22 nových vlastností
od kulatých rohů po
Flexbox layouty.

Vzhůru do CSS3

Martin Michálek

Vzhůru do CSS3 (verze 1.4)

Na úvod

Děkuji!

Ještě jednou díky za zakoupení ebooku „Vzhůru do CSS3“. Vážím si vaší důvěry. Díky ní mohu ebook dále vylepšovat a rozšiřovat.

Pokud se k vám náhodou dostal jinou cestou, nebojte nic není ztraceno. Stále můžete dát vše do pořádku koupí licence na webu vrdl.cz/ebook.

Zpětná vazba

I když jsem udělal vše proto, abych se jim vyhnul, nezbývá než se smířit s tím, že i v tomto ebooku nějaké chyby zůstaly. Moc pomůže, když je nahlásíte. Všichni uživatelé dostanou jednou za čas aktualizovanou verzi.

Moc rád si přečtu i obecnou zpětnou vazbu nebo rovnou recenze ebooku. Pište na martin@vzhurudolu.cz.

Proč a pro koho?

Základy pro ebook „Vzhůru do CSS3“ pokládá leta aktualizovaná CSS3 příručka na webu Vzhůru dolů – vrdl.cz/prirucka/css3.

Text, který jste právě začali číst provádí vývojáře aktuálním stavem ve vývoji webového uživatelského rozhraní (UI).

Bude se hodit jako každodenní pomocník při používání vlastností v praxi hlavně začínajícím a mírně pokročilým. Zkušení ale ocení první podrobný český text o Flexboxu nebo úvodní shrnutí principů kódování pro současný web.

Zabýváme se kodéřinou, tedy UI vývojařinou. Nikoliv frontend programováním a Javascriptem.

O co tady půjde?

Text má dvě části:

O dnešním kódování webů – jak se vývoj webového uživatelského rozhraní v posledních letech změnil a jaké nástroje,

technologie a postupy se dnes hodí využívat.

Referenční příručka CSS3 vlastností – ukázky jejich využití a nejčastější špeky, na které si dát pozor v praxi.

Autor

Martin Michálek je frontend designér na volné noze. Navrhuje a implementuje uživatelská rozhraní responzivních webů, například pro VašeČočky.cz. Píše blog Vzhůru dolů a ebook Vzhůru do CSS3. Spoluzaložil a vede spolek Frontendisti.cz.

Pořádá školení se zaměřením na moderní webový frontend – CSS, responzivní design, rychlost načítání, framework Bootstrap, SVG nebo Javascript. vrdl.cz/kurzy

vrdl.cz/martin

Co je nového?

Čtete „Vzhůru do CSS3“ verze 1.4 z 5. ledna 2017. Tady je seznam obsahových novinek:

- Zgruntu přepsaná kapitola o Media Queries.
- Přepsaná a aktualizovaná kapitola o prohlížečích.
- CSS pixel a „Retina“ displeje podrobněji.
- Přepsaná kapitola o CSS prefixech.
- Dlouhé odkazy ven z ebooku jsou prohnané zkracovačem, aby nerušily text.
- Lepší kontrast písma v Kindle čtečkách.
- Drobné opravy a aktualizace (včetně odstranění pitomé chyby se špatným směrem `flex-direction` u flexboxu).

Hlášení případných chyb velmi vítám: martin@vzhurudolu.cz.
A teď už přeji příjemné čtení.

Kapitola I:
O dnešním kódování webů

Než se pustíme do samotných CSS3 vlastností, musíme se podívat na dnešní způsob vývoje webového uživatelského rozhraní.

Jak se v posledních letech změnily pracovní postupy? Docela hodně.

Jaké nástroje, technologie a postupy se dnes hodí využívat? Grunt, SVG, Bower, Node.js a mnoho dalších.

Jste připraveni na lavinu nových pojmů?

Proměny kodéřiny

Jako každá profese, i vývojařina webového rozhraní reflektuje změny prostředí, ve kterém se pohybuje. To, co se v posledních letech změnilo v prostředí webového kodéra, se dá shrnout do osmi bodů.

1. Příchod mobilů
2. Responzivní webdesign = webdesign
3. Mnoho prohlížečů, kodérova smrt
4. CSS pixel
5. SVG a srcset/sizes, nové výzvy pro vkládání obrázků
6. Komplikace s prefixy
7. Méně Photoshopu, více kódu
8. Od ornamentů k typografii a dál

Příchod mobilů

Bezpochyby nejvýznamnější změna. V jistém smyslu matka většiny ostatních změn. Doufám, že o významu mobilních zařízení pro práci webaře už nikdo nepochybuje.

V lednu 2017 je průměr návštěvnosti velkých českých webů z mobilních zařízení na 20 procentech. Každoročně se ale podíl zdvojnásobuje. Vycházím z čísel, která poskytuje Gemius na rankings.cz, ale podívejte se do statistik vašeho webu nebo aplikace.

Na západ od našich hranic musejí webaři počítat často s nadpoloviční návštěvností z mobilů.

Mobilům neutečeme.

Responzivní webdesign = webdesign

> Vše co je dnes navrhováno do fixních rozměrů a neresponzivně není webdesign. Pokud nechcete přijmout proměnlivost vlastní webu, nejste webdesignér. Webdesign je responzivní design, responzivní webdesign je webdesign dělaný správně.

– Andy Clarke v článku „I don't care about Responsive Web Design", vrdl.in/xozn1

Příchod mobilů je možné řešit speciálním mobilním webem — „m tečka" verzí se samostatnými šablonami. Jak už ale mnozí z nás zjistili, je to jak pro návštěvníka, tak pro provozovatele webu dlouhodobě nevýhodné.

Odvažuji si tvrdit, že dříve či později všichni skončíme u jednotné, responzivní verze webu. Přívlastek „responzivní" tedy časem ztratí smysl.

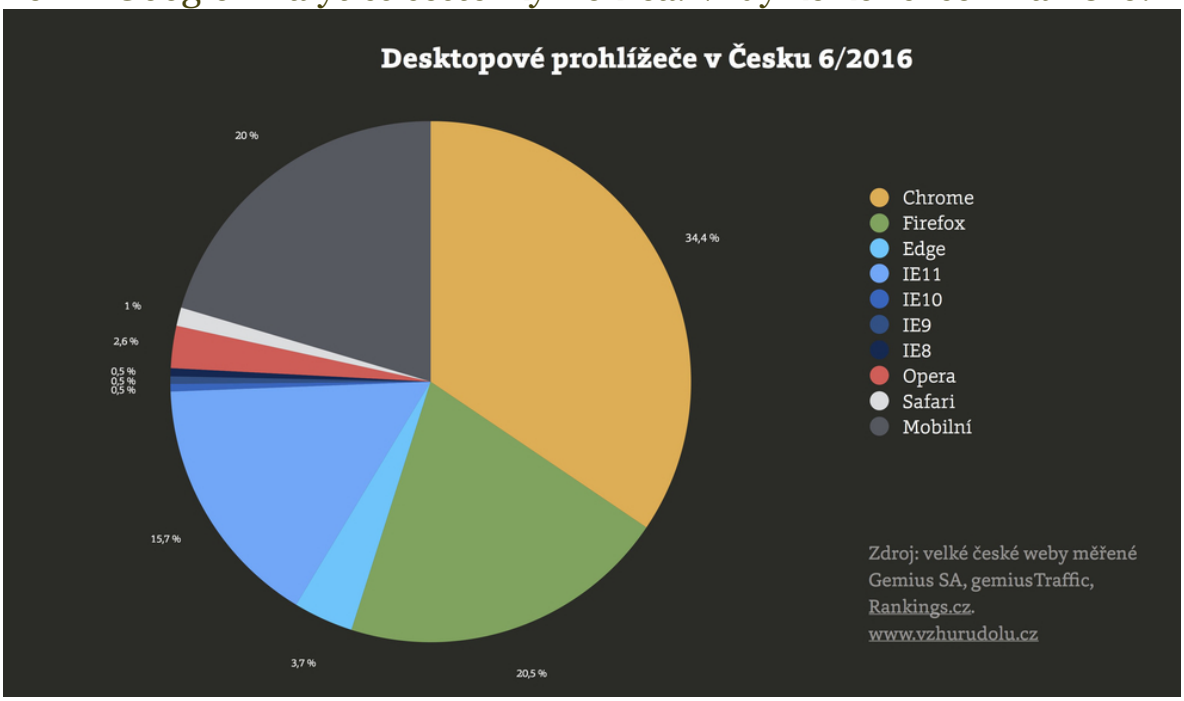
Responzivní je tedy z jistého pohledu označení pro aktuální přerodovou fázi webdesignu.

Na první pohled nám z CSS3 vlastností s responzivním designem pomáhají jen Media Queries, ale nenechme se mýlit. Nějakým způsobem je z pohledu responzivního návrhu výhodná prakticky každá CSS vlastnost z druhé části tohoto textu.

Mnoho prohlížečů, kodérova smrt!

Vstupem Chrome na desktop a nástupem mobilů začaly nové Browser Wars. Máme tady minimálně 16 prohlížečů. Ano, z tohoto pohledu už kodéři zažívali lepší časy.

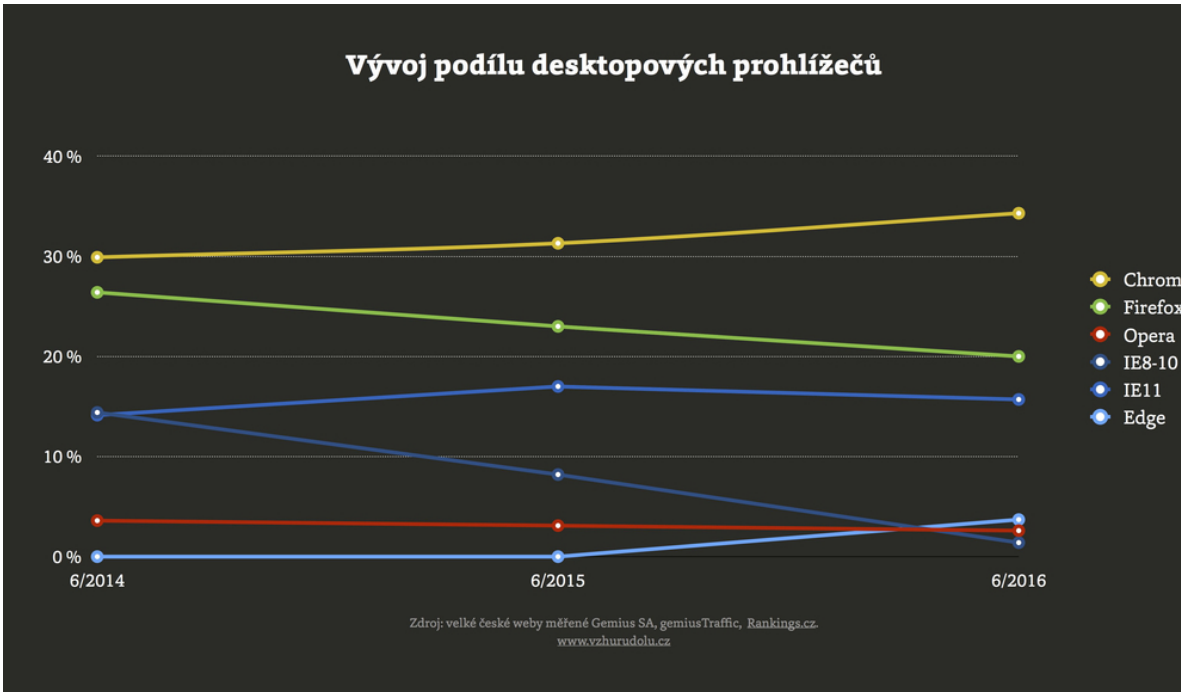
Pojďme si tady shrnout aktuální stav trhu prohlížečů v ČR. Čísla v textu jsou vytažená z měření Gemius SA na Rankings.cz a u mobilních zařízení z Google Analytics cestovky Rekrea. Vždy ke konci června 2016.



Nejprve tři zásadní fakta z poslední doby:

1. **Chrome požírá trh.**
   Už nyní se v Česku dostal přes třetinu shlédnutých stránek. Ostatní prohlížeče stagnují nebo klesají.
2. **Staré Explorery už skoro vymřely.**
   Každý z Internet Explorerů kromě verze 11 už má podíl pod půl procenta a rychle klesá. Dnes je to poprvé co vám vyloženě nedoporučím pro ně dělat pracnější fallbacky.
3. **Mobilní zařízení mají přes pětinu celku.**
   A více než polovina shlédnutí stránek na mobilech je opět z Chrome.

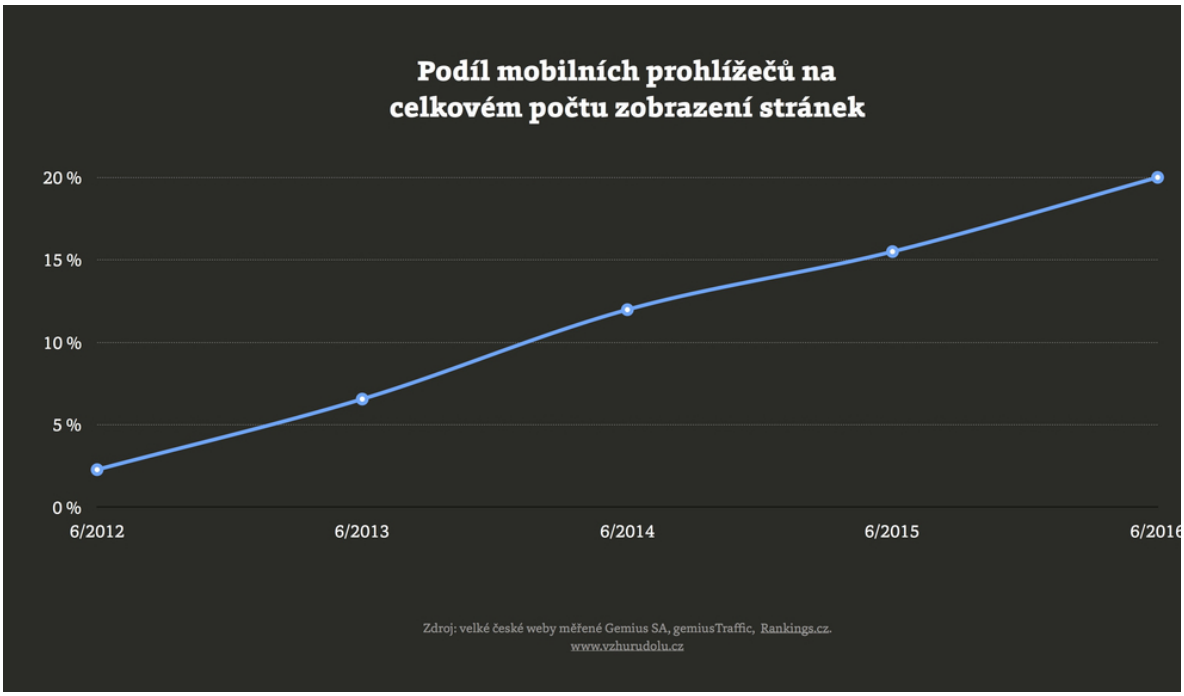
Desktop: Chrome jako jediný masivně roste. Na desktopu má více než třetinový podíl



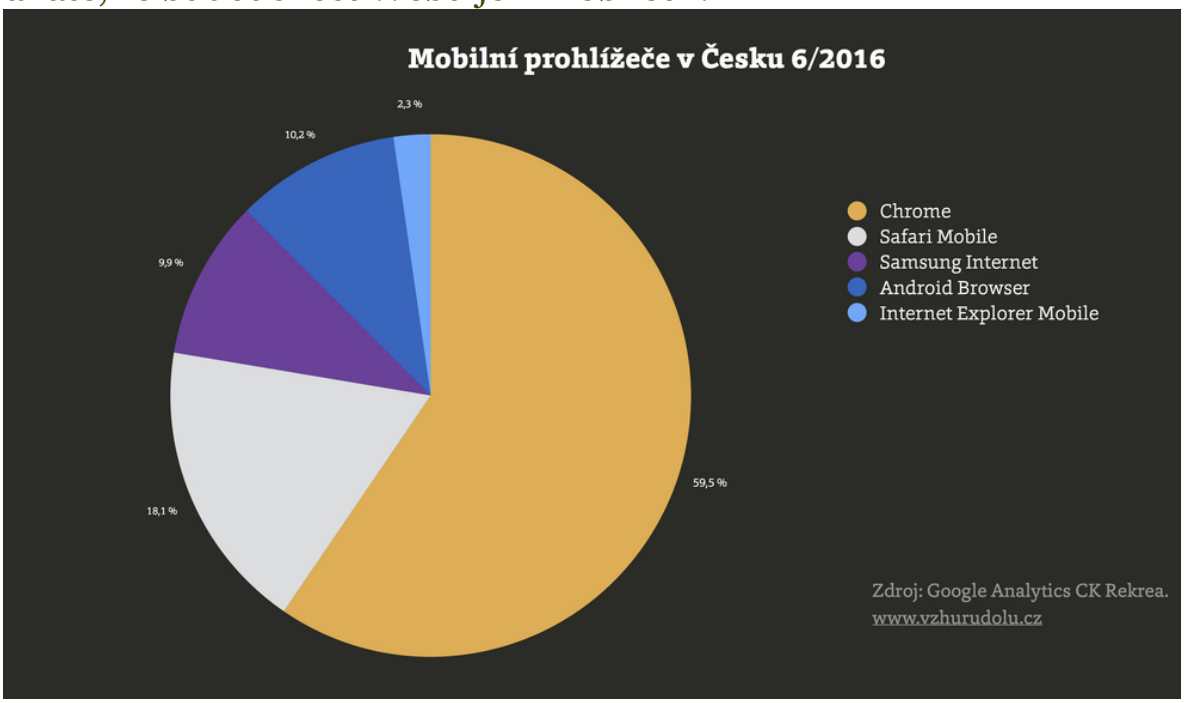
Poznámky:

- V uvedeném období přestal klesat Firefox. Držím palce, aby nešlo o výjimku, protože konkurence je pro Chrome potřeba jako sůl.
- Konec starých Internet Explorerů (IE) je tady. Už dlouho se to týká verzí 9 a 10, teď už konečně i osmičky.
- Edge, nový moderní prohlížeč od Microsoftu, roste. Ale pomalu.
- Opera se dlouho držela kolem čtyřprocentního podílu. V posledním roce ale také začala klesat.
- Podíl Safari na Apple počítačích v ČR k mému překvapení mírně roste.

Mobily: pětinový podíl na shlédnutích stránek a dominance Chrome



Na pětinovém podílu ze shlédnutých stránek se mobily samozřejmě nezastaví. Za rok budou mít minimálně čtvrtinu a tak dále. Čísla ze Západu ukazují, že to půjde přes polovinu a výše. Ale vy už jistě dávno neváháte, že budoucnost Webu je v mobilech.



Podíly mobilních prohlížečů mám z Google Analytics cestovky Rekrea, protože Rankings.cz neumí rozumně rozeznat jednotlivé mobilní prohlížeče.

Data k relativně novému prohlížeči Samsung Internet odhaduji. V Analytics jsou schovaná za čísly mobilního Chrome. vrdl.cz/blog/71-samsung-internet

Mezi mobilními prohlížeči tedy vede Chrome Mobile s 18,1 procenty podílu. Druhé je Safari Mobile (5,5 %), třetí Android Browser (3,1 %). Dle mých odhadů následuje Samsung Internet (kolem 3 %), Internet Explorer Mobile (0,7 %) a další méně významné prohlížeče jako Opery a zatím i mobilní Edge.

Mobilní Chrome samozřejmě v budoucnu statistiky ovládne ještě výrazněji, protože je to nyní hlavní prohlížeč na Androidu. Ano, na té platformě, která dnes vlastně na mobilech nemá masově zaměřenou konkurenci.

K jednotlivým prohlížečům: ke dnešku je jich minimálně 16

Ke krátkému komentáři pro zajímavost přidávám i skóre na HTML5test.com. To udává podporu moderních HTML5 technologií. Čím vyšší, tím lepší.

9 PROHLÍŽEČŮ, SE KTERÝMI PROSTĚ MUSÍTE POČÍTAT

- **Chrome**
  Ten, co všechno sní. Celosvětově podle má podle Statcounter.com už k šedesátiprocentnímu podílu. HTML5test.com: 492/555 (verze 52)
- **Firefox**
  Celosvětově mírně klesá. Teď má na světě podle Statcounter podíl 14 %. Mozille nelze upřít snahu. Firefox chce zrychlit, zavřela boční projekty a přichází s drobnými inovacemi. Jenže Google má v případě Chrome z jejich pohledu dost nešťasnou kombinaci *umu* vyrábět výborný prohlížeč a *síly* cokoliv protlačit. HTML5test.com: 461/555 (verze 48)
- **IE 11**
  Podle vývoje za poslední dva roky ubrala relativně moderní jedenácka podíl ostatním prohlížečům od Microsoftu. Přepokládám, že teď začne klesat ve prospěch Edge a ostatních prohlížečů. HTML5test.com: 312/555

- **Edge**
  Moderní prohlížeč od Microsoftu roste méně než bych čekal. HTML5test.com: 460/555 (verze 14)
- **Opera** Od verze 15 je Opera postavená na stejném jádru jako Chrome, takže s testováním na ní tolik práce není. Tedy, ne že by se dalo úplně vynechat. Nějaké rozdíly tam jsou. Opera každopádně mírně klesá. HTML5test.com: 496/555 (verze 40)
- **Safari**
  Z moderních prohlížečů je to dnes největší *brzda*. Ještě nedávno byl WebKit synonymem inovací na webu, pamatujete? Aktualizuje se klasicky až s verzemi operačního systému. HTML5test.com: 370/555 (verze 9.1)
- **Chrome Mobile**
  Na mobilech zatím jednoznačně kraluje. Jedinou konkurenci má v iOS zařízeních od Apple a v možném nástupu prohlížeče od Samsungu. U Chrome na iOS pozor. Je to jen pseudoprohlížeč, tedy jiné rozhraní pro mobilní Safari. Píšu o tom dále.
  HTML5test.com: 486/555 (verze 52)
- **Samsung Internet**
  Nový prohlížeč od Samsungu, který se ve statistkách (včetně Google Analytics) aktuálně schovává pod mobilní Chrome. Moc o něm nevíme, ale Samsung zařízení patří mezi nejprodávanější i v ČR, takže s ním nějak musíme začít počítat. V mobilech má ikonu fialové zeměkoule. vrdl.cz/blog/71-samsung-internet
- **Android Browser**
  Starší prohlížeč postavený na Webkit jádře. Modrá zeměkoule s nápisem Internet. Týká se Androidů ve verzích 4.x. Často jej upravovali výrobci zařízení, takže ho můžete znát třeba i pod jinými názvy. V téhle rodině prohlížečů je pěkný galimatyáš. Už se myslím ale nevyvíjí. slides.com/html5test/the-android-browser
  HTML5test.com: 356/555 (verze 30)
- **Safari Mobile**
  Jediné vykreslovací jádro dostupné na iOS. Všechny tamní prohlížeče, včetně Chrome, používají pro renderování stránek Safari. HTML5test.com: 378/555 (verze 9.3)

SPECIÁLNÍ KATEGORIE: WEBVIEW A PROHLÍŽEČE UVNITŘ APLIKACÍ

Většina dnešních shlédnutí webů na mobilech se neodehrává vědomým spuštěním prohlížeče, ale otevřením stránky kliknutím na odkaz uvnitř aplikací. Ve Facebooku, Twitteru nebo třeba emailové apce. Dříve jsem o tom podrobně psal na Vzhůru dolů. vrdl.cz/blog/19-prohlizec-facebook

Jakým prohlížečem se pak stránka vykreslí? Vývojáři nativních mobilních aplikací jej znají jako WebView komponentu. Ta startuje jádro výchozího prohlížeče pro konkrétní operační systém. Na iOS je to vždy mobilní Safari, na dnešních Androidech obvykle Chrome.

I v těchto kontextech doporučuji weby testovat. Prohlížeče tam mívají trochu jiné uspořádání ovládacích prvků a některé funkce prostě neumí.

VYMÍRAJÍCÍ NEBO ZATÍM SLABĚ ZASTOUPENÉ PROHLÍŽEČE

Weby je možné a slušné vyrobit tak, aby se zásadně nerozsypaly ani v méně obvyklých browserech. Rozhodně ale nedoporučuji trvat na plnohodnotném zobrazení v nich. Náhradních řešení se pro ně dá vymyslet celá řada.

- **IE 8**
  Děkujeme, ale už opravdu odejdi. Vypadá to, že jeho éra právě končí spolu s Windows XP.
  HTML5test.com: 33/555
- **IE 9**
  IE9 běží na systémech, kde lze obvykle aktualizovat na novější verzi.
  HTML5test.com: 113/555
- **IE 10**
  Také IE10 běží na systémech, kde lze obvykle aktualizovat na novější verzi. Tipuji, že IE8, 9 i 10 v roce 2017 vymřou úplně.
  HTML5test.com: 265/555
- **Internet Explorery Mobile**
  Výchozí prohlížeče na Windows Phone verzí 7 a 8. V Analytics u relevantní projektů pod jednoprocentním podílem trhu.
  HTML5test.com: 310/555 (verze 11 na Windows Phone 8.1)

- **Edge Mobile**
  Výchozí prohlížeč na mobilních Windows 10. V Analytics vidím u cestovky s průměrnou českou návštěvností kolem 0,2 % podílu. Ale poroste.
  HTML5test.com: 444/555 (Windows Phone 10)
- **Opera Mobile** Běžná mobilní Opera s jádrem Blink. Může mít asi 0,3 % podílu.
  HTML5test.com: 481/555 (verze 37)
- **Opera Mini**
  Proxy prohlížeč bez vlastního renderovacího jádra co dokáže šetřit datový objem, ale weby renderuje ošklivě. Dříve populární, dnes už v ČR zapadá. U cestovky vidím 0,1 % návštěv.

PROHLÍŽEČE S MENŠÍM NEŽ PĚTIPROCENTNÍM PODÍLEM TVOŘÍ ASI 15 % TRHU

Jak už jsem na Vzhůru dolů psal, pětiprocentní nebo jiná hranice pro podporu prohlížečů je velmi zrádná. Prohlížeče pod touto hranicí teď tvoří kolem šestiny pageviews. Čtěte „pohádku o pěti procentech“. vrdl.cz/blog/20-pet-procent

Podporujte prostě různé prohlížeče různým způsobem a vynakládejte na to energii, které odpovídá byznys hodnotě jejich uživatelů s výhledem do budoucna.

Renderovací jádra: vede samozřejmě Blink

K polovině prázdnin 2016 má jádro Blink (Chrome, Opera) podíl na trhu 48 procent. Gecko (Firefox) 20 %, Trident (Internet Explorer) 18,3 %, WebKit/KHTML (Safari) 9,0 % , EdgeHTML (Edge) 3,7 % a už nevyvíjené Presto (Opera 12-) 0,5 %.

Závěrečná doporučení pro webaře

- **Moje obecná čísla berte s rezervou**. Sledujte hlavně vlastní Google Analytics.
- **Nebojte se nových technologií**. Flexbox má v těchto číslech přibližně 98 % podporu. Totéž SVG. Obojí vám děsně ušetří práci a nabídne nové možnosti. Fallbacky ve starých prohlížečích

rozhodně nedělejte plnohodnotné se zobrazením v moderních prohlížečích. Obvykle se vám to nevyplatí.

- **Nepodceňujte menší prohlížeče**. Zařízují sedminu shlédnutí stránek. Naučte se testovat tak, abyste s tím neměli moc práce. Doporučím zase svůj článek. vrdl.cz/prirucka/jak-testovat-responzivni-weby
- … a raději doslovně pro méně zkušené: Pokud by vás snad napadlo, že Web se prohlíží hlavně na Chrome a pak trochu Firefoxu, ošklivě se klamete.

„Retina“ displeje a CSS pixel

CSS pixel. Referenční pixel. Ať už tomu říkáme jakkoliv, pixel už hold není co za našeho mládí býval.

Na to, jak je CSS pixel v dnešním webdesignu důležitý, je pohříchu málo známý. Proto tady začneme úplně od základů.

Hardwarové rozlišení nás moc nezajímá

Retina, Amoled, QuadHD. Asi jste si všimli, že mobilní zařízení mají v poslední době dost šílená rozlišení. A trend u mobilních zařízení nekončí. Podívejte se na MacBook Pro s Retina displejem nebo další laptopy.

Kouká se přes ně na weby dobře, což o to. Ale kdo má pro ně ty weby navrhovat?!

Vezměme iPhone 5S, ten má rozlišení 640×1136 pixelů. Někteří webaři se pořád ještě čertí: „Když si někdo otočí iPhone na šířku, zobrazí se mu web v na malém zařízení web v rozlišení pro tablet. Hrůza!“

Není to tak. Hardwarové pixely nás webaře totiž skoro nezajímají. Zato CSS pixely jsou naši kamarádi.

CSS versus hardwarové rozlišení

Autorům webů totiž prohlížeče hardwarové rozlišení přepočítají do takzvaného CSS rozlišení.

V případě iPhone 5S to bude polovina, tedy 320×568 pixelů. To už je docela normální „mobilní“ rozlišení, že?

Retina displej na iPhone má tedy poměr mezi CSS a hardwarovým rozlišením 1:2. Ale pozor, když v CSS vykreslíme objekt velký 1 pixel, bude zabírat 4 hardwarové pixely. Půjde o mřížku o velikosti 2×2 pixely, odtud ten poměr 1:2.

Když tedy do stránky vložíme obrázek…

```
<img src="fotka.jpg"
 width="100" height="100"
 alt="Fotka z dovolené">
```

…vykreslí se na Retina displeji v ploše 200×200 hardwarových pixelů.

A tady vznikají problémy. Prohlížeč totiž nebude mít fotografii v dostatečné kvalitě, protože vykresluje 100×100 velký obrázek na 200×200 mřížce. Na iPhone s Retina displejem pak naše úžasná fotografie z dovolené prostě nebude tak krásně ostrá. Kurnik šopa!

Zjednodušeně řečeno je tedy lepší fotografii rovnou vyrobit ve velikosti 200×200 pixelů. Do stránky ji ale vložíte stejným HTML kódem. Prohlížeč ji zmenší na 100×100 CSS pixelů, na běžných displejích nic nepoznáte a na Retina displeji to bude vypadat hezky.

Jenže to by znamenalo velké obrázky a neúměrnou datovou zátěž. Raději ještě chvilku čtěte.

Nejdříve ale ještě o tom, kde všude s CSS pixely pracují webaři. Stručná odpověď zní — všude.

Autoři stránek pracují jen s CSS pixely

Raději doslovně připomenu, když použijete správnou meta značku pro viewport, pak v HTML, CSS i Javascriptu vždy pracujeme s CSS pixely. K těm hardwarovým prostě jako vývojáři přístup nemáme.

Takže když napíšu následující dotaz na médium…

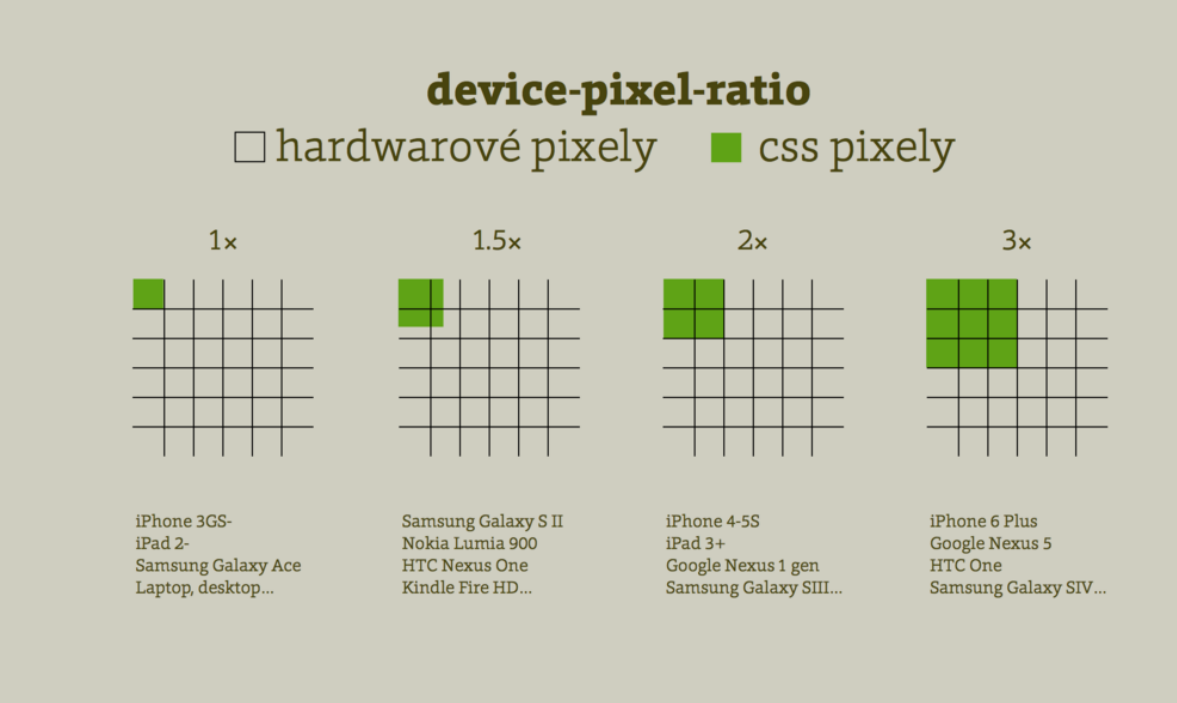
```
@media only screen and
(max-width: 600px) { … }
```

…cílím s jeho pomocí na rozlišení obrazovky nebo velikosti okna, které mají méně než 600 CSS pixelů. A cílím tedy i na zmiňovaný iPhone 5S.

