovat i na zařízeních s více než dvojnásobným poměrem hardwarových a CSS pixelů, ale mám zkušenost, že dvojnásobek obvykle postačuje.

Vyrobíme varianty a uvedeme je do `srcset`

Jak jsem psal v textu o srcset a sizes, varianty generuji po dvě stě a tři sta pixelech. Můžu to zařídit nějakou automatizací na svém počítači nebo na serveru. Pro jednorázovou práci ale doporučuji výborný generátor variant obrázků „Responsive Image Breakpoints Generator“. responsivebreakpoints.com

Ten varianty chytře vytváří podle minimálního kroku datové velikosti. Když jsem nastavil 30kB, dostal jsem následující verze prvního obrázku:

Šířka v pixelech	Velikost v kB
200	12
442	43
617	72
762	100
903	129
1036	160
1080	180

Zapsáno v kódu to vypadá takto:

```
<img
  src="boty-1_617.jpg"
  srcset="
    boty-1_200.jpg 200w,
    boty-1_442.jpg 442w,
    boty-1_617.jpg 617w,
    boty-1_762.jpg 762w,
    boty-1_903.jpg 903w,
    boty-1_1036.jpg 1036w,
    boty-1_1080.jpg 1080w"
  alt="Dětské celoroční trekové boty Fare - pohled ze strany"
  width="520" height="520">
```

Pro další dva obrázky to bude vypadat trochu jinak. Podívejte se pak do HTML zdroje nebo na odpovídající commit na Githubu. git.io/vDVjw

Nastavíme velikost obrázku v layoutu: `sizes`

Layout a velikost obrázků v něm jsou v tuto chvíli jednoduše zjistitelné. Stačí vzít vývojářské nástroje, zmenšit okno a postupně ho zvětšovat. Všechny body zlomu layoutu tam krásně uvidíte a z vývojářských nástrojů snadno vyčtete patřičné rozměry.

- Na malých displejích zabírá layout celou obrazovku bez postranních okrajů a obrázek jakbysmet: `calc(100vw - 2 * 1.5rem)`.
- Od šířky okna kolem 530 pixelů už se dále nezvětšuje. Šířka obrázku tam zůstává fixně na `480px`.
- Od 640 pixelů je zase obrázek široký `540px`.

Teď si tři typy layoutu zapišme do atributu `sizes`:

```
<img
  src="boty-1_617.jpg"
  srcset="…"
  sizes="
    (min-width: 640px) 540px,
    (min-width: 530px) 480px,
    calc(100vw - 2 * 1.5rem)"
  alt="…"
  width="520" height="520">
```

Prohlížeč uplatní první vyhovující pravidlo, proto jsem typy layoutu řadil od největších obrazovek po nejmenší.

3. Tabulka

V tomto rozlišení ještě vypadá hezky. Ale není tabulky, která by se někde nerozpadla. Stačilo by okno zmenšit o trochu více. Aplikujeme řešení pro posun do stran z podkapitoly o tabulkách.

Uvidíte to v souboru `style/media/rwd-table.css` a následujícím commitu. git.io/vDwJJ

4. Vkládané video

Stačí vzpomenout na „Pružné vkládané elementy se zachováním poměru stran“ ze začátku této kapitoly. A opět se o slovo hlásí trik s `padding-bottom`, který ostatně v responzivním designu budete potřebovat velmi často.

Tuto věc má v našem případě na starost komponenta `style/media/rwd-object.css`.

Aktuální stav příkladu si můžete naživo prohlédnout nebo stáhnout na následujících adresách.

- Otevření v prohlížeči: vrdl.in/vdwdmed
- Stažení v ZIPu: vrdl.in/vdwdmedzip

Dostali jsme se tedy do stavu naformátovaného dokumentu s ošetřeným mediálním obsahem. Teď už pojďme navrhnout pokročilejší prvky uživatelského rozhraní. A pak i layout. Nejdřív ale znalosti, které byste měli mít, pokud si na to chcete troufnout.

Zapamatujte si

- Trik s `padding-bottom` umožní zachovat poměr stran jakéhokoliv elementu.
- Grafiku, kterou lze vyjádřit vektorem, vkládejte do webu v SVG.
- `<img src="">` vám pro bitmapové obrázky stačit nebude.
- Zvažte využití formátu WebP.
- Do atributu `srcset` značky `<img>` vkládejte seznam předpřipravených variant obrázků. Do `sizes` patří popis breakpointů layoutu webu.
- Značka `<picture>` slouží hlavně pro posílání různých ořezů a souborových typů na různá zařízení.
- SVG je dokument, ne obrázek.
- Ve zdrojovém kódu SVG nastavujte `viewbox`, nikoliv `width` a `height`.
- Uvnitř SVG můžete používat Media Queries. Je to fajn.
- S responzivními tabulkami vám pomůže knihovna Tablesaw.

Díky za přečtení ukázky!

Celou knihu můžete zakoupit na Vzhůru dolů.

Zdraví vás
Martin Michálek
martin@vzhurudolu.cz