Správnou meta značku pro viewport rozebírám v jiném článku. vrdl.cz/prirucka/viewport-meta

2×? 1.5×?! 1.325×! 2.37×! 3×! 4×…

Ještě mi rozumíte? Výborně, trochu to zkomplikujeme.



Poměr mezi hardwarovým a CSS rozlišením udává vlastnost `resolution` (dříve `device-pixel-ratio`. Mimochodem, pomocí dotazu na média je možné zacílit zařízení s displeji v určitém poměru i v CSS:

```
@media only screen and
(min-resolution: 2dppx) { … }
```

Jenže tady se děje další častá chyba webařů. Jejich responzivní weby počítají jen s poměrem 1:2, technicky řečeno `resolution: 2dppx`.

Existují zařízení s poměry 1,5; 1,325; nebo třeba 2,37. A ne vždy na nich obrázek ve dvojnásobném rozlišení vypadá uspokojivě, zejména pokud jde o ikonku.

Dnes už jsou běžné 3× a 4× displeje. Tam ani obrázek ve dvojnásobném rozlišení stačit vždycky nebude.

Vezměme si pár oblíbených zařízení. Jaký je tam poměr mezi HW (hardwarovým) a CSS rozlišením?

Poměr mezi HW a CSS rozlišením u vybraných zařízení

Zařízení	HW rozlišení	CSS rozlišení	device-pixel-ratio
iPad Mini	768 × 1024	768 × 1024	1
iPhone 4	640 × 960	320 × 480	2
Google Nexus 7	800 × 1280	604 × 966	1,325
HTC One	1080 × 1920	360 × 640	3
Xiaomi Mi3	1080 × 1920	270 × 480	4

Další zařízení najdete na canbike.org/CSSpixels.

Ježíši, to je průšvih, co? Budeme vytvářet obrázky pro každé `device-pixel-ratio`?

Nějak se to řešit dá, nebojte. Jen člověk musí opustit staré zvyky.

SVG a srcset/sizes, nové výzvy pro vkládání obrázků

Díky CSS pixelu a snaze posílat uživateli ve stránce datově co nejúspornější materiál řešíme daleko více otázek než jen — „Mám použít PNG, nebo JPG?“.

Obrázky užívané ve stránkách si nejdříve ale musíme rozdělit do dvou kategorií – obrázky v rozhraní a v obsahu.

Obrázky v rozhraní: ikony, logotypy, dekorace

Tady je rozhodně jedinou udržitelnou cestou použít vektorovou grafiku.

Takzvané ikonfonty – tedy ikony vkládané jako znaky ve speciálních webových fontech – považuji za dobré, ale spíše dočasné řešení. Hlavně proto, že fonty nebyly navrženy jako náhrada vektorové grafiky. S ikonfonty jsou spojené problémy s vykreslováním, ale ani práce s nimi nebývá příliš pohodlná.

Zajímavější možnosti nabízí vektorový formát SVG. vrdl.cz/prirucka/svg

Pro dekorace v rozhraní (vlastní stíny, vlastní vzhled tlačítek nebo rámečků…) je určitě nejvýhodnější využít možností CSS3, alternativně opět SVG.

Obsahové obrázky: fotografie

Fotky samozřejmě můžete připravit v ohromném rozlišení – klidně více než čtyřnásobném – a v HTML kódu stránky zmenšit.

Vyřešíte problém s `device-pixel-ratio`, ale nárůst datového objemu stránky bude tak šílený, že vás uživatelé brzy jistojistě přijdou ubít svými chytrými telefony. Připomínám, že fotka připravená pro Retina displej (2×) neobsahuje 2×, ale 4× více pixelů, takže její datový objem naroste klidně čtyřnásobně.

V praxi preferuji řešení pomocí nových atributů tagu `<img>` — `srcset` a `sizes`. vrdl.cz/prirucka/srcset-sizes

Komplikace s prefixy

Předpony experimentálních implementací CSS vlastností od výrobců prohlížečů.

Nebo také podivná věc, která nás nutí duplikovat CSS kód. Dnes už prefixy nepáchají tak moc komplikací, ale kvůli novým vlastnostem jako je flexbox si s nimi nějak poradit musíme.

Pojďme si to ukázat na příkladu. CSS3 transformaci otočením bychom v době největší slávy prefixů měli zapsat takhle:

```
 .box {
  /* Chrome, Safari 3.1+: */
  -webkit-transform: rotate(7.5deg);
  /* Firefox 3.5-15: */
  -moz-transform: rotate(7.5deg);
  /* IE 9: */
  -ms-transform: rotate(7.5deg);
  /* Opera 10.50-12.00: */
  -o-transform: rotate(7.5deg);
  /* Firefox 16+, IE 10+, Opera 12.10+: */
  transform: rotate(7.5deg);
}
```

Dnes už to tak složité není. Podívejte se ostatně na web „Should I Prefix“, který skladuje aktuální potřebu prefixů pro jednotlivé vlastnosti stylů. shouldiprefix.com

Na posledním místě by měla být neprefixovaná a tedy standardní verze zápisu. I v situaci, kdy ji aktuální verze prohlížečů nepodporují a je pravděpodobné, že jednou budou. Bezprefixová implementace bude totiž lepší než prefixová. I proto je nutné ji mít na závěr, přebíjí tu starší a horší, pokud se v některé verzi prohlížeče vyskytují obě najednou.

Typy prefixů

Říká se jim občas „vendor prefixy“. Vendor je dodavatel software, v tomto případě jádra prohlížeče.

- `-webkit`. Prohlížečové jádro Webkit v tuhle chvíli používá Safari včetně iOS verze a některé menší prohlížeče. Dříve také Google Chrome a od verze 15 i Opery. Poslední dva jmenovaní ovšem odešli k vlastnímu jádru Blink.
- `-moz`. Gecko je jádro všech verzí Firefoxu.
- Prefix `-ms` používá hádejte kdo… Microsoft, přeci.
- Prefix `-o` se vztahuje k aktuálně již neexistujícímu jádru Opery jménem Presto.

Minulost a budoucnost prefixů

Výrobci prohlížečů jsou v posledních letech docela horliví při implementaci nových vlastností z HTML5 a nových verzí CSS. Implementují pak i vlastnosti, které W3C zatím neposlalo do finálních fází procesu standardizace.

Implementují tedy něco, co nemá hotový standard. Je to tedy zkušební, prototypová implementace. Aby ji odlišili od finální implementace, zavedli vendor prefixy. Detailně proces vzniku prefixů kdysi hezky popsal Martin Hassman. vrdl.in/i4s8m

Během roku 2016 ovšem vyvrcholila snaha výrobců prohlížečů a tvůrců standardů o nový přístup. Standardy se už nějakou dobu vyvíjejí v menších modulech, takže jdou vpřed rychleji. Z toho kromě jiného vyplývá, že prefixy přestávají být potřeba.

Nové vlastnosti teď výrobci prohlížečů přidávají pod „vlaječky“ (flags), skrytá nastavení. Můžete si je snadno zkusit otevřením `chrome:flags` v Chrome nebo třeba `about:config` ve Firefoxu. Bližší vysvětlení přechodu od prefixů k vlaječkám je na Mozilla Developer Network. vrdl.in/akgm5

Jak si s prefixy poradit? Autoprefixerem

Autoprefixer je nástroj, kterému řeknete jaké prohlížeče chcete podporovat. On se pak podívá do vašeho CSS kódu a doplní prefixy. Nejlépe se používá automaticky, v kombinaci s Gruntem nebo podobným nástrojem. github.com/postcss/autoprefixer

Alternativně je možné prefixy nechat doplňovat editorem kódu nebo CSS preprocesorem. To je ale postrádá kouzlo Autoprefixeru, který sám aktualizuje databázi prefixů podle jejich aktuální podpory v prohlížečích.

Méně Photoshopu, více kódu

Jeden z největších omylů webdesignu? Photoshop jako hlavní tvůrčí nástroj. Mám radost, že se s příchodem responzivního designu opět rozšiřuje názor, že Photoshop nemá mít v procesu návrhu webu tak silné slovo. A doufám, že kódování HTML/CSS podle PSD brzy skončí v muzeu vedle ruchadla bratranců Veverkových.

Jak to souvisí s CSS3? Asi jako Bolek s Lolkem. Velmi.

Od návrhu ke kódu daleko rychleji

Dříve se veškerá webová grafika, která nešla vyjádřit v CSS, exportovala z Photoshopu v podobě obrázků. CSS3 přineslo možnosti, jak kódem popsat nejen statickou grafiku, ale také jednodušší tvorbu interakcí.

A pokud jde něco vyjádřit kódem, lze snadno propojit vizuální tvorbu s praktickým použitím v prohlížečích. Máme tu nadějné pokusy v podobě editorů jako Sketch, Adobe Edge Reflow nebo WebFlow. Ty buď rovnou exportují kód či kousky kódu, nebo jsou z pohledu workflow lépe uzpůsobeny potřebám dnešního webdesignu – například podporou SVG grafiky nebo responzivního návrhu webů.

Kodérům zase ohromně šetří práci nástroje jako Brackets nebo Avocode, které problematiku řeší z jejich pohledu a část kódu ze zdrojové grafiky vygenerují automaticky.

Komponentový webdesign

Další vliv mají UI knihovny typu Bootstrap. Díky nim se programátoři pro tvorbu uživatelského rozhraní interních webových aplikací obejdou bez grafika, a dokonce bez vývojáře rozhraní.

Týmy, jež pracují na vlastním webovém produktu, zase zjistily, že nejefektivněji půjdou dopředu nikoliv kreslením dalších a dalších obrazovek ve Photoshopu, ale vytvořením vlastního Bootstrapu. Rozhraní webů a hlavně webových aplikací je možné pojmout jako skládanku samostatných komponent. Znovupoužitelných lego kostek.

Jednou z nejnadějnějších metodik komponentového přístupu k designu webového uživatelského rozhraní je atomický design a nástroj

Pattern Lab. vrdl.cz/prirucka/pattern-lab

V ČR mezi průkopníky tohoto přístupu patří například LMC se svým Jobs UI. jobs.cz/ui

Jako frontend designér mám zase zajímavé zkušenosti s návrhem responzivních webů rovnou v prohlížeči. Kombinace designu v HTML a CSS s grafickým editorem by ovšem vydala na samostatnou knížku. Začít ale můžete v tomto článku: vrdl.cz/blog/38-design-v-prohlizeci.

Pokud to pracovní procesy vašeho týmu alespoň trochu dovolují, zkuste zaexperimentovat s některou ze zmíněných cest.

Od ornamentů k typografii a dál

V souvislosti s nástupem mobilů se rozšířily graficky velmi úsporné styly typu „flat“ (plochý) design. Graficky velmi jednoduché jsou také designérské systémy velkých firem, například Material Design od Google. material.io/guidelines

Důvody jsou samozřejmě velmi praktické. V responzivní éře webdesignu není dopředu jisté, jak velký navrhovaný element bude a jaké budou vykreslovací schopnosti zobrazovacího zařízení. Grafické styly bohaté na ornamenty by v kombinaci s dnešními nedokonalými pracovními postupy prodražovaly implementaci.

Mnoho webařů si na tenhle trend stěžuje, připadá jim „omezující“ z pohledu grafického designu. Na jednu stranu je to pravda. Na druhou stranu ale na webu máme relativně nový, ve starších médiích naopak prastarý nástroj pro kreativní grafický design – typografii.

Typografie nabízí velkou škálu možností, jak návštěvníkovi stránky s informací předat i pocit či emoci. Počínaje volbou řezů písma až po jejich správné kombinování nebo CSS3 či SVG efekty na ně aplikované.

V tomto e-booku budeme typografii řešit hlavně z technického pohledu – CSS3 vlastnost `@font-face`. Vězte ale, že z grafického pohledu je typografie ohromně zajímavým nástrojem. Pokud se do oboru chcete alespoň trochu ponořit, začněte na webtypography.net.

U typografie ale staronové vyjadřovací schopnosti webdesignérů teprve začínají. Historickým pohledem je dnes na Webu nevídaně jednoduché animovat, provádět složitější interakce nebo spouštět video bez potřeby externích modulů jako byl Flash.

Nástroje, postupy, technologie

Na začátku této části musím ventilovat svůj pocit, že se vývoj webového rozhraní v poslední době na nástroje zaměřuje až příliš. Nastavením dokonalého workflow v Gruntu trávíme někdy zbytečně moc času v porovnání s tvorbou produktu zaměřenou na uživatele.

V brašně UI vývojáře by to chtělo více designérského uvažování, méně ladění nástrojů.

Ani v následující části to tedy nebudeme přehánět s hloubkou znalostí. Dřevorubec si vystačí i s postarší motorovou pilou, když ji má správně servisovanou a zvládá ji dokonale používat.

Uděláme si jen takový rychlý přehled toho, co se motorové pily frontendu za posledních pár let naučily.

1. CSS preprocesory
2. Invaze Node.js ekosystému
3. Správa balíčků: NPM a Bower
4. Buildování: Prepros, Grunt, Gulp
5. Postprocesing: Autoprefixer, pixrem a další
6. Udržitelný kód pomocí OOCSS
7. Bootstrap a hotové UI knihovny
8. Testování v alternativních prohlížečích: Browserstack a spol.
9. Efektivita převodu grafického návrhu: Avocode, CSSHat...
10. Živý náhled v prohlížeči: BrowserSync
11. Další nástroje a weby

CSS preprocesory

LESS, SASS a další preprocesory frontendistům zjednodušují práci s CSS. Jedná se o jazyky postavené nad CSS. Přidávají do něj nové vlastnosti nebo zjednodušují organizaci kódu. Už na vývojářově pracovní stanici jsou pak kompilovány do CSS, aby kódu rozuměly prohlížeče.

Vlastnosti preprocesorů

PROMĚNNÉ

Znáte je z imperativních programovacích jazyků a možná byste nevěřili, jak moc mohou být užitečné v CSS.

Aktivní barvu ve své implementaci frameworku Bootstrap změním jednoduše předeklarováním na vyšší úrovni. Zápis v LESSu:

```
@brand-primary: #428bca;
@import "bootstrap/bootstrap";
```

Konkurenční framework Foundation má zase v proměnné `$medium-up` uložen celý dotaz na média a nemusíte jej tedy psát pořád znova. Zápis v preprocesoru SASS:

```
$medium-up: "only screen and (min-width: 40em)";
@media #{$medium-up} {
  // Kód pro viewporty od šířky 40em
}
```

ZANOŘOVÁNÍ

```
 .nav {
  // …

  @media only screen and (min-width: 768px) {
    width: 25%;
  }
}
```

Zkompiluje do:

```
  @media only screen and (min-width: 768px) {
  .nav {
    width: 25%;
  }
}
```

Vypadá to jako zbytečnost, ale je to jeden z důvodů, proč jsem CSS preprocesory začal používat já. V CSS se ve většině prohlížečů Media Queries zanořovat nedají a to trochu svádí k organizaci kódu přes Media Queries. Výhodnější je vždy organizace přes komponenty. V příkladu výše tedy bude hlavní organizační jednotkou modul `.nav` a Media Queries budou zanořené v něm. To chceme.

MIXINY

I vaše CSSko je plné opakujícího se kódu. Proto se mixiny dnes ukazují už v každé kodérské školce. Jde o pojmenovaný kus kódu, který používáte na různých místech pouhým voláním názvu.

Klasické použití neparametrické mixiny pro ukončení obtékání (LESS):

```
  .clearfix() {
  &:before,
  &:after {
    content: " ";
    display: table;
  }
  &:after {
    clear: both;
  }
}

.el {
  .clearfix;
}
```

Zkompiluje do:

```
   .el:before,
.el:after {
  content: " ";
  display: table;
}
.el:after {
  clear: both;
}
```

Mixiny ovšem mohou mít také parametry, to je teprve legrace!

@IMPORT

Pokud importujete parciální komponentu napsanou v preprocesoru, nechová se `@import` tak, jak jste zvyklí z CSS, kde vytváří requesty navíc. A requesty bolí, protože zpomalují načítání stránky, hlavně na mobilních zařízeních.

```
@import "module.less";
```

Pokud importujeme běžný CSS soubor, preprocesory ve zkompilovaném kódu zachovají direktivu @import. Toto chování můžeme změnit nastavením vlastnosti LESS:

```
@import (less) "fancybox.css";
```

DALŠÍ VLASTNOSTI

Preprocesory mají mnoho a mnoho dalších vlastností. Podívejte se na ně:

- lesscss.org/features
- sass-lang.com/guide

Každá složitější vlastnost ale komplikuje srozumitelnost CSS kódu. Osvědčilo se mi trvat na jednodušším kódu a z preprocesorů si brát jen ty nejdůležitější vlastnosti. Pokud pracujete v týmu, je jednoduchost CSS obzvlášť důležitá.

JAKÝ PREPROCESOR ZVOLIT?

Když to hodně zjednoduším, mám dvě rady:

- Pokud s preprocesory začínáte nebo jste kodér a píšete hlavně CSS, zvolte LESS.
- Pokud jste programátor a píšete i JavaScript nebo třeba PHP, zvolte SASS.

Složitější text o výběru jsem dříve sepisoval na blog – vrdl.cz/blog/15-css-preprocesory-4.

Není ovšem potřeba výběr příliš prožívat. Přechod od jednoho preprocesoru k druhému většinou moc bolestivý není.

Nevýhody CSS preprocesorů

- Až moc silný nástroj – odklon od tuposti CSS a příliš mnoho abstrakce sice vede k propracovanému, někdy až imperativnímu kódu, zároveň ale často špatně čitelnému a spravovatelnému. Znáte to přísloví o dobrém sluhovi, ale špatném pánu, že ano?
- Proprietární kód – pokud preprocesory využíváte „tupě“, jsou zaučení nového člověka nebo přechod na jiný preprocesor relativně jednoduché; horší je to ovšem v kombinaci v prvním bodem.

Myslím, že minimálně některé z úkolů, jež nyní řeší preprocesory, časem přejdou do následného zpracování kódu. V dalším textu si proto nenechte ujít kapitolu o postprocesingu.

Invaze Node.js ekosystému

Nástroje kolem Node.js jsou určeny pro vývoj javascriptových aplikací na serveru. Jenže JavaScript tak trochu potřebujeme i v prohlížeči, že?

Člověk nemusí napsat stovky řádků javascriptového kódu, stačí, když občas instaluje jQuery pluginy.

Javascriptoví vývojáři, ale i CSS kodéři prostě potřebují nástroje pro instalaci a správu balíčků a pak něco pro sestavování distribučních verzí kódu.

Budeme se bavit o NPM, Boweru, Gruntu či Gulpu. Vždy je dobré si uvědomit, že to jsou nástroje vycházející právě z Node.js světa. nodejs.org

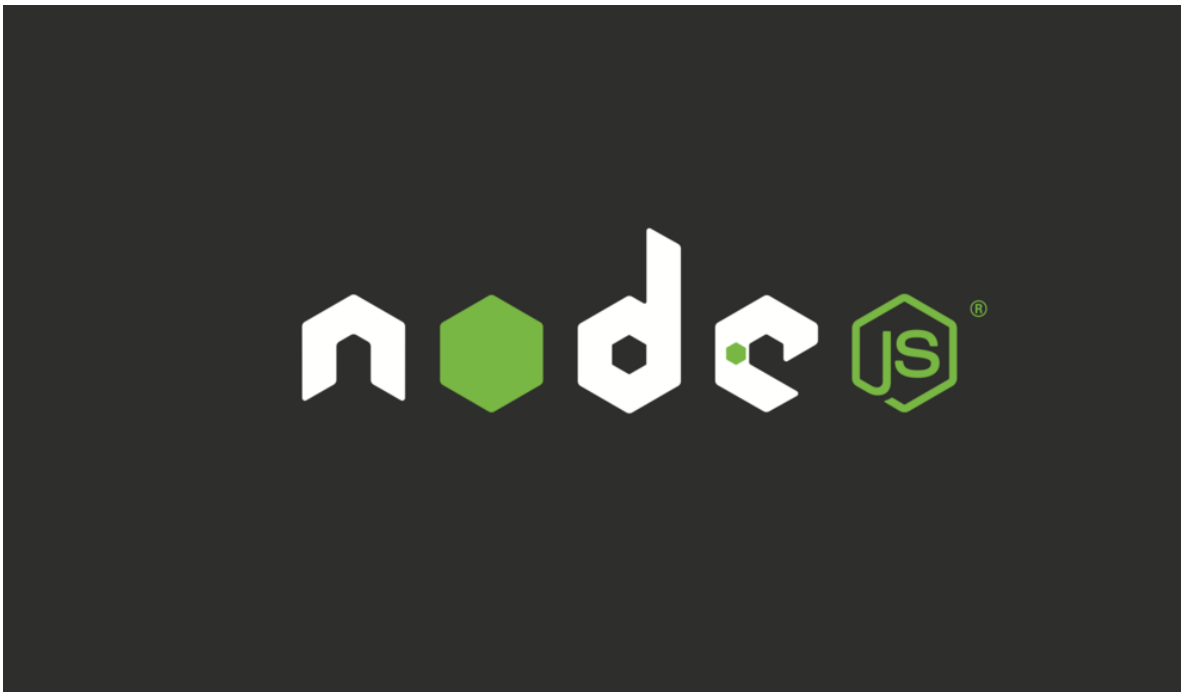
Instalace ekosystému Node.js pro použití na frontendu

Node.js na frontendu potřebujeme hlavně kvůli usnadnění vývojářské práce a správě vývojářských závislostí.

NÁSTROJE, KTERÉ SI NAINSTALUJEME

- Node.js – běh javascriptu na příkazové řádce.
- NPM (Node Package Manager) – správce javascriptových balíčků.
- Grunt – sestavovač nebo buildovač; nástroj pro běh vývojářských úloh.

Někdo si může oblíbit i další nástroje tohoto ekosystému, třeba Bower, správce frontend balíčků. Nebo například automatizátor Yeoman, Gulp nebo WebPack. Mohou být velmi užitečné, ale pro zjednodušení se jim tady nebudeme věnovat.



Budeme pracovat s příkazovou řádkou, takže se hodí znát její základy:

- Návod pro Windows: „Starting to use the Command Prompt“. vrdl.in/6w1a0
- Návod pro Linux & Mac: „How to Use Terminal: The Basics“. vrdl.in/u03m2

NODE.JS A NPM

NPM (Node Package Manager) je balíčkovací systém Node.js. Ten musíte stáhnout a nainstalovat jako první.

Jednoduchý postup instalace pro všechny platformy je popsaný na webu Node.js: nodejs.org/en/download.

Podívejme se ale i na lepší postupy.

Postup pro Windows

1. Nainstalujte Git do příkazové řádky. V kroku „Adjusting your PATH environment“ zkontrolujte zatržení „Run Git from the Windows Command Prompt“. git-scm.com/downloads
2. Nainstalujeme Node.js. nodejs.org/en/download

Postup pro Mac OS a Linux: NVM, nebo hraní s uživatelskými právy

- Ideální varianta – pomocí NVM (Node Version Manager). Nainstalovat jej není úplně přímočaré, ale má to dvě výhody. Na vývojářské mašině vám může souběžně běžet více verzí Node a NPM najednou. A pak – ušetříte si opruz s administrátorskými právy z méně optimálních variant. github.com/creationix/nvm
- Druhá možnost je buď hrát si s uživatelskými oprávněními ve výchozím NPM adresáři nebo ho umístit jinam. vrdl.in/4hlaw
- Třetí, úplně nejhorší varianta pak je spouštění instalace balíčků v administrátorském režimu (`sudo npm install …`) pokaždé, když vám NPM zahlásí problém s právy (`npm ERR! Attempt to unlock …`). Autor NPM o tomhle postupu údajně prohlásil, že je bezpečný jako nechat se stříhat motorovou pilou. Což sedí.

Máte nainstalováno? Jestli vše funguje, zjistíte příkazem pro zobrazení verze NPM:

```
npm -v
```

Všechny ostatní nástroje se pak instalují jako Node balíčky.

GRUNT, BOWER A DALŠÍ NODE BALÍČKY

Budete používat sestavovač Grunt? V příkazové řádce potřebujete globálně nainstalovat balíček `grunt-cli`:

```
npm install -g grunt-cli
```

A co frontend balíčkovač Bower? To už je jednoduché. Opět jej nainstalujeme globálně:

```
npm install -g bower
```

Úplně stejným způsobem si pak můžete nainstalovat Grunt pluginy, alternativní sestavovač Gulp nebo třeba Webpack.

GRUNT PLUGINY

```
npm install <nazev-pluginu> --save-dev
```

Všimněte si, že už neinstalujeme globálně – bez přepínače `-g`. Ano, Grunt pluginy nebo třeba Bower balíčky instalujeme ke konkrétnímu

projektu. Zároveň je chceme uložit do konfiguračních souborů balíčkovacího systému (`packages.json`).

Pro inspiraci: Moje tipy na Grunt pluginy. vrdl.cz/prirucka/grunt-pluginy

Jste na Windows? Pak pozor, některé Grunt pluginy vyžadují poněkud speciální péči. Například tyto dva:

- PhantomJS, na kterém závisí třeba plugin pro generování kritického CSS. vrdl.in/fa8jb
- ImageMagick, který zase potřebují pluginy pro práci s obrázky jako grunt-contrib-imagemin. vrdl.in/ti9jp

BOWER BALÍČKY

```
bower install <nazev-pluginu> --save
```

`--save-dev` přepínač uloží plugin do vývojářských závislostí. U začátečníků se používá hlavně v souvislosti s Gruntem a jeho pluginy. Ty instalujeme pomocí NPM.

Pokud bychom instalovali jQuery, půjde o uživatelskou závislost. Použijeme `--save` přepínač. Pro takové závislosti se více hodí Bower.

TAHÁK PRO PRÁCI S BALÍČKOVACÍMI SYSTÉMY

Bower i NPM mají naštěstí podobné příkazy:

```
# Vyhledávání balíčku v centrálním repozitáři:
npm/bower search jquery-ui

# Zobrazit detaily o balíčku:
npm view jquery-ui
bower info jquery-ui

# Instalace všech balíčků projektu:
npm/bower install
# Instalace balíčku a přidání do uživatelských závislostí:
npm/bower install jquery-ui --save
# Instalace balíčku a přidání do vývojářských závislostí:
npm/bower install jquery-ui --save-dev
```

```
# Instalace specifické verze balíčku:
npm/bower install jquery-ui@1.11.x

# Aktualizace všech balíčků:
npm/bower update

# Výpis stromu závislostí:
npm/bower list
# Výpis stromu závislostí a verze konkrétního balíčku:
npm/bower list jquery-ui

# Odstranění balíčku:
npm/bower uninstall jquery-ui

# Smazání cache. Hodí se v případě reinstalace sady balíčků:
npm cache clean

# Zobrazení balíčků, které je potřeba aktualizovat:
npm outdated
bower list

# Nápověda:
npm help
```

VYZKOUŠEJTE SI TO

Nainstalováno? Pokud nemáte nic lepšího po ruce, vezměte příklad ze školení Dnešní webová kodéřina a postupujte podle návodu. git.io/vMqaj

Správa balíčků: NPM a Bower

Dva balíčkovací systémy? Začátečníkům to dělá docela problémy. V následujícím textu se také dozvíte, k čemu se více hodí NPM a k čemu Bower.

V kontextu vývoje UI se přes ně instalují dva typy balíčků:

- komponenty webu – např. Bootstrap nebo jQuery a jeho pluginy (obvykle obstará Bower)
- software pro interní potřebu vývojáře – např. Grunt pluginy vylepšující naše workflow (obvykle obstará NPM)

Proč vlastně balíčkovací systém?

Kodéři tradičně žádný balíčkovací systém nepoužívali, pojďme si proto říct argumenty na jeho podporu.

Především je to **snadnější instalace** – napíšu `bower install jquery` a do adresáře `bower_components` se mi nainstaluje aktuální verze jQuery.

S tím souvisí i **jednodušší aktualizace** – když vyjde nová verze jQuery, napíšu jen `bower update jquery` a mám ji v projektu.

To celé pak **zlepšuje spravovatelnost projektu** – NPM i Bower komponenty se ukládají do zvláštních adresářů, které neverzujeme. Závislosti projektu máme verzované jen v konfiguračních souborech NPM nebo Boweru. Díky tomu máme jednodušší repozitář a jeho historii.

Node Packaging Manager, NPM.js

Balíčkovací systém pro jazyk Javascript. Tak jako PHP svět má svůj Composer, Javascript má svůj NPM.

V mém workflow slouží hlavně k instalaci knihoven, které využívám pro interní potřebu – typicky ony Grunt pluginy. Přístupů k balíčkování je ale více, a tak je dobré vědět, že knihovny využívané webem jako jQuery lze pomocí NPM nainstalovat také.

Používá konfigurák `package.json` a standardně vše instaluje do adresáře `node_modules`.

Více na npmjs.org.

Bower

Bower je balíčkovací systém pro frontend. Komponenty, které s jeho pomocí spravuji, jdou nad rámec Javascriptu. Obsahují často také CSS nebo obrázky.

Jak vyplývá z předchozího, Bower v mém případě spravuje závislosti projektu. Nejen tedy obligátní jQuery balíčky, ale také polyfilly typu Respond.js nebo Picturefill a další knihovny, jako je Bootstrap.

Používá konfigurák `bower.json` a standardně vše instaluje do adresáře `bower_components`.

Více na bower.io.

Pár užitečných příkazů

Bower naštěstí převzal syntaxi NPM, takže práce s ním je velmi podobná.

Vyhledání knihovny:

```
npm|bower search jquery
```

Instalace jQuery:

```
npm|bower install jquery
```

Můžete použít přepínač `--save-dev`, který vám knihovnu rovnou uloží do vývojářských závislostí v konfigurácích `bower.json` nebo `package.json`. V případě jQuery jde ale o uživatelskou závislost, proto byste při vývoji webu použili nejspíše `--save` přepínač.

Pojďme dál. Aktualizace jQuery:

```
npm|bower update jquery
```

Instalaci i aktualizace je možné dělat pro celý projekt najednou. Bower i NPM porovnají konfigurační soubor s aktuálně vydanými verzemi závislostí:

```
npm|bower install|update
```

Bower versus NPM

Pojďme si tedy zopakovat rozdíly v účelu:

- NPM je balíčkovací systém pro jazyk Javascript. Ve workflow kodéra slouží typicky pro software interní potřeby.
- Bower je balíčkovací systém pro frontend. Typicky se používá pro komponenty webu.

Kromě účelu se oba balíčkovače liší i ve způsobu ukládání závislostí:

- NPM instaluje balíčky, na kterých je instalovaný balíček závislý, pro každý balíček zvlášť. Ve složce, kam závislosti ukládá, tedy bývá více verzí jednoho balíčku. Například pro Grunt pluginy to dává naprostý smysl, každý balíček potřebuje trochu jinou verzi jiného balíčku.
- Bower naopak závislosti instaluje vždy jen jednou. Opět to v jeho kontextu dává smysl, jQuery chcete na webu mít jen v jedné verzi.

U průměrného projektu tedy bude složka `bower_components` datově vždy výrazně menší než složka `node_modules`.

Buildování: Prepros, Grunt, Gulp

Skoro vždy se nám na frontendu hodí provádění nějakých operací nad zdrojovými kódy. CSS chceme kompilovat z preprocesoru, pak ještě třeba minifikovat. Javascriptové soubory chceme zmenšit a spojit. Obrázky chceme datově optimalizovat a slepit do jedné „CSS sprite“.

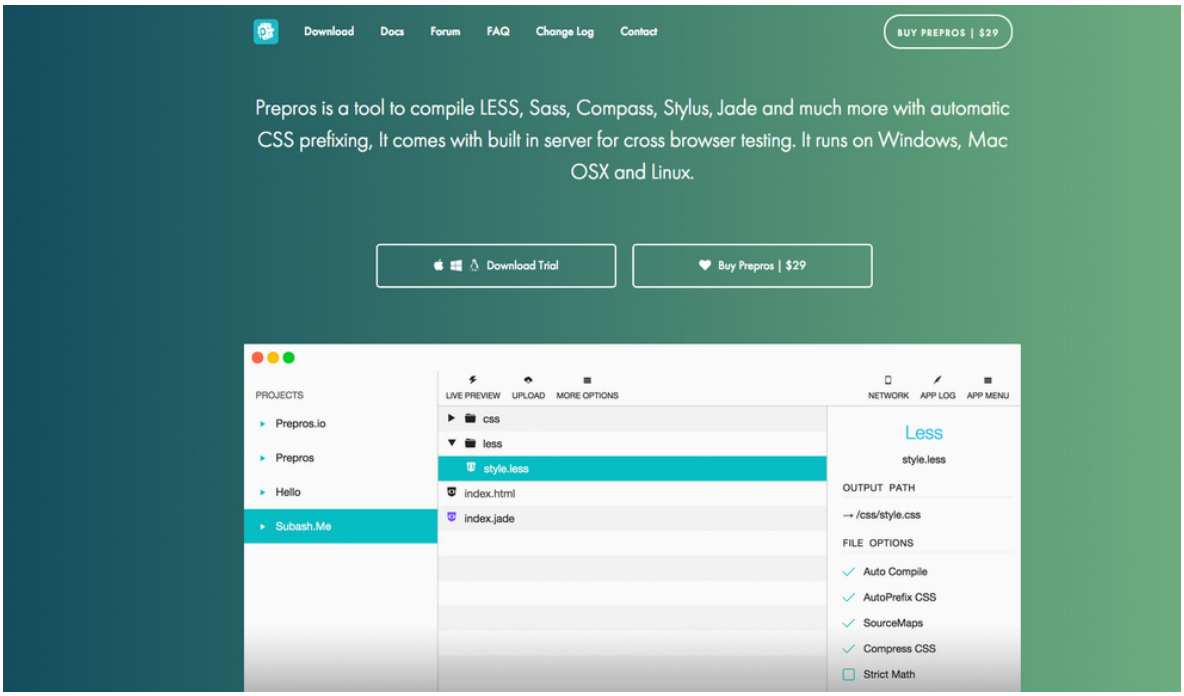
Zdrojové soubory tedy v HTML kódu nelinkujeme prohlížečům přímo. Nahradily je optimalizované distribuční verze. No a pro vytvoření distribučních verzí potřebujeme sestavovací (buildovací) nástroje.

Sestavovací nástroje pro webový frontend jsou dvojího typu:

- zjednodušené – snadněji se používají, ale mají omezený rozsah funkcí (Prepros, CodeKit a další)
- plnohodnotné – rozsahem funkcí téměř neomezené, ale začátečníci nebo neprogramátoři se do nich dostávají hůř (Grunt, Gulp a další)

Prepros

Zástupce zjednodušených buildovacích nástrojů. Obvykle doporučuji začít s ním, protože je multiplatformní a pro jeho ovládání je jen potřeba umět klikat.



Kromě všech základních úkolů nad CSS, Javascripty a obrázky umí aktualizovat weby přes FTP a provádět synchronizované prohlížení webu na více zařízeních.

Pokud se ale nechcete omezovat a jste ochotní investovat čas do učení složitějšího nástroje, začněte používat rovnou Grunt.

Grunt.js: plnohodnotný sestavovací nástroj

Grunt si říká „The JavaScript Task Runner“. Není to nic jiného než skript sloužící webovým vývojářům. Automatizuje opakující se nebo nudné úkoly.

Pokud si rádi šetříte práci, je Grunt naprosto návyková záležitost.

JAK GRUNT FUNGUJE?

V praxi to vypadá tak, že si něco spustíte na příkazové řádce, ono vám to hlídá změny v souborech a po jejich provedení vyvolá nějaké akce. Další úlohy, třeba nahrání webu na server, se zase spouštějí ručně.

Jednoduchým příkladem budiž minifikace CSS, JS souborů. Nebo jejich spojování do jednoho kvůli šetření requestů pro zvýšení rychlosti načítání. Nebo zpracování preprocesorového kódu do CSS. Grunt sám o sobě nic neumí, to až stovky existujících pluginů z něj dělají tu velkou věc.

Navíc jeho konfigurační soubor (`Gruntfile.js`) verzujete, takže nastavení úloh sdílíte se všemi členy týmu. Nemusíte se tak například bát, že jeden vývojář kompiluje LESS kód jedním způsobem a druhý jiným. A úlohu, kterou si napíše jeden lenivý vývojář pro šetření práce, okamžitě sdílí celý tým.

JAK SI GRUNT NAINSTALOVAT?

Je dobré hned na začátku zmínit, že pro používání Gruntu musíte alespoň trochu kamarádit s příkazovou řádkou. Žádné pokročilé vědomosti ale potřeba nejsou. Kolem příkazové řádky chodím po špičkách, s Gruntem jsem se ale spřátelil docela rychle.

Budete potřebovat nainstalovaný Node.js a Node Package Manager (NPM):

Globálně si pomocí NPM nainstalujete rozhraní Gruntu pro příkazovou řádku:

```
npm install -g grunt-cli
```

Pak si nainstalujete některý z pluginů, třeba pro kompilaci LESS do CSS:

```
npm install grunt-contrib-less --save-dev
```

Z pluginů pak vytváříte Grunt úlohy. Tady je ukázka jednoduchého `Gruntfile.js`:

```
module.exports = function(grunt) {
"use strict";
grunt.initConfig({
  pkg: grunt.file.readJSON('package.json'),

  // Kompilace z LESS do CSS:

  less: {
    default: {
      files: {
        'dist/css/style.css':
        'src/less/index.less'
      }
    }
  },

  // Vytvoření minifikovaného CSS pro produkci:

  cssmin: {
    css: {
      files: {
        'dist/css/style.min.css':
        'dist/css/style.css'
```

```
      }
    }
  },

  // Sledování změn v LESS souborech:

  watch: {
    less: {
      files: 'src/less/**/*.less',
      tasks: ['css']
    }
  }
});

// Alias úlohy:

grunt.registerTask('css', ['less', 'cssmin']);
grunt.registerTask('default', ['watch']);
};
```

Nejdříve nakonfigurujeme pluginy `less`, `cssmin` a `watch`. Pak z nich vytvoříme „alias úlohy“ `css` a `default`.

Když tedy začínáme práci na projektu, příkazem `grunt` pustíme `default` úlohu a začnou se nám sledovat změny v souborech.

Když už tedy máme `Gruntfile.js` s konfigurací, můžete Gruntu říct, aby vám zobrazil všechny dostupné příkazy:

```
grunt --help
```

To by pro začátek mohlo o Gruntu stačit. Tady jsou odkazy:

- Homepage: gruntjs.com.
- Seznam Grunt pluginů pro inspiraci: vrdl.cz/prirucka/grunt-pluginy.

GRUNT VERSUS GULP

Konkurenční nástroj – Gulp – umí v zásadě totéž co Grunt, jen je v základu rychlejší a konfigurace úloh provádí pomocí javascriptového kódu. Je proto snadnější v něm věci zobecňovat, a tak je vhodnější pro programátory nebo větší projekty. Grunt je zase přehlednější pro CSS kodéry a začátečníky.

Více na gulpjs.com.

Postprocesing: Autoprefixer, Pixrem a další

Postprocesing neboli následné zpracování CSS kódu je zajímavý trend v CSS světě.

Obvykle slouží k doplňování kódu tvořícího fallback pro starší prohlížeče, ale za pár let může zčásti nahradit CSS preprocesory. Nejlépe si to ukážeme na příkladech nástrojů pro následné zpracování.

Autoprefixer: doplňování prefixovaných deklarací

Kód píšete podle W3C syntaxe ...

```
transform: scale(1.1);
```

... a Autoprefixer pak do vašeho CSS souboru doplní prefixované varianty:

```
-webkit-transform: scale(1.1);
-ms-transform: scale(1.1);
transform: scale(1.1);
```

Standardně doplňuje prefixy pro poslední dvě verze prohlížečů. To ale můžete snadno změnit. Tady je potřeba konfigurace grunt-autoprefixer, plugin pro Grunt, který nastavuje širší podporu prefixovaných variant CSS vlastností:

```
autoprefixer: {
  options: {
    browsers: [
      'last 3 versions', 'ios 6',
      'ie 7', 'ie 8', 'ie 9'
    ]
  },
  style: {
      src: 'dist/css/style.css',
```

```
      dest: 'dist/css/style.css'
  }
}
```

Více na github.com/postcss/autoprefixer.

pixrem

Pokud se rozhodnete využívat jednotku `rem`, což vřele doporučuji, vznikne vám nejspíš problém se zpětnou kompatibilitou ve starších prohlížečích.

Pixrem pluginu sdělíte přepočet mezi `rem` a pixely a do CSS kódu takhle pěkně doplníte fallbacky:

```
.box {
  margin-bottom: 24px;
  margin-bottom: 1.5rem;
}
```

Více na github.com/robwierzbowski/grunt-pixrem.

legaccsy

Plugin Robina Pokorného, jenž umí vytvořit verzi CSS souboru bez Media Queries pro starší prohlížeče, které dotazům na média nerozumí. Záchrana pro IE8 nebo IE9.

Více na github.com/robinpokorny/grunt-legacssy.

postcss

Jak už jsem na více místech zmínil, postcss může časem v některých aspektech nahradit LESS, SASS a další preprocesory. Filozofie je jednoduchá – CSS kód píšete v syntaxi, kterou navrhuje W3C, ale prohlížeče ji zatím neumějí. Postcss vám kód zkompiluje do dnešního CSS.

Budete se možná divit, ale po letech mrtvolného spánku se dnes standard CSS vyvíjí i v rovině syntaxe jazyka.

V postcss tedy tenhle kód používající CSS proměnné a pojmenovaná média…

```
   :root {
  --fontSize: 1.2rem;
}

@custom-media --viewport-medium (width <= 600px);

@media only screen and (--viewport-medium) {
  body { font-size: calc(var(--fontSize) * 1.2); }
}
```

... zkompiluje do tohoto dobře známého kódu:

```
   @media only screen and (max-width: 600px) {
  body { font-size: 1.44rem; }
}
```

Postcss má krásný slogan:

> Use tomorrow's CSS syntax, today.

Více na cssnext.io.

Udržitelný kód pomocí OOCSS

OOCSS je zkratka pro „Object Oriented CSS”. Je to koncept organizace kódu od Nicole Sullivan.

CSS objekt je opakující se vizuální komponenta, která může být abstrahována kouskem HTML a CSS, případně JS. Je znovupoužitelná na jiném místě projektu. Ideálně pak i na dalších projektech.

Cílem OOCSS je zajistit znovupoužitelnost kódu, zlepšit jeho spravovatelnost a také zmenšit objem CSS souboru.

Ukažme si nejprve zjednodušený kód komponenty s tlačítkem:

```
   /* Komponenta */
.button { … }

/* Elementy komponenty */
.button-icon { … }

/* Modifikátory komponenty */
.button-primary { … }
.button-login { … }
```

V mém pojetí má OOCSS pět principů:

1) Nezávislost vzhledu na struktuře

Do CSS selektorů nikdy nedáváme HTML tagy. Mohou se změnit. Proto `.button` raději než `input.button`.

2) Nezávislost obsahu na kontejneru

Do CSS selektorů nikdy nepromítáme strukturu HTML. Může se změnit. Proto `.button.button-login` raději než `.login-form .button`.

3) Vývoj zaměřený na komponenty, znovupoužitelnost

Komponenty (neboli objekty) nezávislé na struktuře HTML lze snadno používat i na jiných projektech.

Komponenty pak tvoří uzavřený celek, který v preprocesorech importujeme.

```
@import "button";
```

Zpřehledňují nám nejen kód samotný, ale i commity do repozitáře.

4) Objekt, element, modifikátor

Tři typy prvků:

- objekt – jinak též komponenta nebo blok (`.button`)
- element – prvek uvnitř objektu, jinak též podobjekt (`.button-icon` pro ikonu uvnitř tlačítka)
- modifikátor – vlastnost objektu (`.button-primary` pro hlavní call-to-action tlačítko)

U větších projektů může být výhodné tyto tři typy prvků odlišit vizuálně. Podívejte se na metodiku BEM: vrdl.cz/prirucka/bem.

5) Co nejnižší specifičnost

V CSS nikdy nepoužíváme selektory identifikátorů (`#id`) a klauzuli `!important` si necháváme jen pro debugovací účely.

Kvůli zachování nízké specifičnosti se také snažíme co nejméně používat:

- selektorů potomka (v CSS nepíšu `.button .button-icon`, jen `.button-icon`)
- kombinovaných selektorů (v CSS nepíšu `.button.button-primary`, jen `.button-primary`)

Více o specifičnosti v CSS: specificity.keegan.st.

Bootstrap a hotové UI knihovny

Bootstrap si sice říká frontend framework, ve skutečnosti je to ale spíše knihovna komponent uživatelského rozhraní.

Bootstrap ohromně ulehčil práci programátorům, kteří jsou občas nuceni tvořit uživatelská rozhraní. A moc jim to nejde. V tomto scénáři použití stačí umět základy HTML, zatímco do CSS a Javascriptu nemusíte moc sahat.

Další velmi zajímavý scénář je využití Bootstrapu jako startovacího řešení, ze kterého vzniká váš vlastní kód. Takhle Bootstrap používám sám a nemůžu si jej vynachválit.

Co mi Bootstrap nabídne?

- Základní stylování dokumentu
- Plně responzivní mřížku pro tvorbu layoutu
- Pokročilé stylování základních komponent webu: formuláře, tlačítka, tabulky…
- UI komponenty typu záložkové navigace, akordeóny, modální okna…
- jQuery pluginy pro zajištění chování výše uvedených UI komponent
- Možnosti měnit vzhled a chování komponenty pomocí LESS proměnných

Není Bootstrap zbytečně velký „moloch“?

Ano, je. Když jej používáte špatně. :-)

Bootstrap se trošku nešťastně tváří jako jednolitý celek. Vnitřně je ovšem systematicky rozdělen do menších komponent.

Když si vyberete jen to, co na projektu potřebujete, komplexnost kódu a datová náročnost se sníží.

Kdy Bootstrap pomůže a kdy ne?

Bootstrap vám pomůže při tvorbě rozhraní webových aplikací typu administrace, při prototypování a když chcete využít některé z jeho

komponent. Při dodržení jeho principů se hodí i na prezentační weby. Moc se nehodí na graficky unikátní mikrosajty. Viz článek vrdl.cz/blog/11-bootstrap-pomuze.

Zmíněné typografické principy Bootstrapu jsem popsal zde: vrdl.cz/prirucka/bootstrap-typografie.

Správnou práci s Bootstrapem jako systémem komponent ukazujeme na Bootstrap školení: vrdl.cz/kurzy/bootstrap.

Browserstack: testování v alternativních prohlížečích

Co se týká renderování CSS kódu, není mezi dnešními moderními prohlížeči velkých rozdílů.

Naneštěstí je dnes prohlížečů hodně, nemluvě o nově příchozích mobilních. A pak jsou tu prohlížeče v důchodu jako Internet Explorer 8.

Kodér se tedy bez testování v alternativních prohlížečích neobejde.

Je tu možnost stáhnout si alternativní operační systémy přes VirtualBox nebo podobné programy, a pak mobilní prohlížeče testovat v emulátorech a simulátorech.

Druhá možnost je Browserstack nebo podobné online služby. Ty alternativní prohlížeče provozují u sebe a vám je poskytují pro vzdálené připojení.

Browserstack stojí nezanedbatelné peníze. Jsem si ovšem jistý, že ve srovnání s časem stráveným nad údržbou Android emulátorů a VirtualBoxu se investice bohatě vyplatí.

- browserstack.com
- Alternativa: crossbrowsertesting.com

Jak testovat responzivní weby?

Ideální je třífázové testování:

1. Vývojářský desktopový prohlížeč – v mém případě Chrome, jeho DevTools a emulace mobilních rozlišení.
2. Simulátory/emulátory – u mě vyhrál Browserstack.
3. Reálná zařízení – u mě iPhone 4 s iOS, Vodafone 945 se starým Androidem 2.1, iPad Mini s iOS8, Tablet Sencor Element 7 s Androidem 4.1, Samsung Galaxy S III mini s Androidem 4.1 a Nokia Lumia 520 s Windows Phone 8.

Více najdete na blogu: vrdl.cz/prirucka/jak-testovat-responzivni-weby.

Efektivita převodu grafického návrhu: Avocode, Brackets, CSSHat…

Díky několika nástrojům se převod grafického návrhu do CSS a HTML kódu v poslední době velmi zjednodušil.

Jednak vznikly designérské nástroje, které kód přímo produkují, takové Dreamweavery responzivního věku. Příkladem budiž WebFlow (webflow.com) nebo Macaw (macaw.co).

Opačným směrem pak jdou nástroje, které do tradičních grafických editorů přidávají možnost částečného exportu do CSS. Typickým příkladem je CSS Hat (csshat.com). Dnešní dominující designérské nástroje jako je Adobe Photoshop nebo Sketch od Bohemian Coding už ale nějaký ten export do CSS zvládají samy.

A pak je tu zcela nový typ nástrojů, které staví mosty mezi světem grafiků a světem kodérů. Jmenujme zejména český Avocode (avocode.com) nebo Brackets od Adobe (brackets.io).

Browsersync: živý náhled webu a ladění na zařízeních

Browsersync je velmi užitečný nástroj pro lokální vývoj webů. Pomáhá hned s několika činnostmi naráz:

1. Živé promítání změn ve zdrojácích do prohlížeče.
2. Synchronizace interakcí při testování webu.
3. Ladění webu na mobilních zařízeních.

Browsersync je Node.js komponenta, takže je kompatibilní s Gruntem, Gulpem, ale i dalšími nástroji tohoto ekosystému. Je opensource a zdarma: browsersync.io.

V textu budu jeho instalaci a základní vlastnosti ukazovat na příkladu. Zkušenější mohou skočit rovnou na poslední část „Další tipy pro práci s Browsersync“.

Instalace ukázky krok za krokem

Vezmeme tento příklad z ukázek využití Browsersync: git.io/vKfhs.

1. Na lokální mašině předtím potřebujete rozchodit Node ekosystém – hlavně NPM a Grunt. Volitelně také Git v příkazové řádce.
2. Naklonujte repozitář (nebo prostě stáhněte v ZIPu: git.io/vKfhc):
   ```
   git clone https://github.com/Browsersync/recipes.git bs-recipes
   ```
3. Skočte do adresáře s první ukázkou:
   ```
   cd bs-recipes/recipes/grunt.html.injection
   ```
4. Nainstalujte NPM závislosti:
   ```
   npm install
   ```
5. Pusťte příklad:
   ```
   npm start
   ```
6. Ve výchozím prohlížeči se vám otevře okno s adresou podobného tvaru:

```
http://localhost:3000/
```

To bychom měli. A teď ještě k čemu nám to bude, že ano?

Živé promítání změn do prohlížeče

Upravíte CSS nebo HTML soubor a změny se vám hned projeví v prohlížeči bez obnovení stránky. Možná už to znáte z jiných nástrojů, jako je LiveReload.

Pokud živý náhled neznáte nebo nevěříte, že to nějak zásadně pomáhá, opravdu (ale *opravdu*) si to zkuste.

V našem příkladu stačí upravit soubor `app/index.html` nebo `app/css/main.css`. Změny se hned projeví v prohlížeči. Obnovení stránky netřeba. Úplně nejlepší je nastavit si editor, aby ukládal změny v otevřených souborech hned po přepnutí do jiné aplikace. Pak stačí přepínat mezi editorem a prohlížečem. Šetří to hrozně energie, fakt že jo.

Synchronizace interakcí při testování webu

Browsersync vám během spuštění do příkazové řádky vypíše něco takového:

```
   [BS] Access URLs:
 ------------------------------------
      Local: http://localhost:3000
  External: http://192.168.0.2:3000
 ------------------------------------
         UI: http://localhost:3001
UI External: http://192.168.0.2:3001
 ------------------------------------
```

Co je to za adresy?

- `Local` – tam najdete svůj web.
- `External` – kde svůj web uvidíte na všech zařízeních připojených do stejné sítě.
- `UI` – rozhraní s nastavením Browsersync.
- `UI External` – rozhraní s nastavením na připojených zařízeních.

Vezměte mobil připojený do stejné wifi a vyťukejte do tamního prohlížeče `External` adresu. Teď když budete provádět uživatelské interakce v jednom zařízení, druhé bude dělat totéž za vás. Pěkné, ne? Browsersync to umí s klikáním, rolováním stránky nebo taky vyplňováním formulářů.

Proč vám o takové *blbině* vyprávím? Protože šetří děsně energie při testování responzivních webů na reálných zařízeních. vrdl.cz/prirucka/jak-testovat-responzivni-weby

Video: Browsersync: živý náhled webu a synchronizace prohlížení ~ Obě vlastnosti rozebrány ve videu. Podívejte se.

Ladění webu na mobilních zařízeních

Díky Browsersync také dostanete k dispozici jednoduchý nástroj podobný DevTools vašeho prohlížeče. Prostě *bazmek*, co umožňuje ladění HTML, CSS a JS kódu. Browsersync pro to využívá technologii Weinre.

Zkoušíte příklad a máte připojený mobil?

1. Podívejte se v počítači na `UI` adresu (obvykle `http://localhost:3001`). Otevře se vám prostředí pro nastavování Browsersync.
2. V levé navigaci zvolte „Remote Debug“.
3. Zapněte „Remote Debugger (weinre)“.
4. Klikněte na „Access remote debugger (opens in a new tab)“. V prohlížeči se otevře něco jako vývojářské nástroje. Tohle je Weinre.
5. Mezi „Targets“ zvolte ten první. Pravděpodobně to bude mobil, který jste před chvíli připojili.
6. Teď už stačí kliknout třeba na „Elements“ nebo „Console“, protože jste v prostředí podobném DevTools vašeho prohlížeče.

Weinre (vyslovujte jako *„winery“*) není tak pokročilá aplikace jako v prohlížečích vestavěné vývojářské nástroje. Máte ovšem k dispozici DOM, CSS a JS konzoli. To je pro základní ladění dost dobré. Ohromná výhoda Weinre je v tom, že můžete ladit napříč platformami. Třeba se z desktopového Firefoxu připojit do mobilního Safari. Homepage Winre: vrdl.in/abmfz.

Video: Browsersync: ladění mobilních prohlížečů ~ Vzdálené ladění pomocí Weinre a dalších funkcí Browsersync.

Browsersync a Grunt

Stačí nainstalovat a přidat konfiguraci podobnou této:

```
 browserSync: {
  dev: {
    bsFiles: {
      src : 'assets/css/*.css'
    },
    options: {
      watchTask: true,
      proxy: 'vzhurudolu.localhost'
    }
  }
},
```

Co jsem tím nastavil?

- V `bsFiles` je cesta k souborům, které se budou naživo vkládat do prohlížeče, jakmile je změníte.
- `watchTask: true` v nastavení úlohy říká, že soubory sledujete ještě `watch` pluginem. Pravděpdobně totiž po změně souboru provádíte ještě další operace nad nimi – minifikaci, spojování atd. BrowserSync tomuto procesu nesmí stát v cestě.
- V `proxy: 'vzhurudolu.localhost'` je adresa, na které mi projekt už na lokále běží. Využívám tedy jiný server (v mém případě Apache z MAMP balíčku). Je ale dobré vědět, že Browsersync nabízí vlastní server. Více v další části.

Další tipy pro práci s Browsersync

VLASTNÍ SERVER

V nastavení Grunt nebo Gulp tasku stačí uvést parametr a cestu k souborům. Browsersync vám spustí jednoduchý lokální server:

```
   server: {
  baseDir: "./"
}
```

Viz také browsersync.io/docs/grunt#grunt-server.

ŽIVÝ NÁHLED HTML

Pokud chcete vkládat změny v HTML souboru do všech připojených zařízení, použijte plugin HTML Injector. V demíčku ukazovaném v tomto textu je to už nastavené. Viz také github.com/shakyShane/html-injector.

PŘÍKLADY POUŽITÍ

Recipes jsou sada funkčních příkladů s předpřipravenými obvyklými nastaveními Browsersync. Pěkný zdroj inspirace i pro pokročilejší uživatele. Viz také github.com/Browsersync/recipes.

PŘIŠKRCENÍ RYCHLOSTI PŘIPOJENÍ

Pokud nepoužíváte Chrome, kde je možnost zpomalení rychlosti připojení vestavěná, bude se vám tahle vlastnost hodit. V demíčku jděte na `http://localhost:3001/network-throttle`. Výborné pro testování responzivních webů na mobilních internetových připojeních.

RYCHLÉ LADĚNÍ CSS LAYOUTU

Zobrazení obrysů prvků kvůli testování CSS layoutu můžete nastavit na `http://localhost:3001/remote-debug`. Layout je také možné testovat oproti mřížce vykreslené na pozadí. Používá technologii Pesticide. Pesticide.io.

To by mohlo být všechno. Browsersync vám tedy pomůže zefektivnit práci s frontend technologiemi a testování na mobilních zařízeních. Patří k mým nejoblíbenějším nástrojům. Zkuste ho.

Další nástroje a weby

Teď už to vezmeme hodně stručně.

HTML5 Please

Pro každou CSS3 (nebo HTML5) vlastnost vám tento web řekne, zda a za jakých podmínek ji doporučuje používat. Takže – zda vlastnost použít a zda použít s prefixy, nebo s polyfillem. U některých samozřejmě raději počkat. Berte to jako obecná doporučení a vždy se dívejte na cílovou skupinu svého konkrétního projektu. html5please.com

Can I Use

Jsou to tabulky s informací o tom, které prohlížeče vaši vlastnost podporují a které naopak ne. Pokud si chcete zapamatovat jen jeden web, tohle je on. Can I Use je možné si propojit s vašimi statistikami z Google Analytics. Podíly podpory vlastností na globální úrovni samozřejmě nemají pro konkrétní projekt vypovídající hodnotu. caniuse.com

Should I Prefix

Musím u konkrétní CSS3 vlastnosti použít prefixované verze, nebo nemusím? Může se hodit, ale obecně je lepším řešením Autoprefixer zmíněný v předchozích částech webu. shouldiprefix.com

Modernizr

Toto je javascriptová knihovna pro detekci vlastností. Vyberete si CSS3 nebo HTML5 vlastnosti, které na svém projektu využíváte, a web Modernizru vám vytvoří kus kódu, který pak vložíte do vlastního zdrojového kódu. V CSS pak můžete využívat podmínky jako `.no-flexbox .form`, v Javascriptu třeba něco jako `if (Modernizr.canvas) { … }`. modernizr.com

Šablony projektů

Pokud vás nebaví začínat projekty úplně od nuly, podívejte se na HTML5 Boilerplate, šablonu výchozího HTML souboru. Osvědčilo se mi brát ji jako zdroj znalostí, ale nepoužívat ji na reálných projektech. html5boilerplate.com

Masivnější základna projektu je pak Web Starter Kit od Google. Kromě výchozích HTML, CSS a JS souborů je tam rovnou hotové Gulp workflow. developers.google.com/web/starter-kit

Spoustu (i docela složitých) šablon pro start projektu obsahují generátory v rámci nástroje Yeoman. yeoman.io/generators

Dokumentace

Vynikající dokumentaci nových webových technologií najdete na Mozilla Developer Network. Třeba i pro CSS3. developer.mozilla.org.

Část obsahu tohoto ebooku najdete online na Vzhůru dolů. vrdl.cz/prirucka/css3.

Pokročilejší nástroje

Jak už jsem psal v úvodu, s nástroji se to nemá přehánět. V e-booku se soustředím na méně až středně pokročilého vývojáře uživatelského rozhraní. Pokud se počítáte k pokročilým, neměly by vám utéct nástroje Broccoli, Browserify, Webpack či CSSNext. Případně Babel a Traceur, pokud píšete hodně v Javascriptu.

Fallback strategie pro starší prohlížeče

Jednou z krásných vlastností HTML a CSS je *křápovzdornost*. Můžete používat úžasné nové CSS3 nebo i HTML5 vlastnosti, a přitom nerozbít zásadní vlastnosti stránky ve starých prohlížečích.

Když se budete alespoň trochu snažit.

V každém prohlížeči by měl být dostupný hlavní obsah nebo funkčnost

Mírné rozbití minoritních částí stránky ve starších prohlížečích asi nevadí. Pokud ale nemáte zvláštní důvody, je skoro vždy možné zajistit plnění hlavní funkce i v hodně starých prohlížečích.

Nejde jen o přístupnost a slušnost vůči takto znevýhodněným uživatelům. Jde také o současné mobilní prohlížeče – vezměme třeba širokou škálu verzí Android Browseru (a jeho úprav dodavateli zařízení) nebo prohlížeče jako Opera Mini. Nebo prohlížeče ve čtečkách e-booků. Není v lidských silách otestovat všechny. Blbuvzdorné řešení to dělá za vás.

A pak – představte si prohlížeče, které teprve přijdou. Prohlížeče v chytrých hodinkách, autech, televizích nebo – *ehm* – ledničkách. Že je lidé nebudou používat? Je to možné, ale o prohlížeči v iPhonu jsme si to mysleli taky.

Připravovat technické řešení webu *křápovzdorně* – s ohledem na nejnižší společný jmenovatel – se vyplatí i vzhledem k budoucnosti.

Progressive enhancement, postupné vylepšování

> Progressive enhancement – začít se základní funkcionalitou a poté krok po kroku zlepšovat uživatelský prožitek s tím, že před aplikací daného vylepšení nejprve otestujeme jeho podporu.
>
> – Honza Sládek, „Graceful degradation vs. progressive enhancement“, Zdroják.cz. vrdl.in/ybtvm

Cílem postupného vylepšování je garance poskytnutí obsahu nebo hlavní funkce stránky bez ohledu na technologické vybavení uživatele.

Progressive enhancement je postup, který vám *křápovzdornost* zajistí. Pojďme se podívat na různé strategie pro vytvoření řešení ve starých prohlížečích.

Křápovzdorné technické řešení v CSS3

Máme pět možných strategií: nulový nebo definovaný fallback, detekci vlastností nebo polyfilly, a pak speciální kategorii generovaných fallbacků.

1. Nulový fallback

Počet znaků fallback kódu: nula. Žádný alternativní kód prostě nepíšete. (Ovšem za podmínky, že přesně víte, co děláte.)

Nulový fallback využívá chytré vlastnosti HTML a CSS — ignorace neznámého. Prohlížeče odjakživa oba jazyky zpracovávají tak, že v případě, že narazí na neznámou značku, atribut, vlastnost nebo hodnotu, ignorují ji a pokračují ve zpracování kódu.

```
   .box {
  color: red;
  transition: .5ms;
}

.box:hover {
  color: darkred;
}
```

Tady se netrápíme tím, že starší prohlížeče nezvládají animace pomocí transition. Ty nám obvykle slouží k vylepšení uživatelského prožitku v moderních prohlížečích.

Pokud by ale animace nesla informaci (třeba jako indikace během načítání souboru), museli bychom informaci nějak předat i uživatelům starých prohlížečů.

Nulový fallback tak představuje možné řešení právě pro animace, kulaté rohy, stínování, vlastní fonty a mnoho dalších CSS3 vlastností.

2. Definovaný fallback

Fallback, který využívá toho, že prohlížeč aplikuje vždy poslední známou deklaraci. Vzpomeňte si na postprocesor grunt-pixrem

z dřívějšího textu:

```
 .element {
  /* Staré křápy: */
  font-size: 16px;
  /* Moderní prohlížeče: */
  font-size: 1rem;
}
```

Jde o možné řešení pro všechny nové CSS3 jednotky či pro RGBa. Nebo pro flexbox layouty.

Určitě si vzpomenete, že definovaného fallbacku využíváme také u prefixovaných variant CSS vlastností:

```
 .element {
  /* Chrome, Safari, Android:  */
  -webkit-column-count: 2;
  /* Firefox: */
  -moz-column-count: 2;
  /* IE 10, Op 11.1+: */
  column-count: 2;
}
```

3. Detekce podpory vlastnosti

U některých vlastností bohužel není možné přirozený CSS fallback použít. Řešení pro starší prohlížeče může obnášet odlišné řešení uživatelského rozhraní nebo jinak strukturovaný kód.

Tady přichází ke slovu detekce vlastností. Na tomto místě bych se rád vyhranil vůči staršímu postupu – detekci prohlížečů. Pomocí CSS hacků nebo zjišťování User Agent podpisu prohlížeče jsme detekovali prohlížeč, jenž vlastnost neumí. To bylo fajn ve světě, kdy tyto prohlížeče tvořila skupina dvou nebo tří starých verzí Internet Exploreru.

U každé CSS3 vlastnosti je ale skupina prohlížečů, které vlastnost neumí, různá. Někdy jen IE8 a starší. Někdy IE9 a starší a k nim třeba Opera Mini. Někdy se ještě přidají starší Android prohlížeče. Čert ví, co přijde za rok. Proto je daleko efektivnější cestou detekovat podporu

vlastností, nikoliv prohlížečů. Prohlížeče pak máme „v ceně“ a můžeme je pustit z hlavy.

Detekce vlastností se velmi hodí třeba u vektorového formátu SVG, o kterém už byla řeč:

```
.icon {
  background-image: url(icon.svg);
}

.no-svg .icon {
  background-image: url(icon.png);
}
```

V případě CSS3 vlastností se pak hodí zejména u těch, které se týkají layoutu. Příkladem budiž flexbox:

```
.component {
  display: flex;
}

.no-flexbox .component {
  display: table;
}
```

Nebo Multicolumn layout:

```
.text {
  columns: 2;
}

.no-csscolumns .text {
  float: left;
  width: 50%;
}
```

„Detekční“ třídy jako `.no-svg` nám do DOMu přidá javascriptová knihovna Modernizr, o které je řeč v kapitole o nástrojích. Nebo si můžeme napsat vlastní kousek detekčního javascriptu.

DOTAZY NA VLASTNOSTI

Modernizr aktuálně dostává *něco jako* nativní podporu od W3C. Přichází v podobě dotazů na vlastnosti – CSS Feature Queries. Pomocí zavináčového pravidla `@supports` se ptáte na dostupnost konkrétní CSS vlastnosti:

```
   .component {
  display: table;
}

@supports (display: flex) {
  display: flex;
}
```

Více o pravidle `@supports` čtěte na Mozilla Developer Network. vrdl.in/o9lm2

PRÁZDNÝ DOTAZ NA MÉDIA

Zajímavé použití detekce vlastnosti (a zároveň ukázkovou Progressive Enhancement!) představuje prázdná media query. Starší prohlížeče – jako IE do verze 8 nebo nějaké obskurnější mobilní křápy – prostě tuto část kódu nepřečtou. Hodí se třeba pro odstínění těchto prohlížečů od layoutu:

```
   .component {
  /* Základní pravidla pro
  typografii a lineární design */
}

@media only screen {
  .component {
    /* Layout a další
    netriviální pravidla */
  }
}
```

Vše podle těchto dvou hesel:

- Ve starých prohlížečích je podstatná dostupnost hlavního obsahu, nikoliv přesné dodržení grafického návrhu.
- Lineární zobrazení je lepší než rozpadlý layout stránky.

4. Polyfilly

Jde o javascriptové knihovny, které simulují podporu nových vlastností i v prohlížečích, které je neumí. Jsou velmi populární ve světě javascriptových vývojářů, pomohou ale i HTML/CSS kodérům. Příkladem budiž Respond.js, který zapne podporu CSS3 Media Queries i ve starších Explorerech, nebo Picturefill, který zase rozchodí responzivní obrázky – `<img srcset sizes>` a `<picture>`.

Kromě ověřeného Respond.js ale polyfilly u CSS3 vlastností nedoporučuji nasazovat. Obvykle zhoršují výkonnost stránky a činí vzhled závislým na Javascriptu.

5. Generovaný fallback

Zmiňoval jsem CSSnext a postprocessing, musíme to proto ještě jednou udělat v kapitole o vytváření fallbacků.

Alternativou k Respond.js, která zajistí fungování stránky i v prohlížečích bez podpory Media Queries, je zmíněný grunt-legacssy, který vygeneruje verzi CSS bez dotazů na média. github.com/robinpokorny/grunt-legacssy

Generovaný fallback ovšem většinou jen automatizuje už zmíněný definovaný fallback jako v případě grunt-pixrem. github.com/robwierzbowski/grunt-pixrem

Kapitola II:
Referenční příručka CSS3

A jsme u jádra pudla. Referenční příručka o CSS3 vlastnostech je rozdělena do kategorií podle toho co ovlivňují. Nejprve se podíváme na vlastnosti textu, pak vlastnosti pozadí a přes transformace, animace to vezmeme k layoutu.

Téměř každá vlastnost je obohacená živým příkladem na Codepen.io. Pro plný zážitek se tedy ve chvíli učení nové technologie hodí být online.

Seznam CSS3 vlastností

Vlastnosti textu

- Text Overflow
  Způsob přetékání textu. Hlavně pro vytečkování textu, který přesahuje šířku elementu.
- Text Shadow
  Stínování textu.
- Font Face
  Webové fonty, vlastní písma do stránky.

Vlastnosti pozadí

- Background Clip
  Míra roztažení pozadí. Určuje, kde všude se uvnitř elementu vykresluje obrázek nebo barva na pozadí.
- Background Origin
  Pozice začátku pozadí. Určí, kde se v rámci elementu nachází začátek osy pro počítání rozměrů a pozic vlastností jako Background Size nebo background-position.
- Background Size
  Velikost obrázku na pozadí.
- Gradients
  Barevné přechody vykreslené podle úsečky i kružnice.
- Multiple Backgrounds
  Více obrázků na pozadí jednoho elementu.

Vlastnosti rámečků

- Border Image
  Rámeček vykreslený obrázkem.
- Border Radius
  Vykreslování zakulacených a elipsovitých rohů elementu.

Vlastnosti boxu

- Box Shadow
  Stín nejen pod elementem, ale i uvnitř elementu nebo plastický efekt přes element.

- Box Sizing
  Změna způsobu počítání šířky a výšky elementu, jinak též řečeno box-modelu.

Media Queries

- Media Queries
  Podmíněné zobrazení pro média.

Transformace

- Transforms
  Proměna tvaru, pozice nebo velikosti elementu.

Animace

- Transitions
  Jednoduché animace přechodu.
- Animations
  Plnohodnotné animace v CSS.

Layout

- Multicolumn layout
  Vícesloupcová sazba (nejen) textu.
- Flexbox
  Layout pomocí pružných boxů.

Další

- Jednotky
  Jednotky jako root `em` (`rem`) nebo závislé na velikosti viewportu (`vw`, `vh`).
- calc()
  Funkce pro počítání matematických výrazů, včetně kombinací jednotek.
- RGBa
  RGB barva se čtvrtým číslem, alfa kanálem, jež podává informaci o hodnotě průhlednosti.
- Selektory
  CSS3 přichází s armádou nových selektorů.
- Box Reflection
  Odlesk objektu. Vlastnost nezařazená ve W3.org standardech.

- Nezařazené
  Ostatní a zatím nezařazené CSS3 vlastnosti.

Vlastnosti textu

Text Overflow: způsob přetékání textu

Vytečkování textu, který přesahuje šířku elementu.

```
text-overflow:
( clip | ellipsis | <_retezec_> );
```

Když na školeních ukazuji `text-overflow: ellipsis`, opakuje se vždy stejný scénář. Polovina účastníků zívá: „Hm, tohle používám už dva roky…“ A druhá polovina? Nadšeně si píše: „Musím použít hned zítra!“

Hodnota `ellipsis` má háček — aktuálně je možné ji použít jen na vytečkování jednořádkového textu na blokových elementech. I tak je to ale velmi užitečný pomocník.

Příklad s navigací

Představte si navigační lištu, kdy v každém případě potřebujete, aby text neskočil na další řádek. A jeho délku, stejně jako šířku boxu, ve kterém se objevuje, neznáte.

Pak stačí `ellipsis` doplnit o dvě další deklarace zajišťující „jednořádkovost“:

```
.element {
  text-overflow: ellipsis;
  overflow: hidden;
  white-space: nowrap;
}
```

Zkuste si naživo změnit velikost okna v příkladu na cdpn.io/e/FeLkJ.

Podpora v prohlížečích

`text-overflow: ellipsis` má podporu od IE6, takže není co řešit.

Text Shadow: stín textu

Stínování textu.

```
text-shadow:
 _posun_x_
 _posun_y_
 (_rozostreni_)
 _barva_,
 (_dalsi_stiny_);
```

Podívejte se rovnou na příklad: cdpn.io/e/aDLCl.

`text-shadow` má dvojče — stínování boxu box-shadow. Na stránce o stínování boxu najdete detailnější popis syntaxe. Je velmi podobná.

Tip – stíny textu je možné řetězit a vytvořit až pseudo-3D efekty: markdotto.com/playground/3d-text

Podpora v prohlížečích

IE10+. Ve starších prohlížečích si můžete vybrat buď tvrdý fallback bez stínu (preferovaná varianta), nebo s pomocí podmínek Modernizru či IE podmíněných komentářů udělat IE8– variantu s pomocí microsoftího filtru DropShadow. Více hledejte v dokumentaci na webu Microsoftu. vrdl.in/0ar6x

Font Face: vlastní fonty

Webové fonty neboli vlastní fonty do stránky? `@font-face` je dnes už standardní technika s takřka plnou podporou v prohlížečích, které se z technického pohledu není potřeba bát.

Syntaxe

Nejdřív pomocí at-pravidla `@font-face` nadeklarujete název rodiny a cestu k souboru:

```
@font-face {
  font-family: _nazev_rodiny_;
  src: url(_cesta_k_souboru_s_pismem_)
    format(_format_souboru_);
}
```

Pak název rodiny jednoduše zavoláte v běžném CSS:

```
.element {
  font-family: _nazev_rodiny_;
}
```

Formáty souborů s webovými fonty

Pokud nepoužíváte cloudová řešení typu Typekit nebo Google Fonts a uživatelům servírujete vlastní soubory s fonty, potřebuje základní povědomí o formátech souborů:

- **WOFF** (Web Open Font Format) – dnes převládající formát souborů. Podporovaný je ale MSIE až od verze 9 a Android Browserem od verze 4.4 – caniuse.com/woff
- **TTF/OTF** (TrueType/OpenType) – dva formáty, které podporují téměř všechny moderní prohlížeče, jen MSIE až od verze 9. Autor souboru musí navíc nastavit tzv. „embeding bits“ na „installable“. caniuse.com/ttf
- **SVG** (fonty definované ve vektorovém formátu SVG) – potřebujete, jen pokud chcete podporovat opravdu hodně staré verze iOS Safari — 4.3 a starší. caniuse.com/svg-fonts
- **EOT** (Embedded OpenType font) – podporují všechny Explorery od verze 4. Potřebujete, pokud chcete podporovat IE8 a starší. caniuse.com/eot

Pokud potřebujete podporovat všechny systémy, zápis je trošku složitější:

```
@font-face {
 font-family: 'MyWebFont';
 /* IE9 v kompatibilním režimu: */
 src: url('webfont.eot');
 src:
   /* IE6-IE8: */
   url('webfont.eot?#iefix')
     format('embedded-opentype'),
   /* Všechny moderní prohlížeče: */
   url('webfont.woff')
     format('woff'),
   /* Starší Safari, Android, iOS: */
   url('webfont.ttf')
     format('truetype'),
   /* iOS 4.3 a starší */
   url('webfont.svg')
     format('svg');
}
```

Dnes ale typicky potřebujete jen soubory ve formátu WOFF, TTF (kvůli starším Androidům) a EOT (kvůli IE8–). Ale čekají nás světlé zítřky. S formátem WOFF.

DO BUDOUCNA JEN WOFF

Za pár měsíců až let nám bude stačit jen WOFF formát:

```
@font-face {
  font-family: 'WebFont';
  src: url('webfont.woff');
}
```

Opět ale pozor. Vždy tu budou prohlížeče, které žádný z formátů webových fontů neumí. Například Opera Mini. Nebo situace, kdy uživatel moderního prohlížeče webový font nenačte – například proto, že je na velmi pomalé mobilní síti.

Myslete i na tyto případy a nikdy nezapomínejte definovat fallbackový systémový font. Například takto:

```
.element {
  font-family: 'WebFont', Georgia, sans-serif;
}
```

Font Face: tipy a triky

Pozor na falešnou kurzívu a tučný řez

Přidáváte do stránky vlastní soubory s webovými fonty (tzn. nepoužíváte cloudová řešení typu Google Fonts)? Je dobré vědět, že pokud font chcete používat v různých šířkách a variantách, musíte na to myslet v CSS deklaracích a zároveň mít připravené soubory s patřičnými řezy.

Týká se to hlavně kurzívy a tučného řezu. Pokud byste totiž deklarovali font takto…

```
  @font-face {
  font-family: 'WebFont';
  src: url('webfont.woff');
}

.element {
  font-family: 'WebFont', Georgia, sans-serif;
}
```

… a pak ho aplikovali na toto HTML, …

```
  <p>Příliš žluťoučký <b>kůň</b>
úpěl ďábelské <em>ódy</em>.</p>
```

… slovo „kůň” by se na první pohled správně vykreslilo tučně, stejně jako slovo „ódy” kurzívou fontu `WebFont`. Jenže ouha, šlo by o falešnou kurzívu a tučný řez, který se prohlížeč pokusil vytvořit z běžného řezu písma.

Pokud všechny tři řezy potřebujete, jediná správná cesta je dodat prohlížeči tři řezy – „normální“, kurzívu a tučný řez — a v CSS deklaraci mu oznámit, který soubor ke které variantě patří.

Zjednodušeně by to mohlo vypadat takto:

```
  @font-face {
  font-family: 'WebFont';
  src: url('webfont.woff');
  font-style: normal;
  font-weight: normal;
}
```

```
@font-face {
  font-family: 'WebFont';
  src: url('webfont-bold.woff');
  font-style: normal;
  font-weight: bold;
}

@font-face {
  font-family: 'WebFont';
  src: url('webfont-italic.woff');
  font-style: italic;
  font-weight: normal;
}
```

Více na css-snippets.com/web-fonts-faux-bold-and-italic.

Načítání fontů z jiných domén

Kvůli bezpečnostnímu pravidlu o servírování fontů ze stejného původu (Same Origin Policy) je v některých prohlížečích zakázáno načítat soubory s fonty z jiné domény. Například Explorer vám do konzole nahlásí:

```
   CSS3117: @font-face failed cross-origin request.
Resource access is restricted.
```

Řešením je správně nastavit `.htaccess` na doméně, z níž servírujete soubory s písmy:

```
   <IfModule mod_headers.c>
  <FilesMatch "\.(eot|otf|tt[cf]|woff2?)$">
    Header set Access-Control-Allow-Origin "*"
  </FilesMatch>
</IfModule>
```

Nastavení pro jiné servery než Apache hledejte v `server-configs` repozitářích u HTML5Boilerplate — github.com/h5bp.

Dobré zmínit, že načítání z jiných domén mírně zdrží zobrazení fontů kvůli dalšímu DNS lookupu.

Fontokuk

Chamurappiho užitečný nástroj na testování přítomnosti (nejen) českých znaků v souborech v písmech. Vyberete font z operačního systému, z Google Fonts nebo přilinkujete vlastní a také vyberete text, který v něm chcete vysázet. Ten vám pak Fontokuk vykreslí ve vybraném písmu – fontokuk.webylon.info.

Ikonfonty

Zajímavý způsob využití webových fontů představují tzv. „ikonfonty“. Namísto běžných znaků jsou v nich uložené ikonky.

Výhodou je vektorový formát, který vám umožní mít jednu verzi ikony pro všechny velikosti rodičovského prvku nebo pro všechny možné varianty vysokokapacitních displejů.

Aplikace IcoMoon vám vygeneruje fonty s ikonami na míru – icomoon.io.

Hezké využití ikonfontů je vektorová mapa Česka – cezetmap.cz.

Než se do používání ikonfontů pustíte, zvažte, zda pro vás není lepší použít rovnou vektorový formát SVG – css-tricks.com/icon-fonts-vs-svg.

Font Face: netechnické aspekty

Pokud s webfonty začínáte, dejte si pozor hlavně na dvě věci:

1. **Autorská práva.** Ne každý font existuje ve verzi dovolující publikaci na webu.
2. **Vykreslování na starších systémech.** Zkontrolujte, zda se font v dané velikosti vykresluje hezky třeba i na Windows XP (hlavně v alternativních prohlížečích) a na starších verzích Androidu. Více na blog.typekit.com/2010/10/15/type-rendering-operating-systems

Studujte typografii

Pokud sami vybíráte fonty pro váš web, rozhodně nezapomínejte na netechnické aspekty. Typografie je obor sám pro sebe a ne každý font se hodí pro každou stránku, ne každé dva fonty pasují k sobě.

Pokud hledáte zdroje ke studiu, doporučuji tyto:

- typomil.com – starší, ale platný průvodce písmem, sazbou a kompozicí na webu od Martina T. Peciny.
- Jason Santa Maria pak napsal skvělou knížku On Web Typography, pokud se o písmo chcete zajímat v širším kontextu. abookapart.com/products/on-web-typography

Kde hledat fonty?

OPTIMÁLNÍ VARIANTA: TYPEKIT.COM, FONTS.COM, PÍSMOLIJNY

Typekit, Fonts a další služby zaměstnávají špičkové typografy i techniky a vlastní licence velmi dobrých i známých fontů. Budou po vás chtít nějaké peníze, ale vyhnete se spoustě problémů.

Dobré je hledat přímo u typografů. V Česku vyrábí kvalitní webfonty třeba Tomáš Brousil (suitcasetype.com) nebo František Štorm (stormtype.com).

DOCELA DOBRÉ ALTERNATIVY: GOOGLE FONTS, FONTSQUIRREL, …

Google Fonts je fajn služba nabízející velké množství rodin písem zdarma. Ne všechny jsou ale bůhvíjak kvalitní a ne všechny mají českou lokalizaci. google.com/fonts

Fontsquirrel má jednak vlastní databázi fontů s podobnými výhodami i nevýhodami jako Google, ale taky velmi známý generátor, který vám udělá webfont z písma, které vlastníte v podobě OTF nebo jiného ne-webového formátu.

- fontsquirrel.com
- fontsquirrel.com/tools/webfont-generator

POZOR NA FONTY ZDARMA

Na internetu se jich povaluje docela dost zdarma i mimo výše uvedené služby. Ale kvalitních je málo. Font by měl mít nějaké technické (např. velikost souboru), typografické (např. optimalizace způsobu vykreslování pro web, čitelnost) a estetické kvality.

Pokud nemáte po ruce technika a typografa (nebo patřičné znalosti), můžete na své stránce leccos pokazit. Na systémových fontech také není nic špatného.

Vlastnosti pozadí

Background Clip: míra roztažení pozadí

Určuje, kde všude se uvnitř elementu vykresluje obrázek nebo barva na pozadí. Výchozí hodnota je:

```
background-clip: border-box;
```

Znamená, že pozadí se vykreslí i pod rámečkem elementu.
Další možnosti míry roztažení pozadí:
`padding-box` – jen pod obsahovým boxem a paddingem
`content-box` – jen pod obsahovým boxem
Pokud není úplně zřejmé, jaký je rozdíl mezi `border-`, `padding-` a `content-box`, mrkněte se na schéma u vlastnosti box-sizing.
Příklad k vyzkoušení – cdpn.io/e/yamFI.

Podpora v prohlížečích

IE9+. Polyfilly pro IE8 snad ani neexistují a není se čemu divit. Těžko hledat příklad, kdy by byly nezbytné.

Background Origin: pozice začátku pozadí

Určí, kde se v rámci elementu nachází začátek osy pro počítání rozměrů a pozic vlastností jako Background Size nebo `background-position`:

- `content-box` – počítá se jen obsahový box elementu
- `padding-box` (výchozí) – počítá se obsahový box a rozměry vnitřního rámečku (`padding`)
- `border-box` – k výše uvedenému se počítá i šířka vlastnosti `border`

Tip: Rozdíl mezi `content-box`, `padding-box` a `border-box` je schematicky znázorněný u vlastnosti Box Sizing.

Pozor, `background-origin` prohlížeč ignoruje, pokud u elementu zároveň nastavíte `background-attachment: fixed`.

Background Clip versus Background Origin

Všem se to plete, zkusme to tedy objasnit:

Background Clip určuje, zda barva nebo obrázek na pozadí bude vidět i pod rámečkem (`border-box`), nebo naopak jen kolem obsahového boxu (`content-box`).

Background Origin sám o sobě nic nedělá, jen definuje plochu, v jaké platí další vlastnosti (`background-size` nebo `background-position`).

IE8

Opět se musíte obejít bez podpory IE8 caniuse.com/background-origin.

Polyfill pro tuto vlastnost neexistuje, stejně tak ani částečná náhrada pomocí vlastnosti `filter`. V IE8 tedy zbývá nefunkčnost vlastnosti ignorovat nebo použít detekci vlastnosti a vymyslet alternativní řešení pro starší prohlížeče.

Background Size: velikost obrázku na pozadí

Změna velikosti obrázku na pozadí elementu.

Syntaxe

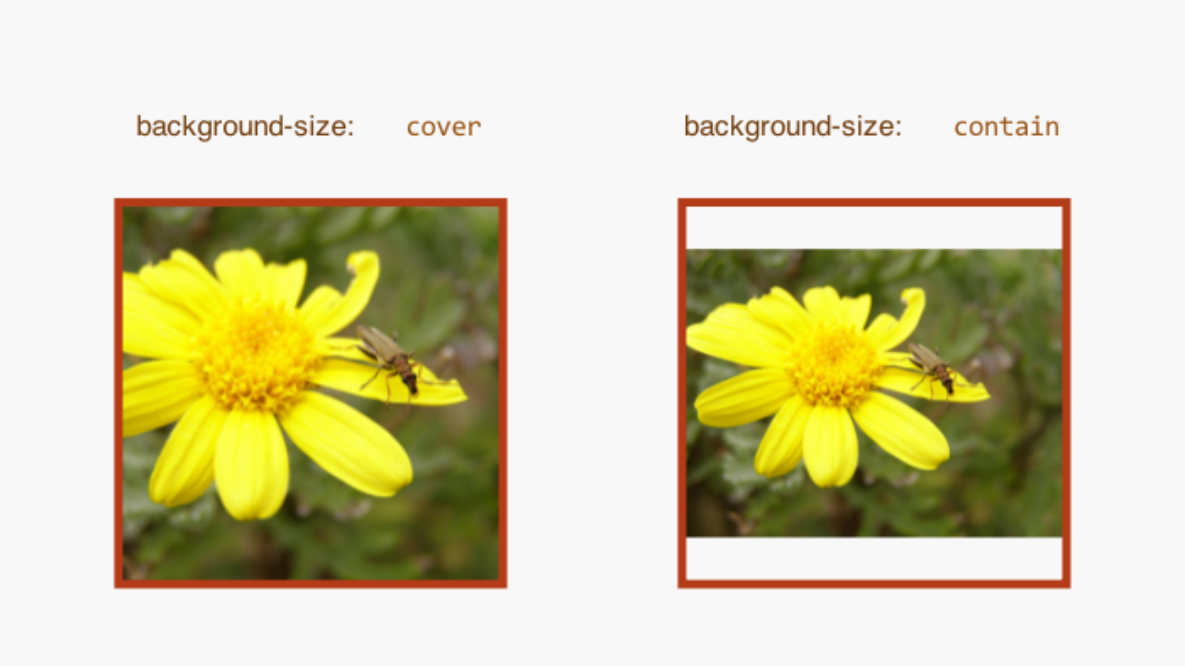
```
background-size:
(cover/contain)
(_sirka_ (_vyska_));
```

Výchozí hodnota `background-size: auto auto` znamená, že si obrázek zachová svou původní velikost.

Podobně jako u ostatních CSS vlastností můžeme zapsat výšku i šířku najednou – `background-size: auto`.

`cover` A `contain`

Klíčové slovo `cover` zajistí, aby obrázek pokryl celou plochu elementu. `contain` obstará to, aby byl obrázek vidět celý.



`background-size: cover` použijete například pro zajištění, aby obrázek na pozadí stránky pokryl celou její plochu. vrdl.in/ognci

`background-size: contain` zase v situaci, kdy máte ikonku na pozadí elementu, jenž bude na různých rozlišeních obrazovky různě velký. Nebo když chcete prohlížeči poslat dvojnásobně nebo třeba trojnásobně velký obrázek kvůli vysokokapacitním displejům. vrdl.in/ef6cp

Tip: Pokud máte tendenci takto pracovat s ikonami, porozhlédněte se po vektorovém řešení (SVG nebo ikonfonty). css-tricks.com/icon-fonts-vs-svg

ROZMĚRY V `px` NEBO PROCENTECH

Šířku a výšku je možné definovat ve standardních CSS jednotkách – `px`, `em` a dalších.

Procenta se používají relativně k šířce nebo výšce elementu, na který je vlastnost aplikována. Například roztažení barevného přechodu na celou šířku a polovinu výšky elementu zapíšeme takto:

```
  background: linear-gradient(to bottom, transparent, black)
 no-repeat bottom;
background-size: 100% 50%;
```

Naživo se podívejte na cdpn.io/e/cmpjE.

Nezapomeňte, že šířka nebo výška pozadí vychází z nastavení vlastnosti background-origin. Standardně se tedy `padding-box` a `background-size` počítá z vnitřního okraje a obsahu elementu.

VÍCE OBRÁZKŮ NA POZADÍ

Pokud používáme více obrázků na pozadí, specifikace změn jejich velikostí opět oddělujeme čárkou:

```
background-size: 50% auto, auto;
```

Podpora v prohlížečích

`background-size` zvládají všechny dnešní prohlížeče kromě IE8 – caniuse.com/background-img-opts.

`background-size` V IE8

Univerzální řešení neexistuje, na míru své situace si můžete vybrat z těchto čtyř:

Nedělat nic. Pokud obrázek dobře vyberete, nemusíte se v některých situacích trápit tím, že se v IE8– nezmenší.

Detekce vlastností. Poskytnout alternativní verzi stylování pomocí Modernizru – `.no-backgroundsize .element { … }`.

Využít parametru `filter`. Hodí se jen pro situace, kdy obrázek na pozadí máte ve stejném poměru stran a zároveň stejně velký nebo větší než rodičovský objekt:

```
  .element {
 background-size: contain;
 filter: progid:DXImageTransform.Microsoft.AlphaImageLoader(
```

```
    src='images/image.jpg', sizingMethod='scale');
}
```

Využít polyfill. Jen pozor, využívá `.htc` soubory, takže může nepřiměřeně zhoršovat výkon – github.com/louisremi/background-size-polyfill.

Gradients: barevné přechody

Způsob, jak kódem vykreslit barevný přechod – jinak též gradient – všude, kde jsme donedávna používali externí obrázek.

Lineární přechod

Rovnoměrný barevný přechod seshora dolů uděláte takto:

```
background: linear-gradient(lightgreen, darkgreen);
```

SMĚR OSY BAREVNÉHO PŘECHODU

Používají se buď **klíčová slova** označující směr gradientu (`to bottom right`, `to right`) nebo **úhly**. Úhel 0° vede zezdola nahoru, 90° zleva doprava a tak dále po směru hodinových ručiček. Přednastavený je 180° vedoucí seshora dolů. V CSS zápisu `180deg`.

Například tento gradient ze světle do tmavě zelené povede z levého dolního rohu směrem k pravému hornímu:

```
background:
linear-gradient(45deg, lightgreen, darkgreen);
```

ZARÁŽKY BAREV

Můžeme samozřejmě ovlivnit, jak se nám jednotlivé barvy rozloží na ose průběhu přechodu. Slouží k tomu zarážky, definovatelné v běžných CSS jednotkách (`%`, `px`, `em`…). Je to stejná zarážka, jakou možná znáte z grafických editorů, jen vyjádřená kódem.

```
background:
linear-gradient(45deg, lightgreen, darkgreen 33%);
```

Barevná zarážka pro tmavě zelenou barvu tady začíná na třetině délky osy gradientu. Příklad si můžete naživo vyzkoušet nebo upravovat na cdpn.io/e/CcdBf.

BARVY

Nemusíme definovat samozřejmě jen dvě barvy. Může jich být libovolný počet. Nezapomeňte, že máte k dispozici všechny způsoby definování barev včetně rgba nebo průhledné `transparent`.

```
  background:
 linear-gradient(to bottom right, transparent,
   lightgreen 25%, rgb(0, 127, 0) 50%);
```

Tento zápis vykreslí gradient z levého horního do pravého dolního rohu. Do čtvrtiny délky úhlopříčky elementu se vykreslí přechod z průhledné do světle zelené barvy. Od čtvrtiny do poloviny pak přechod ze světle zelené do tmavě zelené, jen tentokrát zapsané pomocí RGB barevného modelu.

Kruhový přechod

Jednoduchý kruhový (radiální) přechod vytvoříme například tímto zápisem:

```
.box-1 {
 background:
   radial-gradient(lightgreen, darkgreen);
}
```

TVAR A VELIKOST

Přednastavený tvar přechodu je kružnice `circle`. Lze přenastavit na `ellipse` – elipsovitý tvar.

Hned za tvarem je možné definovat velikost přechodu. První možnost je definovat **velikost jako poloměr**. U kružnice jedním, u elipsy dvěma čísly. První udává výšku, druhý šířku elipsy.

```
.box-2 {
 background:
   radial-gradient(ellipse 50px 30px,
     lightgreen, darkgreen);
}
```

Druhá možnost je definovat **velikost klíčovým slovem**:

- `closest-side` – přechod bude končit u nejbližší strany elementu
- `farthest-side` – přechod bude končit u vzdálenější strany elementu
- `closest-corner` – přechod bude končit u nejbližšího rohu elementu
- `farthest-corner` – přechod bude končit u nejvzdálenějšího rohu elementu

POZICE STŘEDU

Pozice středu barevného přechodu se definuje podobně jako u vlastnosti `background-position`. Je potřeba ji jen doplnit o klíčové slovo `at`:

```
.box-3 {
 background:
   radial-gradient(at top left, lightgreen, darkgreen);
}
```

ZARÁŽKY BAREV

Fungují podobně jako u lineárního přechodu. Do čtvrtiny rozměrů elementu prohlížeč vykreslí světle zelenou kružnici, mezi čtvrtinou a polovinou barevný přechod mezi světle a tmavě zelenou a ve zbytku elementu uvidíte tmavě zelenou:

```
.box-4 {
 background:
   radial-gradient(lightgreen 25%, darkgreen 50%);
}
```

A tady je živý příklad, obsahující všechny čtyři varianty radiálního přechodu: cdpn.io/e/cdyfx.

Opakující se barevné přechody

Deklarují se úplně stejně jako běžné barevné přechody, jen pomocí funkcí `repeating-linear-gradient()` nebo `repeating-radial-gradient()`. Narozdíl od běžných přechodů prohlížeč od poslední barevné zarážky nevykreslí barvu, kterou obsahuje, ale znovu opakuje definovaný gradient.

```
.repeating-linear {
 background:
   repeating-linear-gradient(to bottom right,
     transparent, transparent 10%,
     green 10%, green 20%);
}
```

V tomto příkladu vykreslujeme barevnou „zebru“, pomocí průhledné a zelené barvy.

Všimněte si také, že jsme nadefinovali těsně sousedící zarážky (`transparent 10%, green 10%`). To znamená, že gradient bude „ostrý“. Jinak řečeno se sousedící barvy vykreslí vedle sebe bez barevného přechodu.

Pojďme ještě zkusit opakující se kruhový gradient:

```
.repeating-radial {
 background:
   repeating-radial-gradient(transparent,
```

```
        transparent 10%, green 10%, green 20%);
}
```

Opakujícími se barevnými a zelenými plochami se nám vykreslí „terč“. Tady ovšem pozor, některé prohlížeče (konkrétně Chrome nebo Firefox v době psaní článku) zatím neumějí tyto složitější gradienty vyhlazovat, takže hrany kružnic budou „kostrbaté“.

Podpora v prohlížečích

Barevné přechody neumí IE ve verzi 8 a 9 nebo Opera Mini. Android Browser 2.3 opakované gradienty nezvládne vůbec a v podpoře běžných gradientů má také mezery: caniuse.com/gradients.

Nezapomeňte tedy vždy definovat fallback. Gradient se považuje za obrázek na pozadí, takže si můžete fallback definovat jako běžnou barvu:

```
   color: #fff;
background-color: green;
background-image:
  linear-gradient(lightgreen, darkgreen);
```

GRADIENT, DŘÍVE ČERT Z PREFIXOVÉHO PEKLA

Dnes už to takový problém není, ale každý prohlížeč v různých fázích vývoje implementoval různé fáze vývoje specifikace. Nebo vlastní návrh syntaxe. Takže pokud chcete podporovat i starší verze moderních prohlížečů, věnujte zvýšenou pozornost prefixovým variantám.

STARŠÍ SYNTAXE PROHLÍŽEČŮ POSTAVENÝCH NA WEBKITU

Pokud někde chcete plně podporovat starší Chrome, Safari do verze 5, iOS Safari do verze 4, Android Browser do verze 3 a dalších několik prohlížečů, musíte použít jejich starší syntaxi. Jen pozor, liší se nejen prefixem, ale také způsobem zápisu. Například směr osy se určuje deklarováním růžku nebo strany, ze které gradient začíná:

```
   background-image:
  -webkit-linear-gradient(top, lightgreen, green);
```

U většiny webů to asi nebudete muset řešit a spokojíte se s fallbackem pomocí definované barvy. Poslední verze všech prohlížečů se shodují na W3 syntaxi, kterou používáme v příkladech. A bez prefixů!

`filter` V IE8 A IE9

Jednoduché, dvoubarevné lineární gradienty lze ve starších Explorerech zařídit s pomocí proprietární vlastnosti filter:

```
-ms-filter:
"progid:DXImageTransform.Microsoft.gradient(
  GradientType=0, startColorstr=#00ff00, endColorstr=#008800)";
```

V parametru `GradientType` nastavujete svislý (`0`) nebo vodorovný (`1`) směr gradientu. U filtrů jen pozor na pomalejší vykreslování a na fakt, že `background-image` účinnost filtrů ruší.

Tipy, triky a nástroje

Nezapomeňte, že gradient je vlastně **obrázek na pozadí elementu**, takže ho můžete použít pro definování obrázku odrážky (`list-style-image`) nebo pro obrázek na pozadí rámečku (border-image).

Nejobvyklejší netriviální použití gradientů jsou **grafická tlačítka** vykreslená pomocí CSS: cubiq.org/dropbox/cssgrad.html.

Takřka **vědecké povídání o gradientech**. Ana Tudor jde v následujícím odkazu pořádně do hloubky a na pomoc si bere matematiku: hugogiraudel.com/2013/02/04/css-gradients.

ColorZilla Gradient Editor vám pomůže vygenerovat kód gradientu i pro starší prohlížeče, včetně fallbacku pro IE8 a IE9: colorzilla.com/gradient-editor.

Lea Verou má hezkou galerii **barevných vzorů** vytvořených jen s pomocí gradientů. Berte to ale raději jen jako ukázku možností: lea.verou.me/css3patterns.

Právě zmíněné barevné vzory často využívají tzv. **ostrý přechod**, což je přechod-nepřechod, ve kterém je mezi barvami ostrá hrana: `background: linear-gradient(to bottom, transparent, lightgreen 33%, darkgreen 33%);` cdpn.io/e/licEd.

CSS Scales jsou předdefinované barevné přechody. Hezké, ano. Navíc ovšem při jejich vymýšlení mysleli na přístupnost a barvy jsou vhodné pro barvoslepé lidi. bennettfeely.com/scales.

Multiple Backgrounds: více obrázků na pozadí

Vrstvení více obrázků nebo barev na pozadí jednoho elementu.

Není to přímo CSS3 vlastnost, jen nová možnost již existující vlastnosti `background`.

Syntaxe? Je to snadné, vrstvy oddělujeme čárkou:

```
background:
  url('obrazek_nahore.png'),
  url('obrazek_uprostred.png'),
  #ddccaa;
```

Pozadí před první čárkou je vždy vrstva nejvíc nahoře.

Pokud nepoužijeme shorthand `background`, deklarace dalších vlastností obrázku na pozadí se pro jednotlivé vrstvy rovněž oddělují čárkou:

```
background-image:
  url('obrazek.png'),
  url('dalsi_obrazek.png');
background-repeat:
  no-repeat,
  repeat;
```

Příklad k vyzkoušení

Nezapomeňte, že obrázkem může být i CSS3 gradient s poloprůhledným pozadím. Toho lze využít pro efekt postupného překrytí obrázku, i když neznáte výšku elementu:

```
background:
  linear-gradient(180deg, transparent 0%, #333 100%),
  url('bg.jpg');
```

Naživo zkoušejte na cdpn.io/e/lvKkC.

Podpora v prohlížečích

IE9+. Pozor, vlastnost `background` s vícenásobnou hodnotou je ignorována, pokud ji prohlížeč neumí. Pro starší prohlížeče jako IE8 vždy musíte definovat fallback. Například:

```
   background: #ddccaa;
background:
  url('obrazek_nahore.png'),
  url('obrazek_uprostred.png'),
  #ddccaa;
```

Vlastnosti rámečků

Border Image: rámeček vykreslený obrázkem

Jde o způsob, jak kolem elementu vykreslit vlastní namísto nativních rámečků.

Vezmeme jakýkoliv obrázek obsahující rámeček a prohlížeči řekneme, jak jej má naporcovat. Následuje kouzlo – rámeček se elegantně přizpůsobí šířce i výšce elementu, ať je jakákoliv.

Syntaxe

```
border-image:
_zdrojovy_obrazek_
_rozmery_rezu_
_sirka_ramecku_
_zacatek_rezu_
_opakovani_
```

ZDROJOVÝ OBRÁZEK

Jediná povinná vlastnost. Samostatně jako `border-image-source: url(…)`.

ROZMĚRY ŘEZŮ

Právě touto hodnotou zdrojový obrázek naporcujeme tak, aby prohlížeč věděl, kde leží výřezy pro všechny čtyři rohy a kde výřezy pro svislou a vodorovnou stranu rámečku.

Obsahuje jedno, dvě (vodorovný a svislý řez) nebo čtyři čísla (řez horní, pravý, dolní a levý). Definovat lze v pixelech nebo v procentech z rozměrů zdrojového obrázku. Výchozí stav je `border-image-slice: 100%`.

Pokud je přítomno klíčové slovo `fill`, vezme se ze zdrojového obrázku i jeho střední část a vykreslí se přes pozadí elementu.

ŠÍŘKA RÁMEČKU

`border-image-width` specifikuje šířku rámečku v pixelech nebo procentech, podobně jako u `border-width`.

Pokud má hodnotu `auto`, šířka se počítá z rozměrů řezů.

ZAČÁTEK ŘEZU

Rozměr specifikovaný v `border-image-outset` říká, jak moc obrázkový rámeček přetéká mimo rozměry elementu. Ty se počítají, jako by měl element nastaveno `box-sizing: border-box`.

OPAKOVÁNÍ

Jak bude prohlížeč pracovat se svislou a vodorovnou stranou obrázkového rámečku, pokud má rámeček jiné rozměry než zdrojový obrázek? To můžeme nastavit v `border-image-repeat`. Možnosti jsou tyto:

- `stretch` – obrázek se roztáhne na šířku rámečku
- `repeat` – obrázek se bude opakovat
- `round` – pokud plochu nevyplní celočíselný počet opakování, jednotlivá opakování se roztáhnou, aby plochu vyplnila
- `space` – pokud plochu nevyplní celočíselný počet opakování, prázdná plocha je spravedlivě rozdělena mezi všechna opakování (k jednotlivým opakováním se přidá mezera)

Je dobré připomenout, že i tady je možné nastavit různé hodnoty pro vodorovnou i svislou část rámečku. Například:

```
border-image-repeat: stretch repeat;
```

Může se hodit

- Moc fajn generátor, který vám ulehčí život hlavně při hledání rozměrů řezů — border-image.com
- Pozor, `border-image` nebude podle aktuální verze specifikace fungovat, pokud u stylovaného elementu zapomenete na deklaraci vlastností `border-style` a `border-width`.

Podpora v prohlížečích

IE11+. Se staršími prohlížeči se lze vypořádat definovanou alternativou a detekcí vlastnosti Modernizrem: `.no-borderimage .box { … }` nebo prostým fallbackem pomocí vlastnosti `border-color`.

Jednoduchý příklad s barevným přechodem

Protože CSS gradienty se mezi obrázky počítají také, můžete namísto rámečku použít barevný přechod.

Pamatujte, že vždy je nutné nejprve definovat nativní rámeček obrázku. Jednak kvůli rozměrům, jednak tím vytvoříme fallback pro prohlížeče, které `border-image` nezvládají. V našem příkladu tedy kolem elementu nejdříve vyrobíme 20pixelový zelený rámeček:

```
border: 20px solid green;
```

Teď prohlížeči řekneme, že namísto zelené barvy chceme v rámečku barevný přechod:

```
border-image-source:
linear-gradient(lightgreen, darkgreen);
```

K našemu překvapení ovšem prohlížeč barevný přechod vykreslí jen v rozích rámečku. Důvodem je výchozí hodnota rozměrů řezů: `border-image-slice: 100%`. Znamená, že obrázek se použije právě jen pro všechny čtyři růžky. Předefinujeme tedy tak, aby odpovídal šířce našeho rámečku:

```
border-image-slice: 20;
```

A je to. Příklad si můžete vyzkoušet na cdpn.io/e/zdyIJ.

Příklad s bitmapovým obrázkem na pozadí

Opět si nejdříve nadefinujeme rozměry rámečku a fallback pro staré prohlížeče:

```
   border-color: green;
border-style: solid;
border-width: 21px 23px;
```

Přidáme obrázek na pozadí:

```
   border-image-source: url(border-image-source.png);
```

Dále definujeme řezy. V tomto zdrojovém obrázku máme vodorovný rámeček vysoký 21 pixelů a svislý 23 pixelů.

```
   border-image-slice: 21 23;
```

Nakonec je potřeba prohlížeči oznámit, že postranní řezy hodláme v případě nárůstu velikosti elementu opakovat:

```
   border-image-repeat: repeat;
```

A pojďme si ještě vyzkoušet zkrácený zápis posledních tří deklarací:

```
   border-image: url(border-image-source.png) 21 23 repeat;
```

Hotovo. Příklad si můžete vyzkoušet na cdpn.io/e/DLkjm.

Border Radius: poloměr rohu rámečku

Vykreslování zakulacených a elipsovitých rohů elementu.

Syntaxe

```
border-radius:
_polomer_vlevo_nahore_
_polomer_vpravo_nahore_
_polomer_vpravo_dole_
_polomer_vlevo_dole_
(/ _elipsove_varianty_);
```

Rovnoměrné zakulacení rohů s poloměrem 10 pixelů vyšvihneme takto:

```
border-radius: 10px;
```

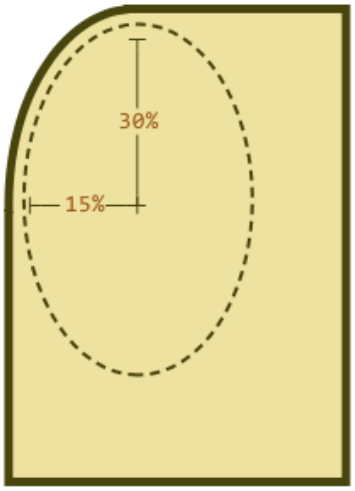
Zakulacovat ovšem můžeme i v procentech ze šířky elementu a v dalších jednotkách dostupných v CSS. Zakulacovat můžeme **pro každý růžek zvlášť**. První je vždy levý horní a postupuje se po směru hodinových ručiček:

```
border-radius: 15% 15% 0 0;
```

Přidáním lomítka zajistíme **zakulacení ve tvaru elipsy**, nikoliv kružnice. První růžek bude zakulacený v elipse se svislým poloměrem 15 % a vodorovným 30 %:

```
border-radius: 15% 15% 0 0 / 30% 15% 0 0;
```

Na následujícím schématu je patrné, jak se podle elipsy v praxi zaobluje:




```
border-radius:15% 0 0 0 / 30% 0 0 0;
```

Je dobré vědět, že `border-radius` je ve skutečnosti zkratka pro deklaraci vlastností jednotlivých rohů. Můžeme je samozřejmě **nastavit samostatně**:

```
   border-top-left-radius: 4em;
border-top-right-radius: 4em;
border-bottom-right-radius: 4em;
border-bottom-left-radius: 4em;
```

Živý příklad se zaoblenými rohy najdete na cdpn.io/e/EljFa.

Tipy a triky

Jak pomocí `border-radius` vykreslit **kruhové avatary**? vrdl.in/uncsg

Jak na **tabulky se zaoblenými rohy**? Na tabulky s `border-collapse: collapse` a rodičovské prvky s obrázkem uvnitř je potřeba aplikovat `overflow: hidden`. cdpn.io/e/jpdFm

Podpora v prohlížečích

Podpora v moderních prohlížečích je bezproblémová. Pokud v osmičkovém Exploreru zaoblené rohy nutně potřebujete, použijte css3pie.com, ale pozor na neblahý vliv na výkonnost stránky.

Velmi tedy doporučuji strategii nulového fallbacku. Uživatelé starších prohlížečů prostě zakulacené rohy neuvidí, a co oči nevidí, to srdce nebolí.

Pokud vám v některých prohlížečích pod zaobleným rohem prosvítá barva pozadí, přidejte `background-clip: padding-box`. vrdl.in/ls2uc

MSIE9 sice `border-radius` podporuje, ale není je možné kombinovat s vlastností `filter` používanou například pro barevné přechody. Dá se to vyřešit nastavením stejného `border-radius` a `overflow: hidden` pro rodičovský element.

Vlastnosti boxu

Box Shadow: stínování elementu

Jde o obvyklý stín nejen pod elementem, ale i uvnitř elementu nebo plastický efekt přes element.

Syntaxe

```
box-shadow:
(inset)
_posun_x_
_posun_y_
(_rozostreni_)
(_roztazeni_)
_barva_,
(_dalsi stin_);
```

Základní stín vytvoříte co by dup. První číslo udává posun po vodorovné – doprava. Druhé po ose svislé – dolů. Záporná čísla stín posunují doleva, respektive nahoru. Třetí je barva a věžte, že pro stíny se nejvíce hodí poloprůhledná RGBa barva:

```
box-shadow: 5px 5px rgba(0, 0, 0, .5);
```

Pokud přidáme další číslo, prohlížeč pozná, že jde o **rozostření** stínu:

```
box-shadow: 5px 5px 10px rgba(0, 0, 0, .5);
```

Ještě jedno číslo a stínu tak nadefinujeme i **roztažení** do stran:

```
box-shadow: 5px 5px 10px 10px rgba(0, 0, 0, .5);
```

Klíčové slovo **inset** pak zajistí, aby se stín vykreslil uvnitř elementu:

```
box-shadow: inset 5px 5px 10px 10px rgba(0, 0, 0, .5);
```

Stíny můžeme **vrstvit**, stačí je oddělit čárkou. První stín je ten nejvíc nahoře:

```
box-shadow: 5px 5px 10px 10px rgba(0, 0, 0, .5),
 inset 5px 5px 10px 10px rgba(0, 0, 0, .5);
```

Živá ukázka příkladu je na cdpn.io/e/lAoDv.

Tipy a triky

STÍN JEN NA JEDNÉ STRANĚ OBJEKTU

Tady chceme stín jen na levé straně. Je to jednoduché – horizontální stín nastavíme na `0`. I přesto je ale díky použití rozostření stín malinko vidět nahoře i dole:

```
box-shadow: 5px 0 5px -2px rgba(0,0,0,.5);
```

Živá ukázka příkladu je na cdpn.io/e/JnGyb.

STÍN JAKO KOPIE OBJEKTU

Kreslit stínem logo Microsoftu je samozřejmě nepraktické, ale hezky to ukazuje sílu řetězení a taky co se stane, když nepoužijeme rozostření. cdpn.io/e/qJuzw

Můžete samozřejmě kombinovat nerozostřené i rozostřené stíny. dabblet.com/gist/2043600

Podpora v prohlížečích

IE9+. Podpora v moderních prohlížečích je téměř bezproblémová. caniuse.com/box-shadow

STARŠÍ WEBKIT PROHLÍŽEČE OBČAS IGNORUJÍ ROZTAŽENÍ

Drobným problémem je jen ignorace nulové hodnoty roztažení v případě nepřítomnosti hodnoty rozostření ve starších prohlížečích postavených na jádře Webkit. V Safari na iOS6 nebo třeba Android Browseru 2.3 nebude fungovat zápis:

```
box-shadow: 5px 5px 0 rgba(0, 0, 0, .5);
```

Tento ovšem ano:

```
box-shadow: 5px 5px 10px rgba(0, 0, 0, .5);
```

Živá ukázka příkladu je na cdpn.io/e/FGtbu.

INTERNET EXPLORER 8

V IE8 můžete stín nechat vykreslit pomoc proprietární vlastnosti `filter`. Například:

```
filter: progid:DXImageTransform.Microsoft.Shadow(color='#cccccc',
Direction=145, Strength=3);
```

Samozřejmě ne všechny typy stínů takto nahradíte.

Vykreslovat stíny ve starších Explorerech umí i polyfill css3pie.com.

Obvykle si ovšem u stínů vystačíte se strategií nulového fallbacku.

Box Sizing: způsob počítání velikosti boxu

Změna způsobu počítání šířky a výšky elementu, jinak též řečeno box-modelu.
Dozvíte se, proč `box-sizing: border-box` milují vývojáři, kteří dělají fluidní layout, a taky nepřátelé počítání. Čtěte dále.

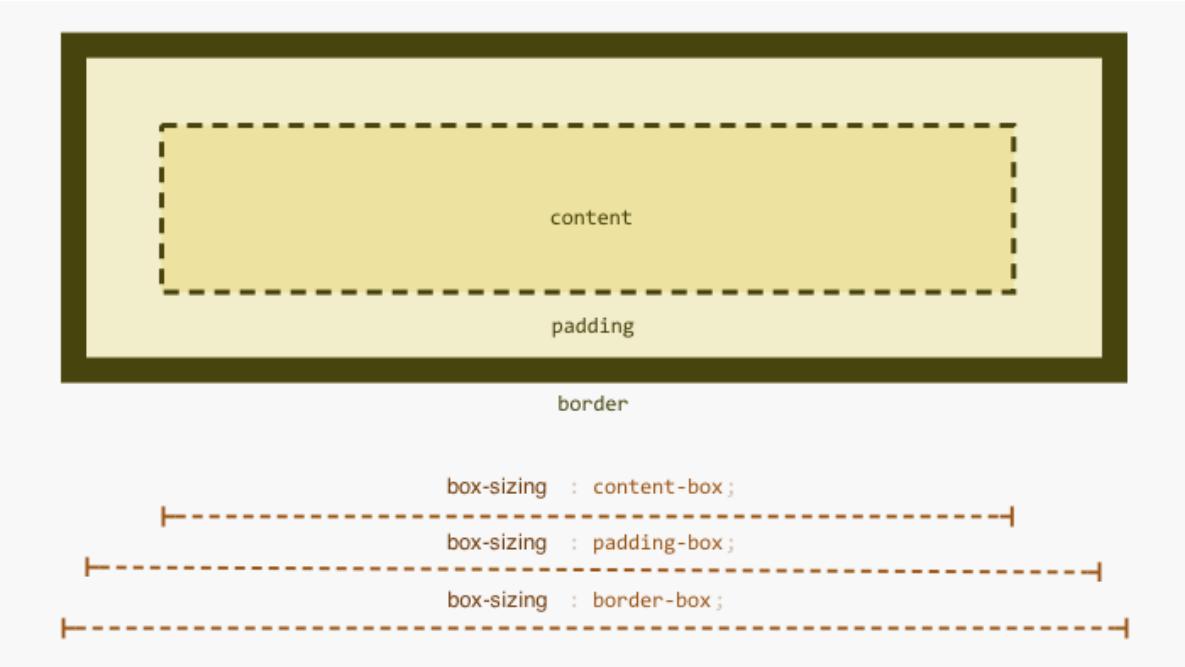
Syntaxe

```
box-sizing: content-box | border-box | padding-box;
```

Vzpomínáte na tradiční box-model, který počítal IE6 a starší v nestandardním módu? Nevzpomínáte? Gratuluji, jste šťastní lidé. wikipedia.org/wiki/Internet_Explorer_box_model_bug

```
Šířka nebo výška elementu =
viditelná šířka nebo výška obsahu
+ padding + border.
```

Už víte? To je **`border-box` box-model**.



Naproti tomu **`content-box` neboli „W3C box-model“** používají všechny moderní prohlížeče. Výpočet znáte:

```
Šířka nebo výška elementu =
viditelná šířka nebo výška obsahu.
```

A to je taky přednastavená hodnota vlastnosti `box-sizing`, kterou – naštěstí – můžeme změnit.

Pro pořádek uveďme, jak se počítá šířka a výška elementu u `box-sizing: padding-box` – je to vlastně `border-box`, kde se do výpočtu nepřipočítá šířka vlastnosti `border`.

Dobře, ale jak to můžeme využít? Podívejme se na několik možných scénářů.

Příklady využití

```
* { box-sizing: border-box }
```

Někdo využívá vlastnosti box-sizing v situaci, kdy se mu špatně pracuje s W3C box modelem. Ten totiž významná část webových vývojářů považuje za neintuitivní. Takoví pak prohlížeče nechávají počítat v border-box všechny elementy. Podobný přístup mají i moderní frontend frameworky Bootstrap nebo Foundation.

FLUIDNÍ LAYOUT

Mnoho využití má tato vlastnost v responzivním webdesignu, konkrétně při práci s layoutem definovaným v procentuálních jednotkách. Představte si například navigaci, jež má vždy 5 položek. Šířka jedné pak bude 20 %. Oddělovač mezi položkami je vytvořený rámečkem fixní šířky:

```
 .nav li {
  width: 20%;
  display: inline-block;
  border-left: .25em solid #fff;
}
```

Jenže takhle nám pátá položka navigace odskočí na další řádku. Potřebujeme však jen prohlížeči oznámit, ať laskavě šířku položek navigace počítá pomocí `box-sizing: border-box`:

```
 .nav li {
  box-sizing: border-box;
  width: 20%;
  display: inline-block;
  border-left: .25em solid #fff;
}
```

Živá ukázka příkladu je na cdpn.io/e/FeLkJ.

ZMĚNA POČÍTÁNÍ ROZMĚRŮ FORMULÁŘOVÝCH ELEMENTŮ

Vlastnost `box-sizing` se moc hodí na sjednocení způsobu počítání výšky nebo šířky formulářových elementů. Některé z nich totiž prohlížeče počítají jako `content-box` a některé `border-box` způsobem (např. `input type=„submit“`nebo `select`). Pokud chcete zajistit stejnou výšku formulářových prvků ve vašem designu, než je začnete stylovat, přepněte si je nejlépe do `box-sizing: border-box`. Živá ukázka problému s formulářovými elementy je na cdpn.io/e/iBquK.

Podpora v prohlížečích

IE7+ a všechny moderní prohlížeče. Pokud jste vlastnost neznali, budete se divit, jak výborně je podporována. caniuse.com/box-sizing

Je ale dobré vědět, že méně používanou hodnotu `padding-box` podporuje jen Firefox.

Media Queries

Dotazy na media umožňují za určité podmínky aplikovat jiná CSS pravidla a ve výsledku změnit vzhled elementu.

Dejme si rychlý příklad:

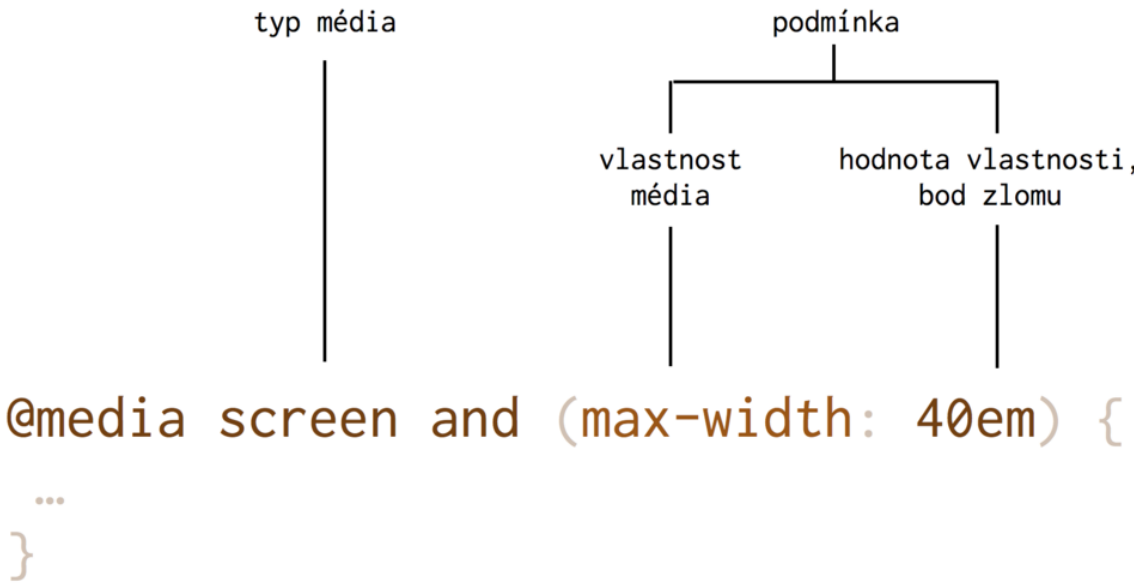
```
   h1 { font-size: 2em }

@media only screen and (max-width: 40em) {
  h1 { font-size: 1.5em }
}
```

Takto zmenšíme nadpis první úrovně pro okna prohlížeče do šířky 40 čtverčíků, což je v přepočtu 640 pixelů. Toto jednoduché použití dotazů na média si můžete vyzkoušet v živé ukázce. cdpn.io/e/Bpajbz

Z CSS2 budete znát podmínky pro typy médií, jako třeba `@media print`. Norma CSS3 Media Queries je vylepšuje o bližší specifikaci vlastnosti médií.

Anatomie media query

Dotaz na medium (anglicky *media query*) se skládá z typu média (*media type*, výchozí je `all`) a podmínky obsahující vlastnosti média (*media features*) s hodnotou nebo rozmezím hodnot.



Obrázek: Anatomie Media Query. Pro zjednodušení jsem odstranil klíčové slovo „only“, které ze zpracování dotazu vyloučí starší Internet Explorery.

Body zlomu

Známý je také výraz bod zlomu (*breakpoint*), což je hodnota vlastnosti média. O „breakpointech“ mluvíme jako o sadě hodnot pro konkrétní web nebo systém designu. Bootstrap má například tyto přednastavené (a nastavitelné) body zlomu: extra small (šířka okna do 767 pixelů), small (768 – 991), medium (992 – 1199) a large (1200 a více).

Minimální nebo maximální výška a šířka

Klasické podmínky v responzivním designu vypadají jako dotazy na vodorovný nebo svislý rozměr okna:

```
  @media only screen and (min-width: 40em) { … }
@media only screen and (max-width: 40em) { … }
@media only screen and (min-height: 40em) { … }
@media only screen and (max-height: 40em) { … }
```

V drtivé většině případů se pracuje jen s vodorovnými rozměry, s vlastností `width`. Ale ukážeme si i další možnosti. Vyčkejte, předtím se totiž naučíme jak kombinovat dotazy.

Logické operátory

Dotazy na médium můžeme pomocí operátoru `and` kombinovat:

```
@media only screen
and (min-width: 30em) and (max-width: 40em) { … }
```

Podmínka se aplikuje jen na všechny zobrazovací média se šířkou okna mezi třiceti a čtyřiceti čtverčíky.

A co „nebo“? Místo možná očekávaného `or` se používá čárka:

```
@media only screen and (max-width: 40em), print { … }
```

Podle CSS specifikace jde o „seznam oddělený čárkou“, kde se jednotlivé položky seznamu vyhodnocují samostatně. Čárka („or“) má proto větší váhu než `and`. Výše uvedený dotaz se tak vyhodnotí jako pravdivý, když budeme na zobrazovacím mediu o šířce viewportu do 40 čtverčíků nebo když budeme stránku tisknout.

Dalším možným operátorem je negace:

```
@media not print { … }
```

Jen pozor, přátelé. Negace vždy postihuje celý dotaz, nikoliv jeho určitou část.

Další vlastnosti média

Detekce orientace zařízení

Drží uživatel zařízení na výšku, nebo na šířku?

```
  @media only screen and
  (orientation: portrait) {
  /* Držení na výšku */
}

@media only screen and
  (orientation: landscape) {
  /* Držení na šířku */
}
```

Podmínka pro poměr stran obrazovky

Obrazovky s poměrem stran 16:9 například zacílíme takto:

```
  @media only screen and
  (device-aspect-ratio: 16/9) { … }
```

Existují samozřejmě i varianty pro rozmezí hodnot: `min-aspect-ratio` a `max-aspect-ratio`.

Detekce vysokokapacitních displejů

Moderní displeje s vyšším počtem hardwarových pixelů, tedy displeje typu Retina, Amoled a další můžeme detekovat takto:

```
  @media only screen and
  (min-resolution: 2dppx) { … }
```

Aplikuje se, pokud má zařízení poměr mezi hardwarovými a CSS pixely alespoň 2. Pokud byste náhodou ztráceli nit, na Vzhůru dolů je o CSS pixelu více informací. vrdl.cz/prirucka/css-pixel

Poměrů je ale dnes celá řada (1,25; 1,5; 2; 3; 4…). Proto doporučuji namísto dotazu na vlastnost `resolution` v kombinaci s bitmapovými obrázky využívat vektorový formát SVG. U vektorových obrázků totiž vlastnost `resolution` řešit nemusíme. vrdl.cz/prirucka/svg.

A co další vlastnosti médií?

V textu jsme zvládli ty nejpoužívanější. Z dalších zajímavých budu jmenovat hlavně sadu vlastností pro detekci způsobu ovládání. Například `@media (hover:hover)`. Tam se ale čeká na podporu Firefoxu. caniuse.com/media-interaction

Vlastností médií existuje ale mnohem víc, i když ty ostatní už tak moc použitelné nejsou. jecas.cz/media

Na co si dát u dotazů pozor?

1. Zápis vynechávající typ média

Tohle může být špatně:

```
@media (min-width: 40em) { … }
```

Podmínka se pak aplikuje na všechna média. Tedy nejen obrazovková, ale také výstupy pro tisk. Výchozí typ média je totiž `all`, nikoliv `screen`. Tyto dva zápisy jsou tedy ekvivalentní:

```
  @media all { … }
@media { … }
```

Klíčové slovo only před médiem vyřazuje ze hry staré Explorery (verze šest a starší), které by se jinak tvářily, že podmínce rozumějí. Občas je krátký zápis bezproblémový, obecně se mu ale raději vyhýbejte.

Správný zápis vypadá následovně:

```
@media only screen and (min-width: 40em) { … }
```

2. Cílení na rozlišení obrazovky

```
@media … (min-device-width: 40em) { … }
```

„Device“ zápis cílí na rozlišení obrazovky, nikoliv na velikost okna prohlížeče. Hodně dávno se to používalo pro „detekci“ konkrétních zařízení, což ale dobrý postup nebyl, není a nebude. Rozlišení obrazovek je dnes děsně moc a nedá se z nich vyčíst, jestli zařízení patří mezi mobily, tablety nebo něco jiného.

3. Rozdělování CSS podle velikosti obrazovky

```
<link rel="stylesheet" href="mobile.css"
media="max-width: 40em">
```

Občas ještě někde vídávám. Organizace CSS kódu přes velikosti zařízení už ale vytvořilo nejednu vrásku na čele webových vývojářů. Kód jedné komponenty rozhraní je pak rozdělený do více souborů a dost špatně se to spravuje.

Jediný rozumný způsob organizace stylů je podle komponent uživatelského rozhraní jako jsou navigace, tlačítka atd. Pokud je to pro vás nové, čtěte můj blogpost o organizaci CSS. vrdl.cz/blog/29-organizace-css-2014

Podpora ve starších prohlížečích: nejlépe s Respond.js

Internet Explorer 8 a starší, budiž jim země lehká, neumějí ani základní vlastnosti médií z CSS3 Media Queries. Pokud to na svém projektu potřebujete řešit, doporučuji použít polyfill Respond.js. Je odzkoušený a dostatečně rychlý. Používá jej například i populární frontend framework Bootstrap. github.com/scottjehl/Respond

```
   <!--[if lte IE 8]>
<script src="respond.js"></script>
<![endif]-->
```

Dejte si pozor na místo vložení. Skript by měl být hned za odkazem na CSS soubor.

Media Queries v Javascriptu

V Javascriptu používejte funkci `matchMedia()`, která přijímá stejné podmínky jako CSS:

```
if (matchMedia('only screen and (max-width: 40em)').matches) {
  // …
}
```

Častou chybou je detekce podle šířky nebo výšky okna a následné spoléhání na událost `onresize`. Je to v kódu zbytečně složité a při zmenšování nebo zvětšování okna výkonnostně problematické. `matchMedia()` sice opět nemá podporu v IE8 a starších, ale na to máme polyfill, který je také součástí Respond.js. github.com/paulirish/matchMedia.js/

Transformace

Transforms: proměny objektu

Transformace tvaru, pozice nebo velikosti elementu.
Existují čtyři funkce: zkosení, otočení, posun a změna velikosti:

```
  /* Zkosení */
.skew {
  transform: skew(-15deg);
}

/* Otočení */
.rotate {
  transform: rotate(-15deg);
}

/* Posun */
.translate {
  transform: translate(0, 50%);
}

/* Změna velikosti */
.scale {
  transform: scale(1.5);
}
```

Vyzkoušejte si: cdpn.io/e/wxoil.
Všechny čtyři základní funkce mají varianty pro transformaci jen po jedné ose – například `skewX()`, `skewY()`.

Kombinace transformací

Je dobré si zapamatovat, že se kombinace proměn neoddělují čárkou:

```
transform: scale(1.5) skew(-15deg);
```

Původ transformace

Souřadnice bodu, ze kterého transformace vychází. Přednastavený je střed objektu `transform-origin: center center`. Pokud si například nastavíme levý horní roh, objekt se nám zvětší z něj:

```
   .scale-2 {
  transform: scale(1.5);
  transform-origin: top left;
}
```

Naživo: cdpn.io/e/brBgk.

Podpora v prohlížečích

IE10+. Pro starší prohlížeče budete nejspíš potřebovat detekci vlastnosti (Modernizr) a pak vyrobit alternativní řešení. Základní 2D transformace jdou ve starých IEčkách provést pomocí proprietární vlastnosti `filter`. vrdl.in/3tm8s

Pro převod z CSS3 do filter existuje chytrý konvertor Zoltana Hawryluka. useragentman.com/IETransformsTranslator/

Animace

Transitions: jednoduché animace přechodu

Animace přechodů mezi stavy vlastností elementu.
Zní to možná komplikovaně. Představte si ale tuto situaci:

```
   .box {
  background: green;
}

.box:hover {
  background: blue;
}
```

Nic složitého. Představte si také, že chcete změnu barvy po najetí myši animovat. A právě k tomu slouží transitions, animace přechodů mezi stavy CSS vlastností.

```
   .box {
  background: green;
  transition: 300ms;
}

.box:hover {
  background: blue;
}
```

CSS přechody se typicky spouští po najetí myši, můžete je ale spustit například přidáním třídy javascriptem po kliknutí `.box.clicked { background: blue; }`.
Zkuste si to na cdpn.io/e/hJljB.

V praxi

Takto můžete animovat téměř libovolnou CSS vlastnost včetně pozicování nebo transformací.
Včetně docela divokých hover stavů nad boxy. tympanus.net/Tutorials/OriginalHoverEffects
Plnohodnotný animační nástroj to ovšem není. Pokud chcete mít průběh animace zcela pod kontrolou, podívejte se na CSS3 animace.
Ale pozor, i s Transitions lze hrát velké divadlo! Čtěte dále.

Syntaxe

```
transition:
(_hlidane_vlastnosti_)
_trvani_animace_
(_funkce_prubehu_)
(_zpozdeni_)
(, _dalsi_transition_);
```

TRVÁNÍ ANIMACE

Jediná povinná položka ve zkratce `transition`. Čas můžete udat v sekundách nebo milisekundách. Samostatně tedy s výchozí hodnotou jako `transition-duration: 0s`.

HLÍDANÉ VLASTNOSTI

Z vlastností, které v elementu měníte, si můžete vybrat jen některé. Ostatní se prostě nebudou animovat. Samostatně s výchozí hodnotou jako `transition-property: none`. Příklad:

```
.box {
  background: green;
  transition: margin 300ms;
}

.box:hover {
  background: blue;
  margin-left: 200px;
}
```

Je dobré vědět, že animované přechody nelze aplikovat úplně na všechny CSS vlastnosti. Třeba vlastnost `display` byste animovali marně. Tady je seznam animovatelných na W3.org. vrdl.in/hgx4v

FUNKCE PRŮBĚHU

Samostatně jako `transition-timing-function: ease`. Vybrat si můžete z přednastavených. Více je na W3.org. vrdl.in/p3rh6

Lze si samozřejmě také nadefinovat vlastní. [matthewlein.com/ceaser] (http://matthewlein.com/ceaser].

ZPOŽDĚNÍ

Animaci můžete spustit o chvíli později, než nastane změna vlastností. Samostatně jako `transition-delay: 0s`.

ŘETĚZENÍ ANIMACÍ

Pokud měníte více vlastností, nemusíte je animovat najednou. Docela snadno je poskládáte za sebe. A otevřete tím úplně novou dimenzi možností tvorby animací.

Obě animace v následujícím příkladu trvají 200 milisekund. Druhá, která animuje `background-color`, se spouští s vteřinovým zpožděním po skončení první:

```
  transition: transform 200ms,
 background-color 200ms 1s;
```

Nejlépe je to opět vidět v prohlížeči: cdpn.io/e/vIGAk.

Podpora v prohlížečích

IE10+. Když budete `transition` používat pro dekorativní (nikoliv funkční) animaci, můžete ji udělat výrazně snadněji než Javascriptem a ve starších prohlížečích nebude chybějící animace nikomu vadit.

Pokud děláte funkční animaci (např. indikaci stavu nahrávání), připravte pomocí detekce vlastností alternativní řešení pro starší prohlížeče.

Animations: plnohodnotné animace

Možná se budete divit, ale toto jsou první nativní webové animace vůbec. Překvapující? Všechny existující způsoby animace jsou buď zapouzdřené ve vlastním technologickém kontejneru (Gif, Flash, Silverlight…), nebo animují prostředkem, který pro tento účel nebyl navržen – javascriptem.

Jak se liší od transitions? V animacích (`animation`) máte celou akci daleko víc pod kontrolou a nemusíte se omezovat na CSS vlastnosti, které u animovaného objektu existují před startem animace. Přechody (`transition`) jsou určené jen pro jednoduché animované přechody změny stavu CSS vlastnosti.

Syntaxe

Animaci si nejdřív nadefinujete pomocí at-rule (zavináčové funkce) `@keyframes`. Následně ji můžete kdekoliv zavolat a nastavit si podle libosti.

```
   @keyframes _nazev_animace_ {
    _cas_ { _deklarace_vlastnosti_ }
    _cas_ { _deklarace_vlastnosti_ }
}

#example {
    animation:
        _nazev_animace_
        _cas_trvani_
        _casova_funkce_prubehu_
        _zpozdeni_
        _pocet_opakovani_
        _smer_prubehu_
        _fill_mode_
        (,_dalsi_animace_);
}
```

`animation-name` – NÁZEV ANIMACE

Můžete použít i samostatně jako `animation-name: my_animation`.

`animation-duration` – ČAS TRVÁNÍ

Nastavíte ve vteřinách (`.5s`) nebo v milisekundách (`500ms`). Výchozí hodnota `animation-duration: 0s`.

`animation-timing-function` – ČASOVÁ FUNKCE PRŮBĚHU

Podobně jako u transition lze využít funkce přednastavené nebo definovat vlastní. Samostatný zápis a výchozí hodnota vypadá takto: `animation-timing-function: ease`.

`animation-delay` – ZPOŽDĚNÍ STARTU

Jakou dobu má animace čekat, než se rozběhne? Opět ve vteřinách nebo milisekundách. Přednastavena je samozřejmě nulová hodnota: `animation-delay: 0`.

`animation-iteration-count` – POČET OPAKOVÁNÍ

Lze nastavit číslo nebo nekonečný počet opakování pomocí `infinite`. Přednastavená je hodnota `animation-iteration-count: 1`.

`animation-direction` – SMĚR PRŮBĚHU

Na rozdíl od transitions se v případě opakování animace stav vrátí na výchozí keyframe (0%) a znovu pokračuje k cíli. Pokud chceme, aby na sebe jednotlivá opakování animace navazovala, musíme směr průběhu nastavit na hodnotu `alternate`. Samostatně jako `animation-direction: alternate`.

`animation-fill-mode` – SMĚR PŘEPSÁNÍ VLASTNOSTÍ

Výchozí stav naší animace bude vypadat takto: Před startem animování a po jeho skončení se na animovaný element nijak neaplikují CSS vlastnosti z keyframů animace. Vlastností `animation-fill-mode` toto chování můžeme změnit.

Může nabývat čtyř hodnot:

- `none` — výchozí hodnota.
- `backwards` — i když má element jiné nastavení vlastností, při jeho zobrazení se použijí hodnoty z keyframe 0%.
- `forwards` — po skončení poslední iterace animace zůstane objekt ve stejném stavu, jako je v keyframe 100%, a nevrací se zpět.
- `both` — aplikuje `forwards` i `backwards`.

`animation-play-state` – STAV PŘEHRÁVÁNÍ

Vlastnost, která jako jediná není součástí zkratky animation a je třeba ji používat vždy samostatně. Animaci můžete dočasně zastavit pomocí `animation-play-state: paused`. Co dělá druhá možná hodnota – `running` – je asi zřejmé.

`@keyframes` – KLÍČOVÉ SNÍMKY PRŮBĚHU ANIMACE

Definují začátek (klíčové slovo `from` nebo `0%`), průběh (pomocí procent z průběhu) a konec (`to` nebo `100%`) animace. Přechod mezi jednotlivými

keyframes vypočítá prohlížeč sám. Začátek a konec je potřeba nastavit vždy, počet keyframes mezi nimi není nijak limitovaný.

Podpora v prohlížečích

CSS3 animace nepodporuje například IE9 a starší: caniuse.com/css-animation.

Strategii podpory starších prohlížečů je dobré zvolit podle typu animace.

V případě **vylepšujících animací** (měnší i větší estetické drobnosti v uživatelském rozhraní, u kterých uživateli nevadí, že neproběhnou) není důvod tvořit alternativní řešení.

Pokud **animace nese informaci** (například indikátor stavu načítání uživatelem vloženého souboru), pak je nutné nahradit CSS3 animaci javascriptem nebo detekovat prohlížeče, jež CSS3 animace neovládají a alternativu nabídnout jen jim.

Tipy, triky, odkazy

Pokud se vám animace zdají pomalé nebo v prohlížeči sledujete nepříjemné probliknutí celé stránky, pomáhá vynucené zapnutí **hardwarové akcelerace**. Například: `.animovany_element { transform: translate3d(0, 0, 0); }`

Novější (a standardnější) způsob je pak `will-change: transform` deklarace — dev.opera.com/articles/css-will-change-property.

Běžící kočka od Rachel Nabors. Úžasná animace dokazující, že bez Flashe se pro animace v budoucnu obejdeme. (Jen to bude chvíli trvat.) – codepen.io/rachelnabors/pen/rCost + 24ways.org/2012/flashless-animation.

Pokud potřebujete snadno **nastavit dráhu animovaného objektu**, mrkněte na nástroj Stylie – jeremyckahn.github.io/stylie.

Nástroj Ceaser vám umožní nadefinovat **vlastní časovou funkci průběhu animace** – matthewlein.com/ceaser.

Pokročilý tutoriál k CSS3 animacím – netmagazine.com/tutorials/masterclass-css-animations.

Animace na příkladech

První: blikající box

Na tomto jednoduchém příkladu si vyzkoušejte úplné základy CSS animování. Elementu v nekonečné smyčce měníme opacity po najetí myši.

Nejdřív si pomocí `@keyframes` nadefinujeme průběh animace:

```
  @keyframes my_blink_animation {
  0% { opacity: 1; }
  50% { opacity: 0; }
  100% { opacity: 1; }
}
```

Pod názvem `my_blink_animation` jsme si tedy nadefinovali animaci, která objektu na začátku nastaví plnou poloprůhlednost (`opacity`). V polovině (`50%`) času průběhu animaci pak nastavíme nulovou poloprůhlednost a na konci průběhu opět poloprůhlednost plnou. Bude to blikat, řečeno prostým jazykem.

Animaci pak na element aplikujeme ve chvíli, kdy jej uživatel aktivuje najetím myši, nebo všemi možnými alternativními způsoby:

```
  .example:hover,
.example:focus,
.example:active {
  animation: my_blink_animation 1s infinite;
}
```

Má se provést animace s názvem `my_blink_animation`, trvat přesně jednu vteřinu a mít nekonečný (`infinite`) počet opakování.

Příklad je k vyzkoušení na cdnp.io/e/pKodf. Vyzkoušejte si to; ale prosím vás, v praxi to blikání raději moc nepoužívejte! :-)

Druhý: řetězení animací

Druhý příklad je pokročilejší. Řetězíme v něm dvě různé animace, využíváme směru průběhu a dalších metod, které jsme se naučili v předchozím textu.

Nejprve si obě animace nadefinujeme:

```
  @keyframes rotate {
  to {
    transform: rotate(45deg);
```

```
  }
}

@keyframes pulse {
  to {
    transform: scale(1.2);
  }
}
```

Pokud jste si přečetli text o transformacích, víte, že animace `rotate` otočí element o 45 stupňů doprava a animace `pulse` jej zvětší o 20 %. Všimněte si, že nemusíme deklarovat výchozí stav (`from` nebo `0%`), prostě se vezmou CSS deklarace, které element má ve chvíli, kdy animaci aplikujeme.

Teď animace aplikujeme na uživatelské aktivování elementu. Chceme, aby nejdříve vteřinu proběhla animace `rotate` a pak animace `pulse`, navíc s vteřinovým zpožděním od začátku animace předchozí.

```
   .element:hover,
.element:focus,
.element:active {
  animation:
    rotate 250ms,
    pulse 500ms 1s infinite;
}
```

Příklad je k vyzkoušení na cdpn.io/e/xipAj.

Layout

Multi-column Layout: vícesloupcová sazba textu

Díky tomuto modulu text snadno nasázíte do více sloupců definované šířky podobně jako v novinové sazbě.

Modul sestává z několika vlastností:

```
column-width: _sirka_sloupce_;
column-count: _pocet_sloupcu_;
column-gap: _sirka_odsazeni_mezi_sloupci_;
column-rule: _vlastnosti_cary_mezi_sloupci_;
```

Kromě „novinové“ sazby textu se hodí také na položky seznamu. Třeba náhledy obrázků ve fotogalerii nebo položky e-shopu.

Příklad k vyzkoušení

Definujeme šířku sloupce pomocí `column-width: 15em` a šířku odsazení mezi sloupci v deklaraci `column-gap: 2em`.

V příkladu je i ukázka oddělovací čáry mezi sloupci – `column-rule: 1px dotted #ddd`.

Když si zmenšíte okno prohlížeče a nezbude dost místa pro více sloupců vedle sebe, prohlížeč sám od vícesloupcové sazby upustí.

Příklad vyzkoušejte na cdpn.io/e/ohLgJ.

Podpora v prohlížečích

IE10+. Starší prohlížeče doporučuji řešit tvrdým fallbackem – text se tam prostě jen nezalomí do sloupců.

Flexbox: layout pomocí pružných boxů

Co je flexbox?

Nová cesta pro tvorbu layoutu, zarovnání a distribuci volné plochy.

Flex v češtině znamená *pružný*, *přizpůsobivý*. Flexboxy jsou tedy *pružné* elementy layoutu. Jednou z hlavních předností flexboxu je totiž schopnost vyplňovat zbylý prostor bez nutnosti přepočítávání javascriptem.

ZÁKLADY V JEDNODUCHÉM PŘÍKLADU

Představme si triviální třísloupcový layout:

```
<div class="container">
  <p class="mandatory-1">One</p>
  <p class="content">Two<br/>…<br/>…</p>
  <p class="mandatory-2">Three </p>
</div>
```

HTML je jednoduché. O to přísnější máme požadavky na design. A víte co? Ukážeme si rovnou, jak je splnit pomocí flexboxu.

1. **Všechny sloupce stejně vysoké.** Ano, i v případech kdy má ten jeden delší obsah než zbylé dva. To je to nejjednodušší. Stačí z rodiče udělat kontejner flexboxu – `.container { display: flex; }`.
2. **Chceme pětinovou mřížku**. První a třetí sloupec má zabírat jednu pětinu – `.mandatory-1, .mandatory-2 { flex: 1; }`. A druhý pak tři pětiny – `.content { flex: 3 }`. Všimli jste si, že jsme nemuseli počítat s procenty? A že bychom nemuseli procenta přepočítávat, když bychom přidali další sloupec?
3. **Na menších rozlišeních chceme změnit pořadí elementů.** Prostě jen do media query napíšeme `.content { order: -1; }`, a sloupec s obsahem se přesune na první místo. Bomba pro responzivní design, že?

Příklad si utíkejte vyzkoušet naživo na CodePen. cdpn.io/e/LhGuD

Je to hezké, že? Ale skeptik by zamručel, že se CSSko konečně naučilo to, co jsme uměli pomocí „tabulkového layoutu“ v roce 2001. Jenže pravdu by měl jen z velmi malé části. Flexbox toho totiž umí daleko (*daleko!*) více než tabulky.

PROČ POTŘEBUJEME FLEXBOX?

Dovolte nejprve otázku. Jak dnes v CSS děláme layouty?

Floaty, inline-blockem, absolutním pozicováním nebo přes `display: table`. A víte, co mají všechny společného? Ani jeden nebyl vymyšlen pro tvorbu dnešních layoutů. Ano, flexbox je první layoutovací nástroj v CSS. „Po dvaceti letech," prohodil by pod fousy náš bručoun.

Floaty, tabulky a jejich parta *Layoutovacích technik nad hrobem* samozřejmě ještě pár měsíců až let poslouží. Každý člen skupiny má ovšem pro tvorbu layoutů větší či menší nevýhody. Ty vyplývají už ze specifikací, které s dnešními layouty příliš nepočítaly.

HLAVNĚ LAYOUTY KOMPONENT

Je dobré zmínit, že flexbox je určený pro layout komponent uvnitř stránek. Tedy navigací, formulářů, stránkovacích komponent atd. atd.

Pro celostránkové layouty se více hodí CSS3 Grid Layout. Ten má ovšem zatím jen malou podporu v prohlížečích. Užití flexboxu pro celostránkové layouty je samozřejmě možné. Jen se na velmi pomalých zařízeních nebo internetových připojeních nebude vykreslovat optimálně. Píše o tom třeba Jake Archibald. vrdl.in/zuscj

Šup na základní pojmy — flex kontejner a flex položka, hlavní a příčná osa

Flexbox tvoří nerozlučná dvojice dvou typů elementů – flex kontejner a flex položka. Flex položkou se stává každý přímý potomek kontejneru.

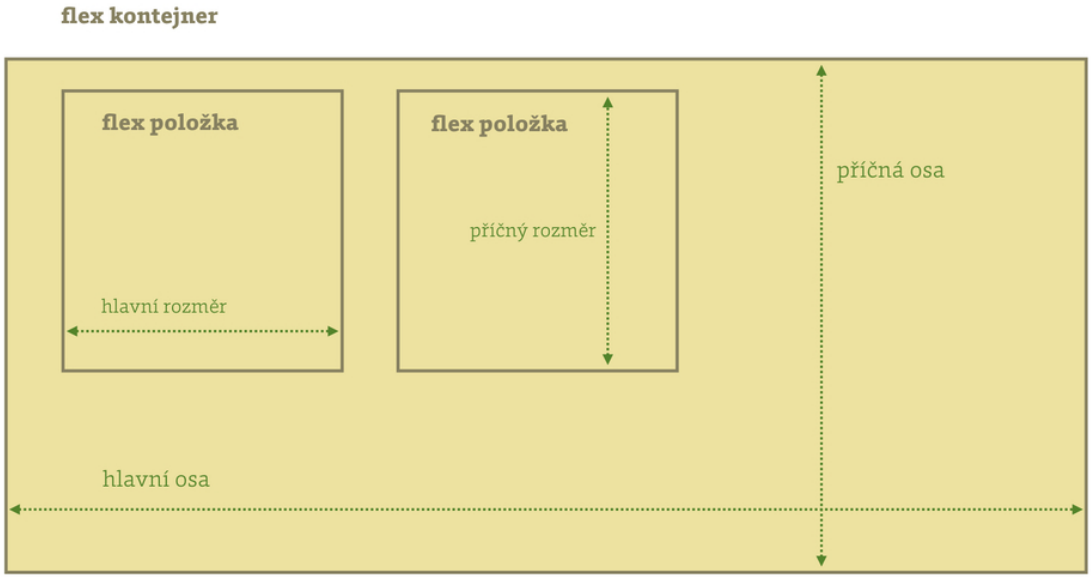
```
   <ul class="flex-container">
  <li>…</li>
  <li>…</li>
</ul>
```

Flexbox nadefinujeme snadno jen pomocí flex kontejneru:

```
   .flex-container {
  display: flex;
}
```

Všechny `<li>` se tady stávají flex položkami.

Kromě flex kontejnerů a položek nás v dalším textu budou zajímat ještě osy. Ukažme si to na zjednodušeném schématu:



- flex kontejner – rodičovský element
- flex položka – všichni přímí potomci flex kontejneru
- hlavní osa – výchozí je horizontální, ale lze změnit
- příčná osa – vždy příčná k hlavní, takže ve výchozí podobě svislá
- hlavní rozměr – výchozí je šířka, ale řídí se nastavením hlavní osy
- příčný rozměr – výchozí je výška

Flexbox: vlastnosti kontejneru

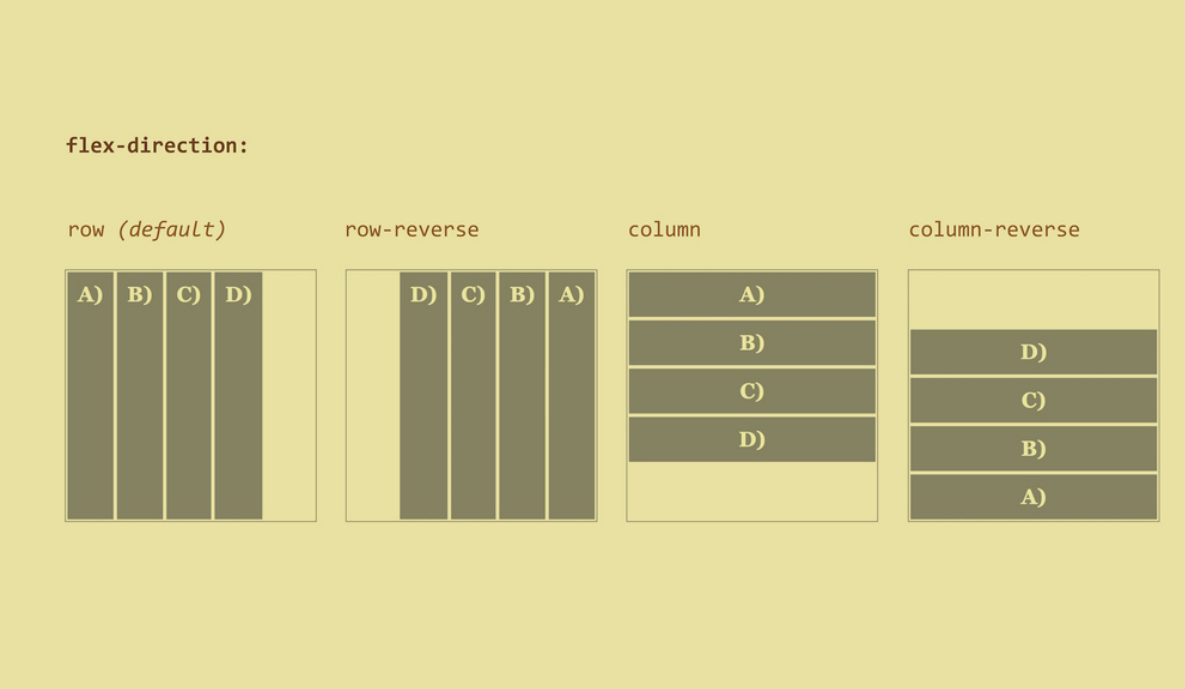
Blokový nebo řádkový?

Kromě `display: flex` můžete kontejner flexboxu definovat jako řádkový – `display: inline-flex`. V obou případech se ze všech přímých potomků stávají položky flexboxu.

`flex-direction` – směr vyskládání položek

Nastaví směr hlavní osy flexboxu.

```
flex-direction:
  row | row-reverse |
  column | column-reverse
```



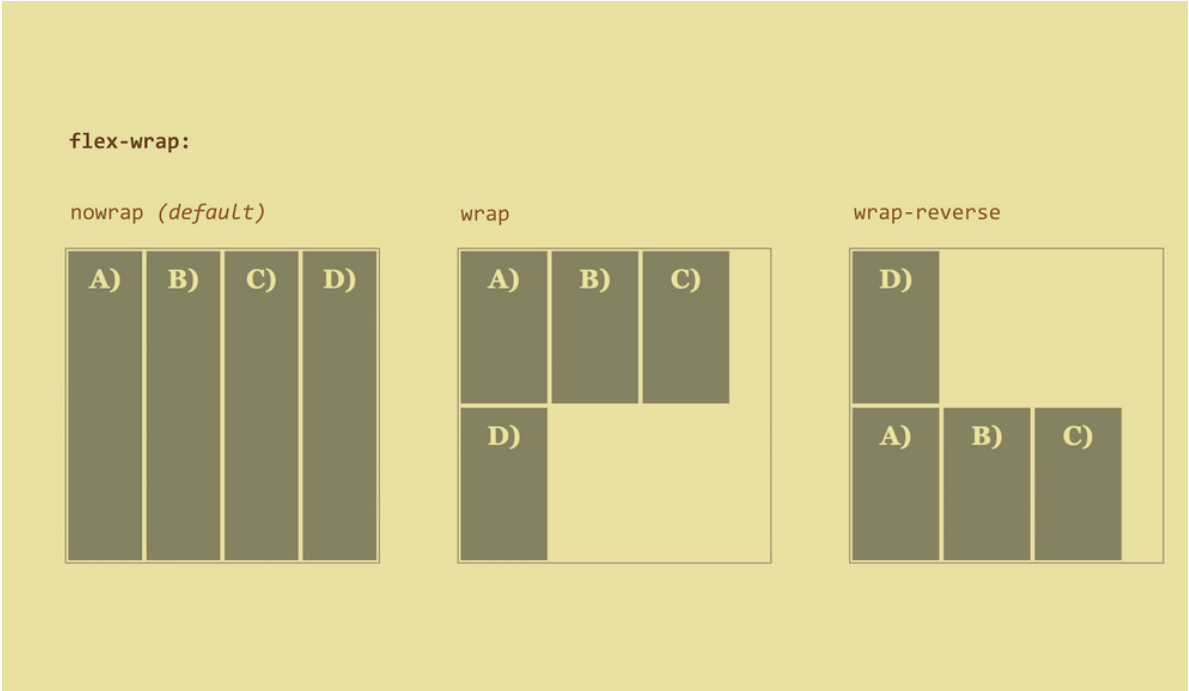
Výchozí (`row`) hodnota vyskládá flex položky do řádky. Pokud chcete dělat layout do svislého směru, použijte hodnotu `column`.

Pořadí položek se v těchto případech bere z pořadí v kódu. Pokud chcete pořadí otočit, prostě zvolte hodnoty `row-reverse` nebo `column-reverse`. To má vliv jen na vizuální vykreslení, nikoliv např. na pořadí vykreslování nebo procházení při navigaci klávesou Tab. Pozorní si asi všimli, že vlastnost lze použít i pro změnu řazení seznamů.

Živé demo: cdpn.io/e/pbarBw.

`flex-wrap` – zalamování položek

```
flex-wrap:
nowrap | wrap | wrap-reverse
```



Výchozí `nowrap` říká, že elementy budou vždy na hlavní ose vedle sebe (nebo pod sebou v případě, že použijete `flex-direction: column`).

Alternativně `wrap`. Pak se flex položky zalomí na další řádku ve chvíli, kdy se jejich obsah zvětší natolik, že se nevejdou do jedné. Poslední flex položka na prvním řádku skočí dolů a zařadí se pod první položku.

`wrap-reverse` zalamuje naopak. Poslední položka řádku skočí nahoru a zařadí se nad první položku.

Živé demo: cdpn.io/e/mERZxB.

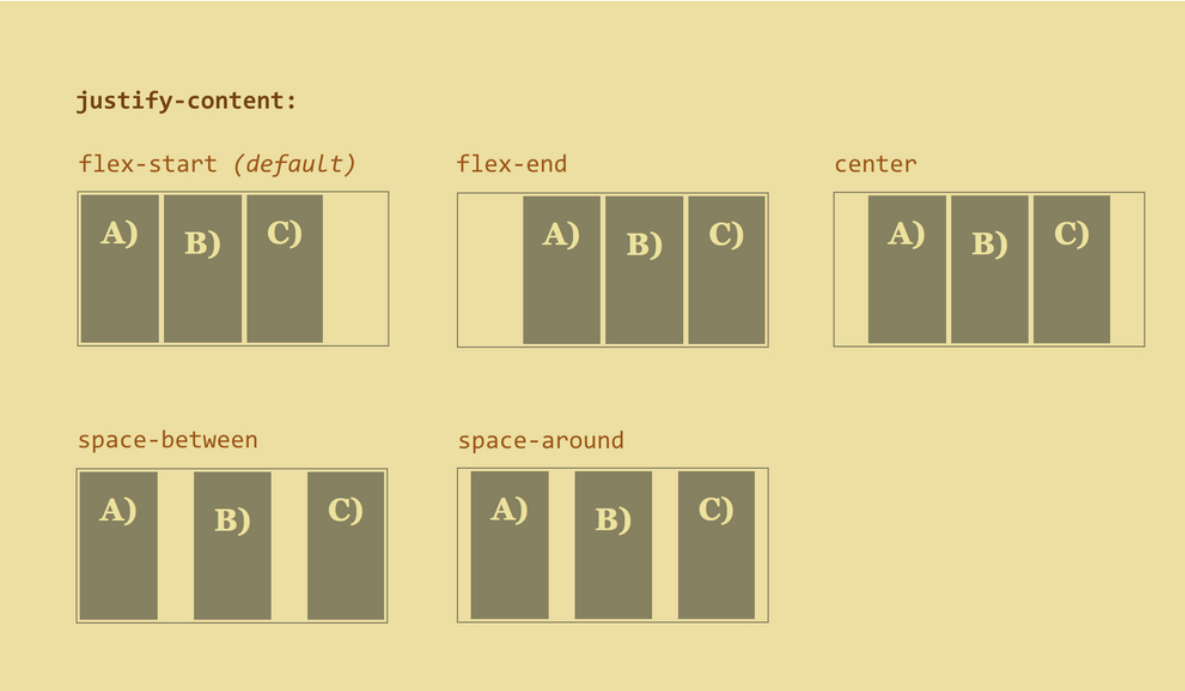
`flex-flow`, zkratka pro `flex-direction` a `flex-wrap`

Nejlépe si to ukážeme na příkladech:

- `flex-flow: row` – výchozí hodnota. Položky se vyskládají do řádku a nezalomí se.
- `flex-flow: column wrap` – položky se vyskládají do sloupce a zalamují se.

`justify-content` – zarovnání položek na hlavní ose

```
justify-content:
flex-start | flex-end | center |
space-between | space-around
```

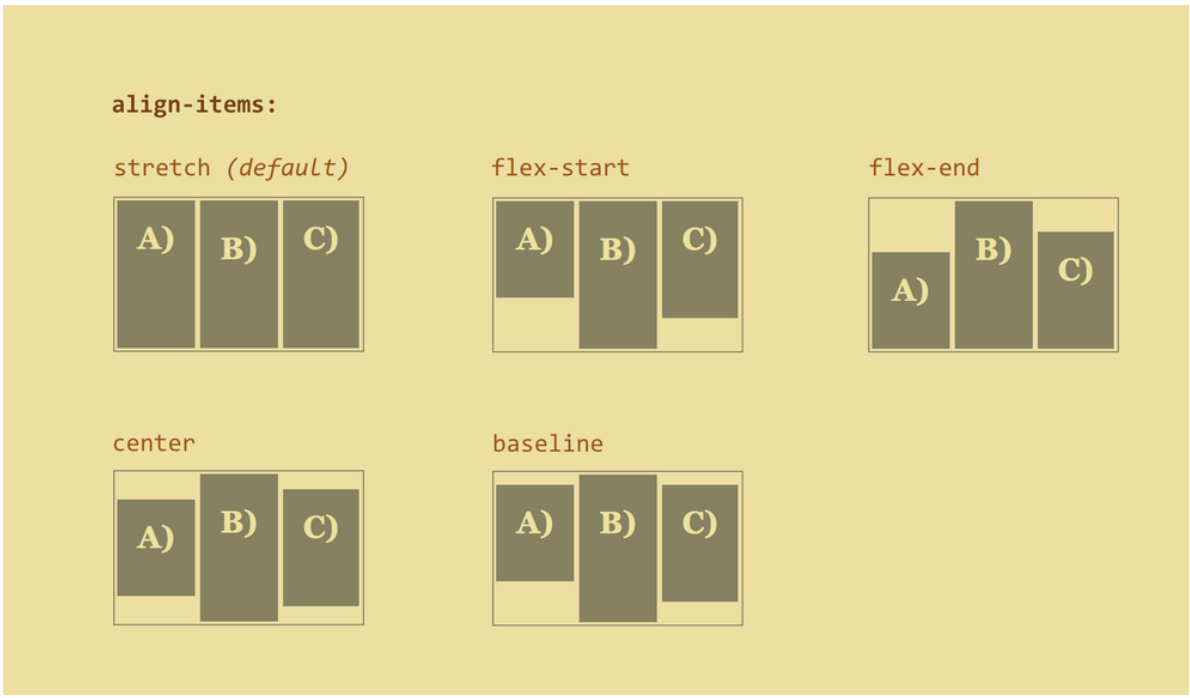


Vlastnost `justify-content` aplikujeme na flex kontejner. Říká, jak budou flex položky zarovnány po jeho hlavní ose. Výchozí hodnota je `flex-start`, tedy zarovnání k začátku hlavní osy.

Živé demo: cdpn.io/e/doGjaZ.

`align-items` – zarovnání položek na příčné ose

```
align-items:
stretch | flex-start |
flex-end | center | baseline
```



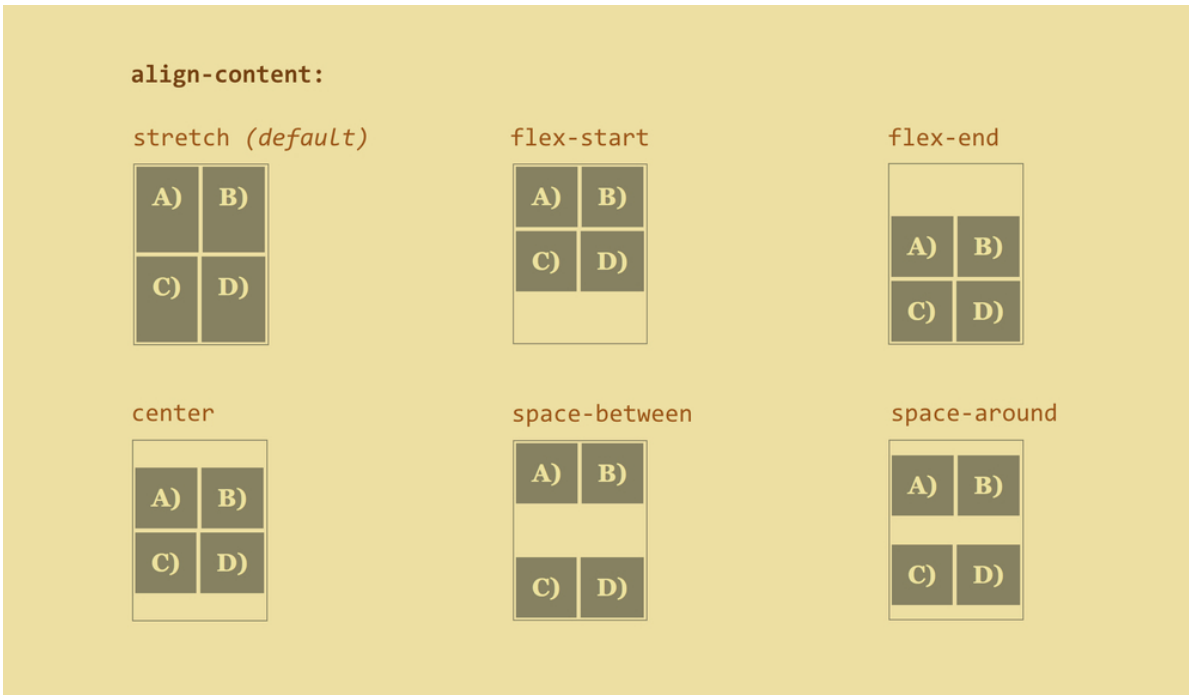
Vlastnost `align-items` lze opět aplikovat na kontejner flexboxu. Výchozí hodnota je `stretch`, tedy roztažení na celou délku příčné osy.

Pozor, hodnota `stretch` nefunguje, pokud mají položky nastavený rozměr pro příčnou osu, tedy ve výchozím stavu hodnotu vlastnosti `height`.

Živé demo: cdpn.io/e/RNmvmr.

`align-content` – zarovnání na hlavní ose víceřádkového kontejneru

```
align-content:
stretch | flex-start | flex-end |
center | space-between | space-around
```



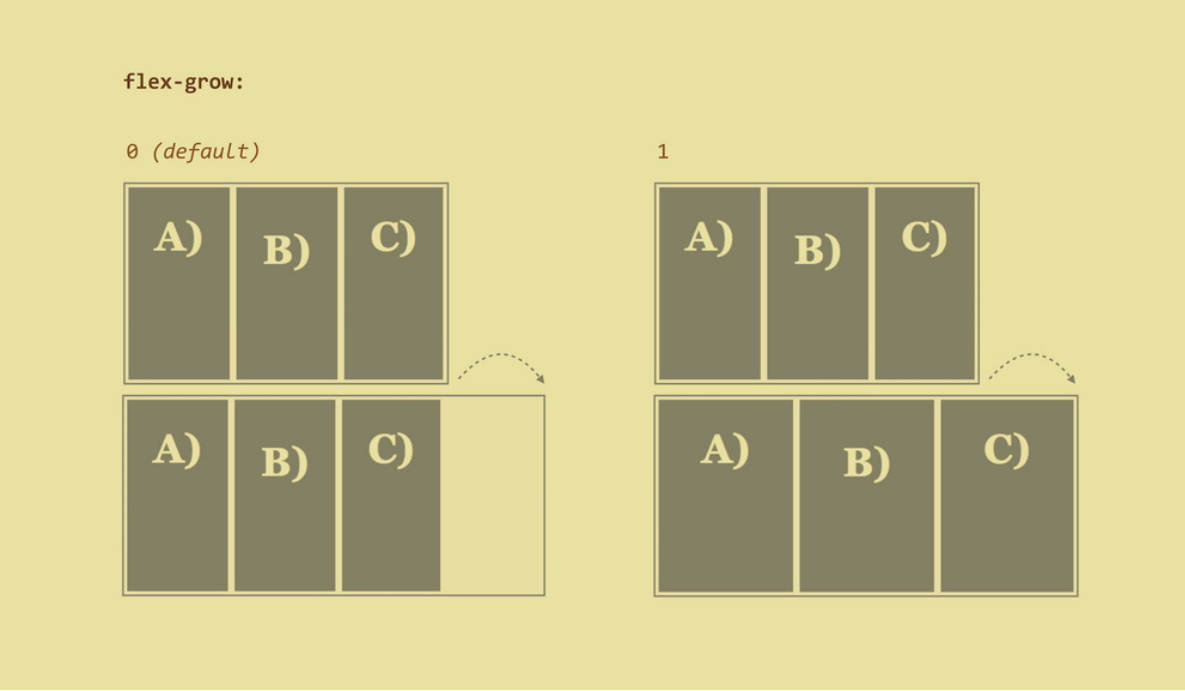
Ještě jedna zarovnávací vlastnost. Tentokrát se vztahuje jen na flex kontejnery, jejichž položky se rozpadnou na více řádků.

Živé demo: cdpn.io/e/oXbMRo.

Flexbox: vlastnosti položky

`flex-grow` – možnost zvětšování

Jak moc může položka růst relativně k dalším položkám, pokud je k dispozici volné místo – například když uživatel zvětší okno prohlížeče.



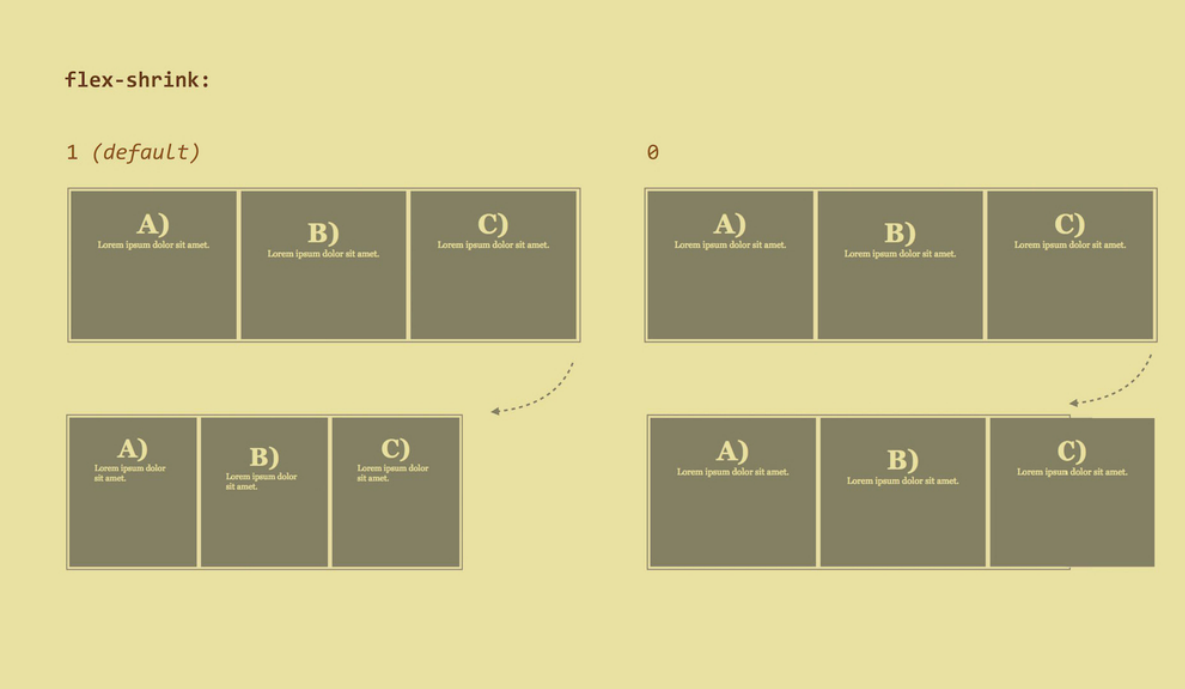
Možné hodnoty:

- `0` (výchozí) znamená, že položky nijak nerostou.
- Celá kladná čísla. Položky si rozdělují podíly z nově získaného místa nad rámec výchozí šířky.

Živé demo: cdpn.io/e/GqrVzL.

`flex-shrink` – možnosti smršťování

Jakým podílem vzhledem k ostatním položkám se může definovaná položka zmenšovat, pokud v rodičovském kontejneru místo ubylo – když uživatel zmenšil okno nebo třeba přibyla nová položka.



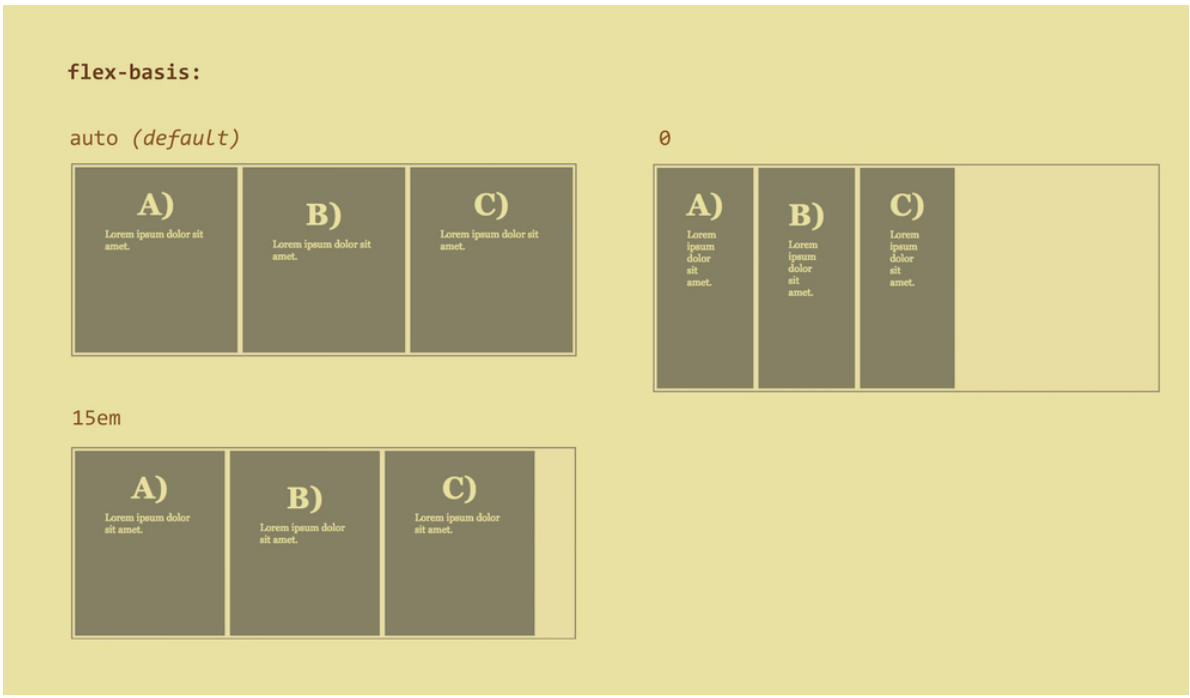
Možné hodnoty:

- `1` (výchozí) – položky si z vlastní šířky ubírají rovnoměrně.
- Celá kladná čísla.

Živé demo: cdpn.io/e/PzWMvM.

`flex-basis` – výchozí rozměr položky

Výchozí šířka položky. Alternativně výška, pokud je `flex-direction: column`.



- `auto` (výchozí) – rozměr určuje obsah podobně jako u `width: auto`. Distribuce volného místa pomocí `flex-grow` a `flex-basis` se pak bude týkat jen místa, které položky okupují nad rámec svého obsahu – tzv. relativní model pružnosti.
- `0` – nehledí se na rozměr obsahu. Distribuce volného místa pomocí `flex-grow` a `flex-basis` se bude týkat celé šířky položky – absolutní model pružnosti.
- Jakýkoliv CSS rozměr, např. `100px`, `15em` nebo `50%`.

Živé demo: cdpn.io/e/oLZvgQ.

`flex` – celková pružnost položky

Zkratka pro všechny vlastnosti definující pružnost flex položky – `flex-grow`, `flex-shrink` a `flex-basis`. Nastaví výchozí velikost elementu a způsob, jakým se smí zvětšovat a zmenšovat.

Je dobré vědět, že autoři specifikace doporučují upřednostňovat zkratku `flex` proti konkrétním vlastnostem, které zastupuje. Důvodem je, že zkratka umí inteligentně nastavovat výchozí hodnoty.

```
flex:
<flex-grow> <flex-shrink> <flex-basis>
```

Výchozí hodnota je:

```
flex: 0 1 auto
```

- `flex-grow: 0` – nebude se nijak roztahovat do volného místa.
- `flex-shrink: 1` – smršťovat se bude stejně jako ostatní položky.
- `flex-basis: auto` – zabere prostor, který jí určí vlastní obsah.

Pokud chcete například nastavit, aby vaše položky zabíraly minimálně `150px` a v případě dostupnosti volného prostoru se rovnoměrně zvětšily a v případě zmenšení prostoru zase rovnoměrně smrštily, uděláte to takto:

```
flex: 1 1 150px
```

Myslím, že častěji se ale budou hodit přednastavené „inteligentní“ hodnoty:

	roste?	smršťuje se?	výchozí velikost
`flex: initial`		✔	`auto`
`flex: auto`	✔	✔	`auto`
`flex: none`			`auto`

- `flex: auto` Odpovídá `flex: 1 1 auto` a dotčené položky se stanou plně pružnými s výchozím rozměrem podle svého obsahu. Asi nejčastější případ.
- `flex: none` Odpovídá `flex: 0 0 auto` a zcela ruší pružnost položky. Druhá nejčastější situace.
- `flex: initial` Zpětné nastavení výchozí hodnoty, tedy `flex: 0 1 auto`. Položky se tak s ubývajícím místem zmenšují, ale nezvětšují nad velikost svého obsahu.
- `flex: <kladné-číslo>` U jednočíselného zápisu pozor! `flex: 1` znamená `flex: 1 1 0`, takže se vám změní výchozí velikost položky a model pružnosti, jak jsme zmiňovali u vlastnosti flex-basis.

Je také dobré vědět, že se flex položky nikdy nezmenší pod minimální šířku obsahu. Ta je dána šířkou nejdelšího slova nebo vnitřního elementu fixní šířky – třeba obrázku. Lze to změnit nastavením `min-width` nebo `min-height` na nějakou nízkou hodnotu.

`order` – změna pořadí prvků

Pořadí flex položky standardně odpovídá zdrojovému kódu, ale to můžeme změnit pomocí vlastnosti order.

Změna pořadí má vliv na vizuální pozici elementu a na pořadí jeho vykreslení prohlížečem. Nemá ale vliv například na pořadí čtení dokumentu čtečkami nebo na pořadí navigace pomocí klávesy tab.

Výchozí hodnota je `0`, což znamená „dodržujeme pořadí ze zdrojového HTML“.

Tímto zápisem pak třeba třetí položku předřadíme první:

```
.flex-item-third {
  order: -1;
}
```

Nezapomeňte, že order nelze použít na jiné elementy ve stránce než přímé potomky flex kontejneru.

Teď je na řadě další z radostí, kterou přináší flexbox. Konečně v CSS snadno zarovnáme prvky layoutu vodorovně, ale i svisle.

Živé demo: cdpn.io/e/JoqxJe.

`margin` – zarovnání položek na hlavní ose pro jednotlivou položku

`margin: auto` funguje podobně jako u blokových elementů. Když se počítají rozměry flex položek, nijak se tato hodnota nezohledňuje. Zbývající volné místo se pak spravedlivě rozdělí mezi všechny takto nastavené vnější okraje.

Díky tomu můžete flex položce nastavit `margin-left: auto` a tím zajistit, aby vnější okraj vyplnil všechno volné místo nalevo od ní a ona se tak zarovnala zcela vpravo. Využitelné to je namísto `float` vlastností.

`align-self` – zarovnání položky na příčné ose

```
align-self:
auto | flex-start | flex-end |
center | baseline | stretch
```

Tato vlastnost se aplikuje na jednotlivé položky, a tak se hodí pro vytvoření výjimky ze zarovnání. Výchozí hodnota je `auto`.

Živé demo: cdpn.io/e/OXWKwe.

POZNÁMKA: BASELINE ZAROVNÁNÍ

Doporučuji všimnout si velmi praktického zarovnání na účaří prvního řádku – `baseline`. K horní hraně flex kontejneru se přilepí položka s největší vzdáleností mezi baseline a horní hranou boxu. Vidět je na předchozím obrázku nebo na cdpn.io/e/QwobXz. Všimněte si, že flexbox nerozhodí ani nastavení horního paddingu v pixelech.

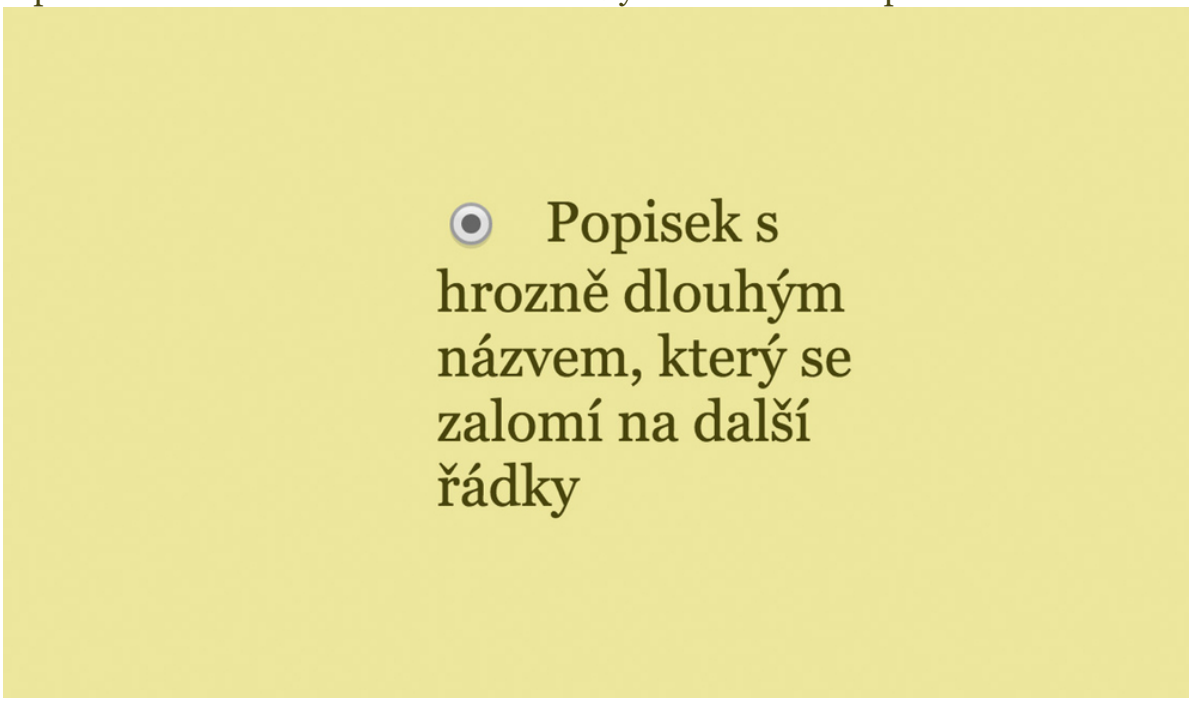
Flexbox: praktické příklady

První příklad: základy pružnosti

Vezměme toto HTML:

```
  <label>
 <input type="radio"> Popisek s hrozně dlouhým
   názvem, který se zalomí na další řádky
</label>
```

V prohlížeči to bude nezarovnané. Za to bychom od klienta pochvalu nedostali:



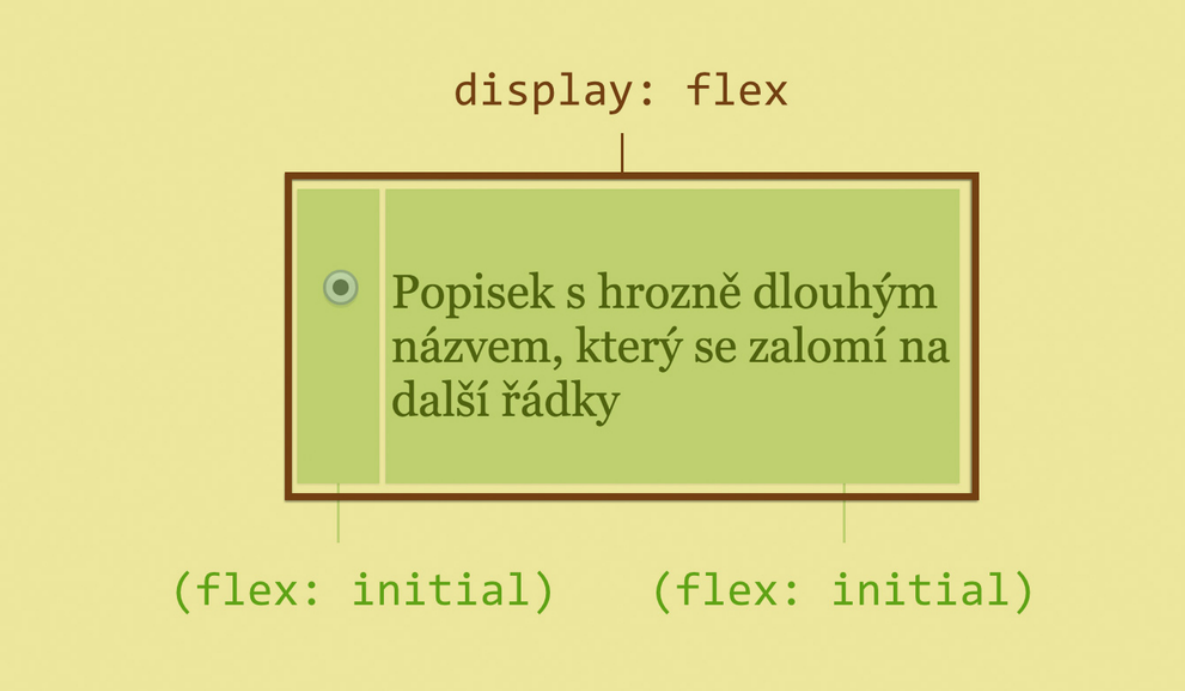
Samozřejmě chceme, aby text nepřetékal pod přepínač (radio button). Zvládli bychom to jistě i bez flexboxu, ten z toho ale dělá velmi jednoduchou záležitost. Stačí nám totiž jedna deklarace:

```
label { display: flex }
```

Tím jsme z `<label>` udělali kontejner flexboxu a z přímých potomků jeho položky.

Asi je dobře vidět, že za položku flexboxu se považuje i prázdný textový uzel, tedy text nezanořený do tagu. V našem případě onen „Popisek s hrozně dlouhým názvem…”.

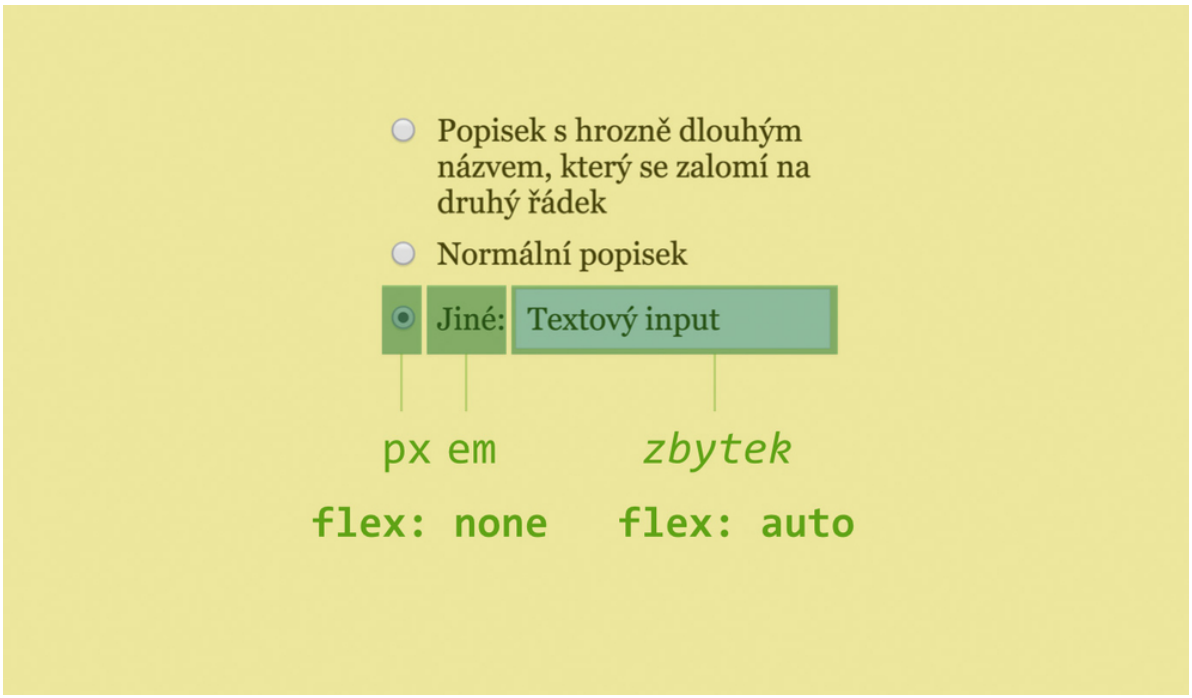
Pojďme si vzniklou situaci ještě ukázat na obrázku:



Technicky vzato má totiž každá položka vlastnost `flex` a její výchozí nastavení na `initial`. Jak si ukážeme v referenční příručce, `flex` je zkratka pro další vlastnosti. Hodnota `initial` znamená, že se položka umí s ubývajícím prostorem smršťovat, ale neroztáhne se, ani když dostane dostatek volného místa.

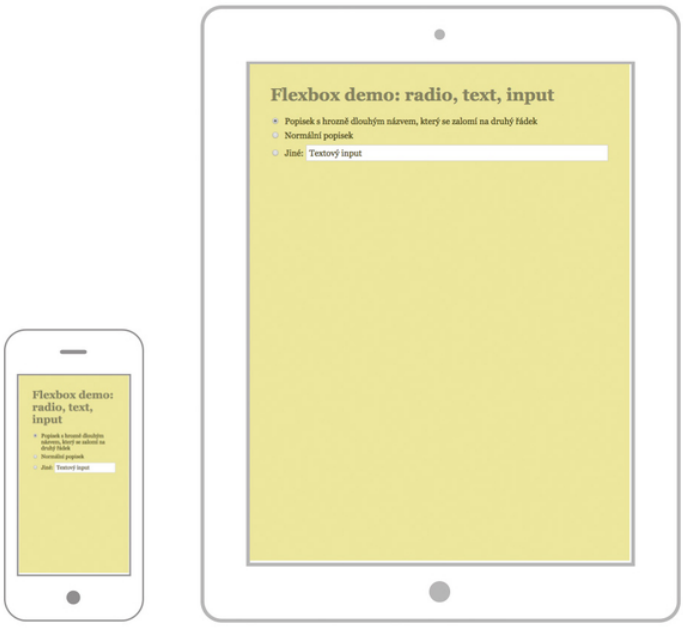
Příklad si můžete vyzkoušet na CodePenu http://cdpn.io/e/raqXZX.

Příklad druhý: kombinování jednotek



Náš formulář rozšíříme. Podívejte se hlavně na poslední řádek. V tomto případě kombinujeme různé jednotky.

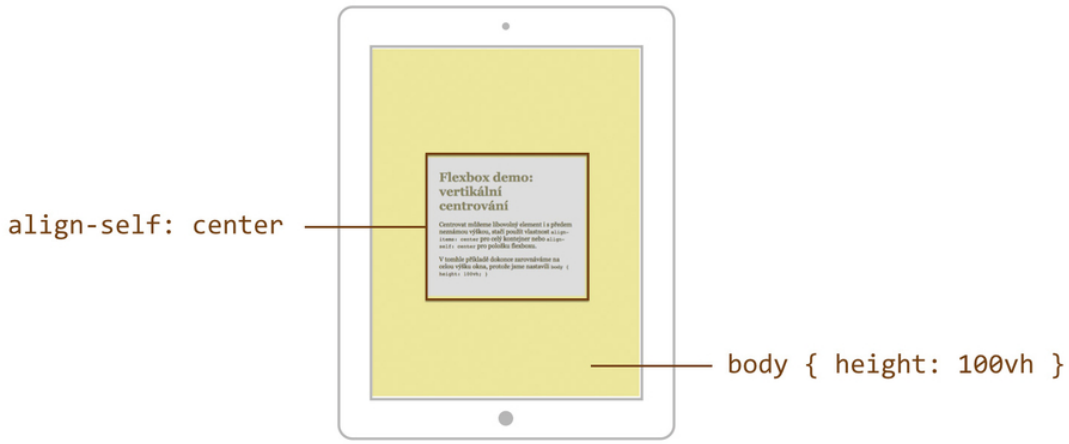
Šířku přepínače jako součásti rozhraní prohlížeče chceme nadefinovat v pixelech. Jenže šířka textového popisku závisí spíše na velikosti písma, proto definujeme v `em`. Pak bychom rádi, aby textový input zabral zbytek plochy. A nechceme přepínač a textový popisek zvětšovat či zmenšovat. S flexboxem to půjde snadno. Prohlížeč si s kombinacemi jednotek poradí sám:



Všimněte si, že jsme přepínači a textu nastavili `flex: none`. Nechceme totiž aby se smršťovaly nebo roztahovaly. Prostě drží šířku za každou cenu. A naopak – textovému formulářovému políčku jsme pomocí `flex: auto` přikázali, aby se vždy smršťovalo a roztahovalo co mu šířka rodiče dovolí.

Příklad si opět můžete vyzkoušet na CodePenu http://cdpn.io/e/jEJbmg.

Příklad třetí: svislé centrování boxu neznámé výšky



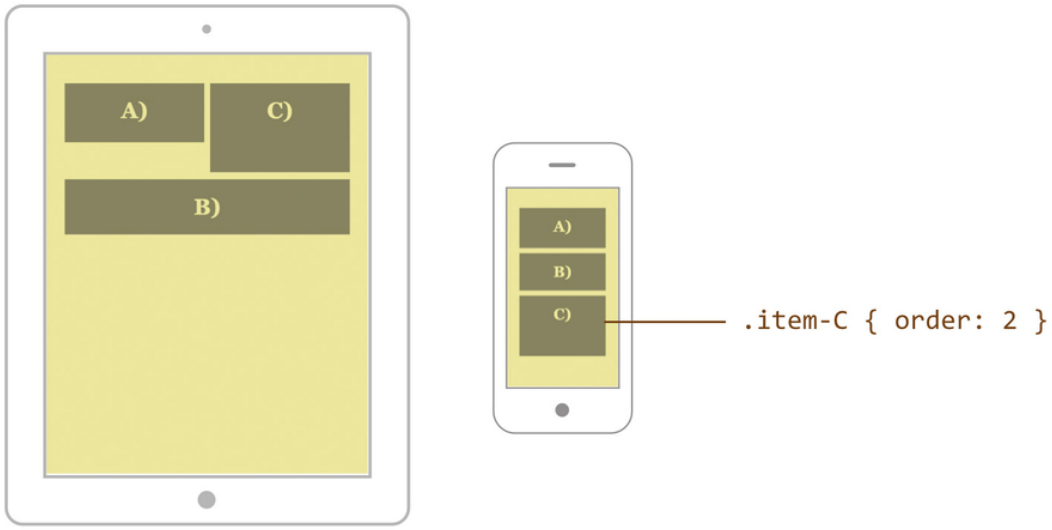
Také jste se s tímto úkolem dřív trápili? S flexboxem už nebudete! Z `<body>` uděláme flex kontejner a roztáhneme jej na 100 % výšky okna – pomocí nové jednotky `vh`. Pak už jen stačí flex položce pomocí `align-self` přikázat, ať se centruje.

Centrování v obou směrech je s flexboxem bezproblémové. Kromě `align-self` se do následující referenční příručky podívejte i na vlastnosti `justify-content` a `align-items`.

CodePen ukázka je na http://cdpn.io/e/zxydom.

Příklad čtvrtý: změna vizuálního pořadí položek

Potřebujete v HTML pořadí položek jedním způsobem, ale zobrazovat je zase jiným způsobem? I tady flexbox pomůže. Zapamatujte si vlastnost `order`, která slouží pro změnu pořadí flex položek.



V kódu máme pořadí: A) C) B). Na malých displejích ale chceme vizuální pořadí změnit. C) bude následovat až po B). Uděláme to takto, protože vlastnost pořadí se počítá od nuly:

```
.item-C { order: 2 }
```

V prohlížeči vyzkoušíte na adrese http://cdpn.io/e/JoqxJe.

Změna vizuálního pořadí se hodí i třeba pro řazení od konce abecedy. Do příručky se podívejte na vlastnost flex-direction.

Příklad pátý: navigace s neznámým počtem položek

Představte si horizontální navigaci, u které předem neznáte počet položek. Ale nepředstavujte si ji prosím moc dlouho, abyste z toho nedostali nějakou ošklivou vyrážku.

Toto je její HTML:

```
<div class="nav">
<ul>
  <li><a href="/produkty">Produkty</a></li>
  <li><a href="/cenik">Ceník</a></li>
  <li><a href="/o-nas">O nás</a></li>
  <li><a href="/kontakty">Kontakty</a></li>
  <!-- Další položky přidané
    v redakčním systému… -->
```

```
  </ul>
</div>
```

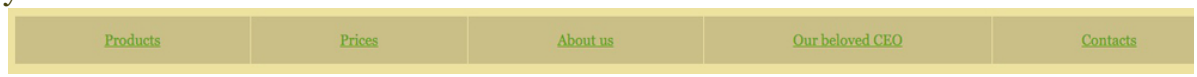
A takhle chceme, aby vypadala:

Bez flexboxu problém dokážeme vyřešit pomocí `display: table`. Ale trápily by nás nevýhody v podobě nutnosti přidání elementu s `display: table-row` a třeba nemožnost absolutního pozicování uvnitř „buněk tabulky".

O flexboxu už toho víme dost, a tak nás řešení asi nepřekvapí:

```
   .nav ul {
  display: flex;
}

.nav li {
  flex: auto;
}
```

Když ovšem přidáme další – nezvykle dlouhou – položku, vezme si daleko více prostoru. Je to kvůli relativní distribuci volného prostoru flex-basis: auto, která je skrytá ve zkratkové deklaraci `flex: auto`.



Můžeme ale chtít, aby všechny položky byly stejně široké, že ano? Pak změníme model pružnosti na absolutní tím, že nastavíme `flex-basis: 0`.

Co je absolutní model pružnosti? Podíl pro rozšiřování položky do volného místa se počítá z její celkové šířky. U relativního modelu se počítá jen z prostoru, který zbývá mezi šířkou obsahu a celkovou šířkou.

Takhle nám to vyhovuje. V prohlížeči vyzkoušíte na adrese http://cdpn.io/e/NPevjg.

Flexbox: podpora v prohlížečích

Není špatná. V době psaní článku je to v ČR 92–95 % a s pomocí rozumných fallbacků pro starší prohlížeče není důvod jej nepoužít hned.

Tři syntaxe v moderních prohlížečích

Raději rovnou píšu, že plnou podporu v moderních prohlížečích vám zajistí dobře nakonfigurovaný Autoprefixer, o kterém byla řeč v úvodu ebooku. Dále proto čtěte, jen pokud vás zajímají detaily.

V moderních prohlížečích má flexbox tři verze syntaxe.

- **Finální syntaxe.** `display: flex` a další vlastnosti, tak jak ukazujeme v ebooku. Podporují poslední verze všech prohlížečů, včetně IE11 nebo Opery Mini.
- **Mezisyntaxe (tweener) z roku 2012.** `display: flexbox`. Dnes už vyžaduje snad jen IE10.
- **Stará (legacy) syntaxe z roku 2009.** Kdekoliv uvidíte `display: box` nebo lépe `-webkit-display: box`. Webkit ve verzi 20 a starší, tzn. třeba iOS6 nebo starší Androidy.

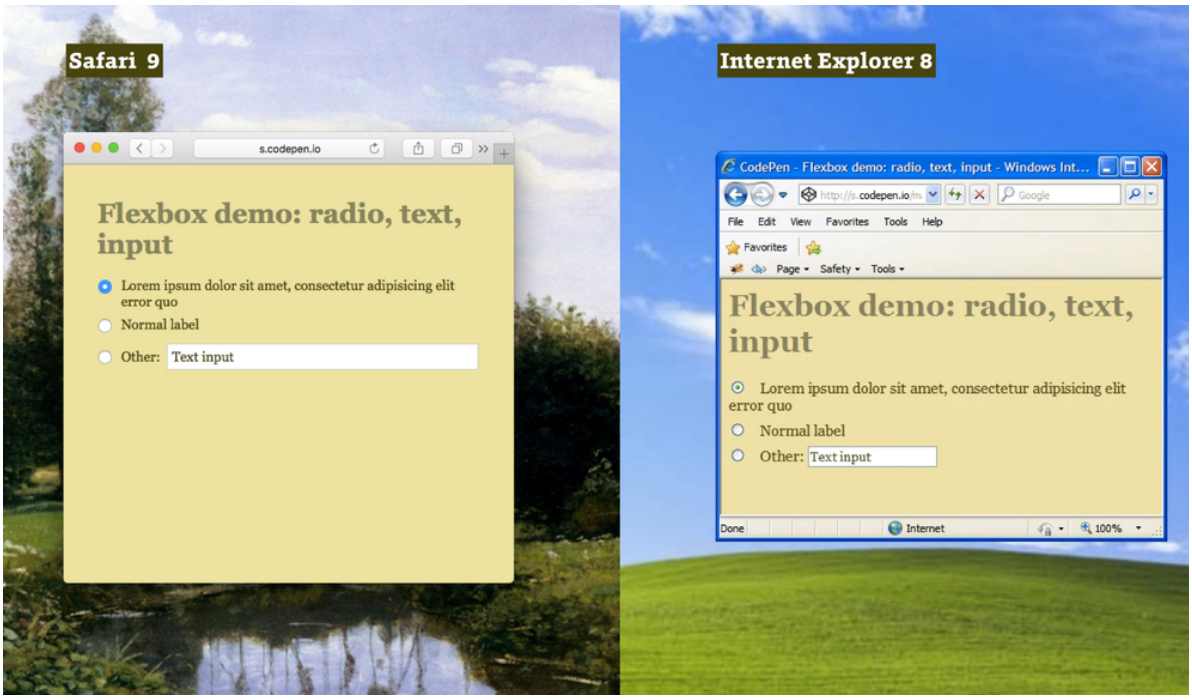
Jak v prohlížečích bez podpory?

Týká se jen IE9 a starších. Raději se podívejte do statistik vašeho projektu, kolik uživatelů vlastně mají.

NULOVÝ FALLBACK – PODPORU VŮBEC NEŘEŠÍME

Je ale dobré si ujasnit a v praxi vyzkoušet, co to znamená „neřešit podporu flexboxu ve starších prohlížečích“.

Vidět je to na následujícím obrázku. Pro layout formuláře se tam docela hojně využívá flexbox. V Safari se zobrazuje, tak jako bylo plánováno. V IE8 nevyplní textový input celou zbývající plochu a popisek „Jiné“ se nebude zvětšovat se zvětšující se velikostí písma:



Je to příklad, který už známe z cdpn.io/e/jEJbmg.

Formulář tedy zůstává použitelný. Jen některé prvky nebudou uživatelsky tak přívětivé. No jistě. Nejen zaoblené rohy, ale i layout můžeme brát jako vylepšení uživatelského prožitku:

DETEKCE VLASTNOSTÍ A ALTERNATIVNÍ ŘEŠENÍ

Nulový fallback ovšem nemůžeme použít u příkladu s centrováním boxu doprostřed stránky. Pokud by v konkrétní aplikaci bylo potřeba dodržet alespoň obdobný vzhled, museli bychom si vypomoci alternativním řešením v CSS nebo případně Javascriptu.

Jak už bylo zmíněno v úvodní kapitole, pomohl by nám detekční skript Modernizr a jeho třída `.no-flexbox`.

Principiálně by řešení vypadalo takto:

```
   .vertical-centered {
  /* Flexbox centrování */
}

.no-brainer .vertical-centered {
  /* Centrování bez flexboxu */
}
```

Takto lze udělat alternativní layout například pomocí `display: table|table-row|table-cell`.

DEFINOVANÝ FALLBACK

Flexbox je jen nová hodnota vlastnosti display. S tím je možné pracovat hlavně u typů fallbacků, kdy chceme, aby se i původně `inline` nebo `inline-block` elementy naskládaly pod sebe. V prohlížečích bez podpory flexboxu se tedy naskládají pod sebe, často podobně jako na mobilních zařízeních.

```
   .box {
  display: block;
  display: flexbox;
}
```

Snadno se například dělá fallback pomocí vlastnosti `float`, protože ta se na flex položky nedá aplikovat:

```
   .container {
   display: flexbox;
}

.box {
  display: block;
  float: left;
}
```

FLEXBUGS – NEJČASTĚJŠÍ CHYBY PROHLÍŽEČŮ

Jak už jste si jistě všimli, flexbox je dost složitý standard a to – spolu s jeho relativní čerstvostí – způsobuje řadu drobných chyb v prohlížečích.

Kromě standardního hledání na Google nebo StackOverflow doporučuji projekt **flexbugs**. Phillip Walton a komunita tam sbírá, dokumentuje a hledá řešení nejčastějších chyb. https://github.com/philipwalton/flexbugs

Co je dobré si o flexboxu zapsat za uši

1. `flex` je nová hodnota vlastnosti `display`

Na rodičovský kontejner ani jeho potomky proto nelze aplikovat vlastnosti související se zobrazovacím režimem `display: block`, `inline` nebo `inline-block`. Typický příklad jsou vlastnosti `float`, `clear` nebo `vertical-align`. Přesněji řečeno – uvedené vlastnosti aplikovat lze, jen nebudou mít v prohlížečích s podporou flexboxu žádný účinek. Využitelné to ale je při tvorbě fallbacků pro starší prohlížeče.

2. Flex položky neslučují vnější okraje

S předchozím bodem souvisí i to, že na rozdíl od blokových elementů se u sousedících flex položek neslučují vnější okraje (`margin`).

3. Flex položky lze pozicovat

Pozicování prvků (`position: absolute|relative|fixed`) lze na rozdíl od vlastností souvisejících s `float` na flex položky běžně aplikovat.

4. S `visibility: collapse` flexbox pracuje jako se řádky tabulky

`visibility: collapse` funguje u flex položek stejně jako u `display: table-row` nebo `table-column` elementů. A to tak, že element okupuje místo a v DOMu se s ním počítá, jen není vidět.

5. Směr hlavní osy flexboxu řídí jazyk rozhraní

Směr hlavní osy flex kontejneru vychází vždy z vlastnosti `writing-mode`. Pokud bychom tedy flexboxem dělali layout stránky v japonštině, budou všechny hodnoty zde zmíněné naopak.

Pojďme na detailní referenční příručku. Už víme, že flex elementy jsou dvojího typu – flex kontejner a flex položka.

Flexbox: zajímavé odkazy

- **A Complete Guide to Flexbox**
  Vizuální referenční příručka Chrise Coyiera. https://css-tricks.com/snippets/css/a-guide-to-flexbox
- **Flexy Boxes**
  Vizuální klikátko pro generování flexbox layoutů a hraní si. http://the-echoplex.net/flexyboxes
- **flexbox in 5 minutes**
  Nestačil ebook? Vysvětlení principů flexboxu z jiné strany. http://flexboxin5.com
- **Flexbox dema**
  Kolekce příkladů k flexboxu. Částečně znáte už z tohoto ebooku. http://codepen.io/collection/Dmrzxz
- **Solved by Flexbox**
  Časté problémy CSS layoutu řešené pomocí Flexboxu. http://philipwalton.github.io/solved-by-flexbox
- **CSS Flexible Box Layout**
  W3.org specifikace. Nebojte, hezky napsaná i z pohledu vývojáře. http://www.w3.org/TR/css3-flexbox

Další

Nové CSS3 jednotky: rem, vw, vh

rem

Rozměr, který odpovídá hodnotě `font-size` na root elementu, tedy `<html>`. *„root-emka“* jsou variantou známé jednotky `em`. Od běžných `em` se liší tím, že nevychází z velikosti fontu rodičovského elementu.

Velikost písma pro `<html>` element je v prohlížečích obvykle nastavená tak, aby odpovídala `16px`.

Pokud všechny rozměry týkající se typografie (nebo klidně i layoutu) nastavíte v `rem` jednotkách, můžete si snadno zvětšit celý dokument pouhou změnou hodnoty ve vlastnosti `html { font-size: … }` a vytvářet tak elastické layouty.

Podobně jako v příkladu výše můžete změnu velikosti písma celého dokumentu a s ním i elastické zvětšení layoutu provést pomocí media query.

PŘÍKLAD

Všechny rozměry v dokumentu nastavíme pomocí jednotky `rem`. Pokud tedy nadpisy první úrovně nastavíme na `1.5rem`, budou velké 1,5 × `16px` – tedy `24px`.

```
  h1 {
 font-size: 1.5rem;
}
```

Pokud se tedy pomocí media query rozhodneme, že od šířky okna 801 pixelů nastavíme základní velikost písma na 25 pixelů, …

```
  @media (min-width: 801px) {
 html {
   font-size: 25px;
 }
}
```

… zvětší se nám všechny rozměry nastavené v jednotkách `rem`. Nadpis `<h1>` tedy bude mít v těchto šířkách okna velikost 38 pixelů (`25px` × 1,5).

Na živo můžete vyzkoušet tady: cdpn.io/e/mnbaA.

PODPORA V PROHLÍŽEČÍCH

IE9+. Pro starší prohlížeče lze vytvořit pixelový fallback:

```
  font-size: 24px;
font-size: 1.5rem;
```

Fallback je lepší nechat si generovat automaticky, například pomocí CSS preprocesoru.

Více o podpoře v prohlížečích: caniuse.com/rem.

Jednotky viewportu: vw, vh, vmin, vmax

Umožňují definovat rozměry v CSS relativně k velikosti viewportu, zjednodušeně řečeno výšce nebo šířce okna.

- `vw` – zkratka pro „viewport width“ – `1vw` označuje setinu šířky viewportu
- `vh` – zkratka pro „viewport height“ – `1vh` označuje setinu výšky viewportu
- `vmin` – zkratka pro „viewport minimum“ – reprezentuje menší hodnotu z porovnání `1vw` a `1vh`
- `vmax` – zkratka pro „viewport maximum“ – reprezentuje větší hodnotu z porovnání `1vw` a `1vh`

PRVNÍ PŘÍKLAD: ROZTAŽENÍ ELEMENTU NA CELOU VÝŠKU OKNA POMOCÍ `vh`

Na rozdíl od procent se jednotky viewportu nevztahují k rozměrům nejbližšího rodiče, ale k šířce a výšce okna prohlížeče. Lze s nimi tedy dělat kouzla, která dříve byla možná jen pomocí CSS hacků nebo Javascriptu.

Příkladem budiž roztažení výšky layoutu stránky na celou výšku okna prohlížeče:

```
  .container {
  height: 100vh;
}
```

DRUHÝ PŘÍKLAD: ELASTICKÁ TYPOGRAFIE S `vw`

S *volkswageny* můžete užitečné věci dělat v oblasti velikosti písma.

Nejjednodušší příklad vypadá takto:

```
  h1 {
  font-size: 6vw;
}
```

Písmo se pak bude zvětšovat podle šířky okna. Zkuste si to sami:

- Téměř elastická typografie: css-tricks.com/viewport-sized-typography/.

- Složitější, ale plně elastická typografie: smashingmagazine.com/2016/05/fluid-typography/.
- Elastická typografie počítaná v procentech z výšky komponenty: vrdl.cz/prirucka/reseni-elasticka-typografie.

TŘETÍ PŘÍKLAD: VELIKOST TEXTU V HLAVNÍM NADPISE STRÁNKY POMOCÍ `vmin`

Jak vám možná došlo u `vw` typografie, nadpisy mohou být na malých displejích příliš malé a na velkých příliš velké. Když se nechcete trápit s Media Queries, zkuste `vmin`.

```
h1 {
  font-size: 6vmin;
}
```

Písmo se pak na malém displeji v režimu na výšku sází v procentech ze šířky. Na velkém monitoru zase v procentech z výšky. Ve spoustě situací se tak obejdete bez Media Queries.

Více v článku „MinMaxing: Understanding vMin and vMax in CSS“ na TheNewCode.com. vrdl.in/afq9y.

PODPORA V PROHLÍŽEČÍCH

V posledních verzích umí všechny moderní prohlížeče kromě Opery Mini: caniuse.com/viewport-units.

Je to ovšem trochu složitější:

- IE9 namísto `vmin` používá `vm`.
- IE10 neumí `vmax`.
- Safari na iOS6+7 má hned několik chyb souvisejících s jednotkami viewportu. Sledujte odkazy na caniuse.com/viewport-units.
- IE8, Android Browser až do verze 4.3 a Opera Mini tyto jednotky neumí vůbec.

RGBa: barva s poloprůhledností

RGB barva se čtvrtým číslem, jež alfa kanálem říká informaci o hodnotě průhlednosti.

```
rgba(_red_, _green_, _blue_, _pruhlednost_);
```

Poloprůhledné RGBa barvy můžete samozřejmě aplikovat všude, kde se v CSS barvy používají. Zdůrazním hlavně rámečky, stíny nebo gradienty.

Porovnání s `opacity`

`opacity: 0.5` zajistí poloprůhlednost celého elementu, ale i jeho dceřiných objektů.

RGBa je barva aplikovatelná na cokoliv bez vlivu na zbytek elementu. Třeba jen na barvu pozadí `background: rgba(20%, 100%, 20%, .5)`.

V prohlížeči vyťukejte cdpn.io/e/HrBsD.

Podpora v prohlížečích

RGBa mají rády všechny prohlížeče kromě osmičky a starších MSIE. Elegantně si se stařečky poradíte definovaným fallbackem:

```
color: rgb(128, 0, 0);
color: rgba(255, 0, 0, 0.5);
```

V moderních browserech se zobrazí červená s padesátiprocentní průhledností. V IE8– pak tmavý odstín červené. Fallback barvu musíme určit s ohledem na barvu pozadí. Tady počítáme s černou.

Alternativně použijte CSS3Pie.com.

Selektory

CSS3 přichází s armádou nových selektorů. Nejjednodušší bude podívat se na ně optikou toho, jakou verzi Internet Exploreru na svých projektech musíte podporovat, abyste je mohli použít.

IE8+

- `span[att^="val"]` – hodnota atributu `attr` začíná řetězcem „val“. Hojně využívají CSS frameworky pro tvorbu gridu. Třídy `.span1`, `.span2` atd. selektují takto: `span[class^="span"]`
- `span[att$="val"]` – hodnota atributu končí řetězcem „val“.
- `span[att*="val"]` – hodnota atributu obsahuje řetězcem „val“.
- `p ~ ul` – mají stejného rodiče, ale nejsou vedle sebe jako u `p + ul`.

IE9+

- `span:empty` – všechny `<span>` elementy, které neobsahují dceřiný element ani text.
- `div:target` – vybírá část zvýrazněnou přes kotvu (`/adresa#kotva`).
- `input:enabled`, `input:disabled` – formulářové pole, kam není možné zapisovat.
- `input:checked` – vybírá aktivovaný input typu `checkbox`, `radio` nebo `option`.
- `*::selection` – nastylování vybraného textu.
- `span:not(.blue)` – všechny spany kromě těch s třídou `blue`.
- `:first-child`, `:last-child`, `:only-child` – první, poslední nebo jediný potomek svého rodiče.
- `:nth-child(n)`, `:nth-last-child(n)` – n-tý potomek od začátku nebo od konce. Nejlépe se tento šílený (ale moc užitečný) selektor naučíte s pomocí online nástroje NthMaster.
- `:first-of-type`, `:last-of-type`, `:only-of-type` – první, poslední, jediný tohoto typu elementu.

Funkce calc()

Funkce, která umožňuje vložit matematický výraz namísto hodnoty vlastnosti.

Je velmi dobře podporovaná, ale málo se o ní ví. Je užitečná, i když se to občas zpochybňuje. Pojďme to napravit. Nejprve si ukažme dvě jednoduchá využití:

```
   .el {
  width: calc(100% / 3);
  margin-bottom: calc(1em - 2px);
}
```

Není to stejné jako matematika v preprocesorech?

Není. V preprocesoru se musíme spokojit s výrazy, které se mohou zkompilovat do CSS ještě předtím, než prohlížeč stránku vidí:

```
   width: (100% / 3)
// → width: 33.33333333%
```

Jenže preprocesoru dochází dech v momentě, kdy potřebuji kombinovat více jednotek. Vezměme příklad třísloupcového rozvržení s vnějším okrajem o šířce `1em`:

```
   width: calc(100% / 3 - (2 * 1em))
```

Že se vám to někdy hodilo? Že to obcházíte podivnými hacky? Vítejte v klubu!

Podpora v prohlížečích

Funkci `calc()` nepodporuje hlavně Internet Explorer 8, jeho starší sourozenci a ani Android Browser. V době psaní textu mohou u průměrného českého webu tvořit maximálně něco kolem 3–4 % návštěvnosti.

Pokud funkci používáte, myslete na tyto uživatele, a pokud je to potřeba, poskytněte jim alternativu v podobě definovaného fallbacku. Může vypadat mírně jinak. Je to lepší, než když se ve starém prohlížeči rozpadnou důležité věci.

DEFINOVANÝ FALLBACK

```
   /* IE8 a spol: */
width: 30%;
/* Moderní prohlížeče: */
width: calc(100% / 3 - (2 * 1em));
```

A prosím pěkně: pozor na chyby v některých nechvalně známých prohlížečích. Na této stránce doporučuji kliknout na „Known issues“ a hledat „crash“: caniuse.com/calc.

POLYFILL

Vždycky říkám, že používání polyfillu na zásadní věci týkající se layoutu je dost nebezpečné. Myslete na situaci, kdy selže Javascript. Myslete na vykreslovací výkon. Myslete na svoje nervy.

Pokud je i tak použití `calc` ve starých prohlížečích nezbytné, z polyfillů vezměte tento: github.com/closingtag/calc-polyfill.

DETEKCE VLASTNOSTI

Knihovna Modernizr umí funkci detekovat, takže směle do toho:

```
   .no-csscalc .el {
  /* IE8 a spol: */
}
.csscalc .el {
  /* Moderní prohlížeče: */
}
```

Příklady s funkcí calc(): kdy ji využít a kdy naopak ne?

1) Ukaž matiku!

```
width: calc(100% / 7);
```

Proč nenapsat rovnou `width: 14.2857`? Ze dvou důvodů:

1. **Čitelnost kódu.** Vsaďte se, že na původ čísla 14,2857 zapomenete. Nejpozději za týden. Správa této části kódu pro vás pak bude znamenat kladení otázek „jak jsem k tomu číslu došel?“.
2. **Zaokrouhlování.** V celé krásе vypadá takto: 14,285714286. Kodéři občas podlehnou pokušení a desetinná místa zaokrouhlí. A pak se diví, že se jim v nějakém konkrétním rozlišení a prohlížeči rozpadlo rozvržení stránky. Nebuďte takoví kodéři. Nezaokrouhlujte.

2) Responzivní obrázky

V parametru `sizes` značky `<img>` se bez `calc()` nedá obejít. Zkuste si bez téhle prima funkce spočítat šířku obrázku uvnitř layoutu, který v responzivním layoutu splňuje následující podmínky:

1. Na větších displejích zabírá polovinu šířky layoutu stránky. Ale bez vnějších okrajů layoutu (10 pixelů) a samotné stránky (15 pixelů).
2. Na menších zabírá celou šířku viewportu bez vnějších okrajů stránky (15 pixelů).

```
 <img
sizes="calc((100vw - 2 * 15px) / 2) - 10px),
       calc(100vw - 2 * 15px)"
>
```

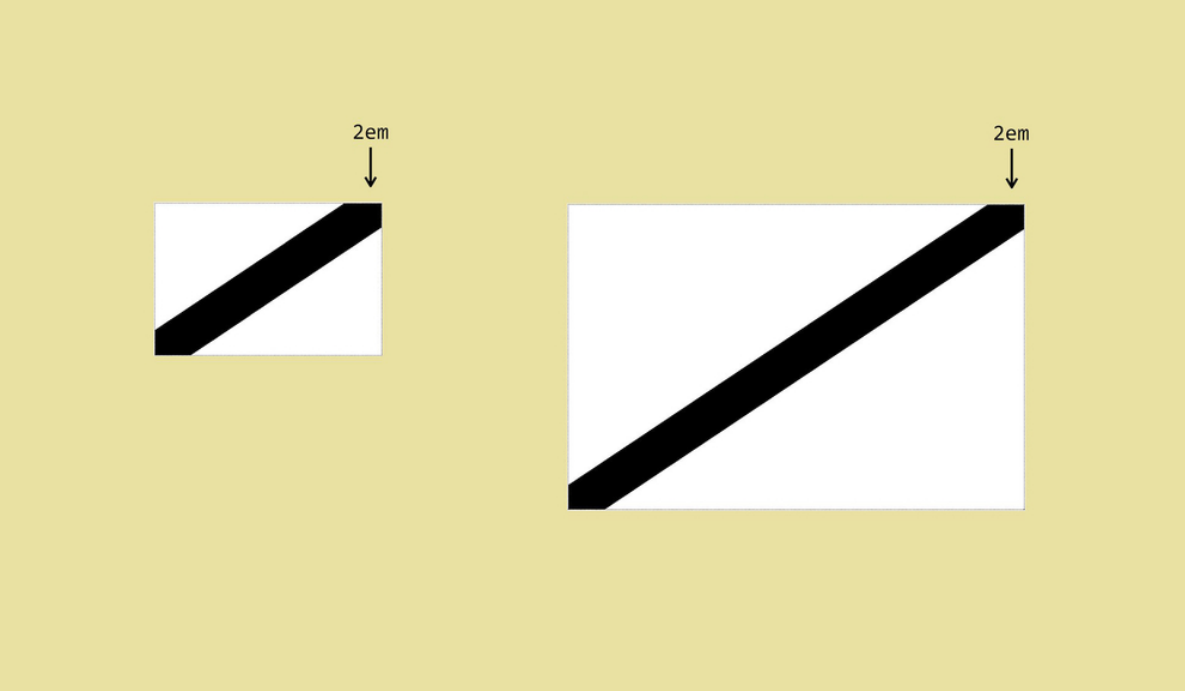
Více o parametrech `srcset` a `sizes` najdete na Vzhůru dolů. vrdl.cz/prirucka/srcset-sizes

Chci zmínit, že knihovna pro práci s responzivními obrázky – Picturefill – už obsahuje také polyfill pro funkci `calc()`. Dostanete s ní tedy řešení pro všechny existující prohlížeče.

3) Barevný přechod se stabilně širokou částí

Vezměme gradient, jehož část chceme definovat stabilní šířkou a ostatní části už standardně podílem z celku.

Více řekne obrázek. Černá část bude mít stabilní šířku. Bílá plocha se zmenšuje a zvětšuje podle dostupného místa.



Takto vypadá kód:

```
 background:
linear-gradient(to right bottom,
  transparent calc(50% - 2em),
  #000 0,
  #000 calc(50% + 2em),
  transparent 0);
```

Tady je pak živá ukázka od Any Tudor: cdpn.io/e/YyGPJo.

Ne všechno, co se třpytí, je `calc()`. Pojďme se teď podívat na příklady užití, které doporučují různé články na webu a které je lepší řešit jinak.

Tři známá využití `calc()`, která naopak stojí za starou bačkoru

1. **Dopočítávání zbytku výšky nebo šířky.**
   Máme fixně vysokou hlavičku. Tělo dokumentu pak má zabrat zbytek výšky okna. Ano, dá se to udělat pomocí něčeho jako `height: calc(100% - {výška-hlavičky})`. Ale daleko elegantněji to uděláte flexboxem, nástrojem, který byl pro účely tvorby layoutu vymyšlen. vrdl.in/53f64

2. **Náhrada `box-sizing`.**
   Ano, můžete použít něco jako `width: calc({width} + {padding} * 2)`. Ale proč byste to dělali? Změna počítání šířky a výšky pomocí vlastnosti Box Sizing má daleko lepší podporu než `calc()`. vrdl.in/53f64
3. **Posun obrázku na pozadí zezdola a zprava.**
   Je možné použít zápis typu `background-position: calc(100% - {posun-zezdola}) calc(100% - {posun-zprava})`. Lepší ale je využít čtyřčíselný zápis pro pozicování obrázku s posunem, který má širší podporu mezi prohlížeči. caniuse.com/css-background-offsets cdpn.io/e/OXpqRm

Takže – `calc()` se hodí hlavně pro responzivní obrázky a pro zpřehlednění kódu. Máte jiné využití? Budu rád, když mi napíšete.

Box Reflection: odlesk objektu

Prohlížeč vykreslí zrcadlový odlesk pod objektem nebo z jeho strany.

Syntaxe

Syntaxe pro prohlížeče postavené na Webkit jádře:

```
-webkit-box-reflect:
_směr_odlesku
_posun_
_maska_;
```

SMĚR ODLESKU

Povinná položka, která může nabývat hodnot `above`, `below`, `left` nebo `right`.

```
-webkit-box-reflect: below;
```

POSUN

Jak moc se odlesk vzdálí od původního objektu. Uvádí se v běžných CSS jednotkách – `px`, `em` a dalších.

```
-webkit-box-reflect: below 5px;
```

MASKA

Maska určuje efekt překrytí odlesku. Pro zajištění zrcadlového efektu maskou bývá CSS gradient. Může to být ale i obrázek. Co je v masce černé, zobrazí se; co je průhledné, nezobrazí se.

```
-webkit-box-reflect:
below 5px linear-gradient(to bottom, transparent, black);
```

Živá ukázka příkladu je na cdpn.io/e/CLEhF.

Podpora v prohlížečích

Jen prohlížeče postavené na Webkit jádře. Ke dnešku tedy funguje ve všech Safari i Android Browseru. Dokonce ve všech Chrome, i když už dnes běží na vlastním jádře Blink. caniuse.com/box-reflect

Ve Firefoxu lze odlesku dosáhnout pomocí vlastnosti `-moz-element()`. vrdl.in/yboz2

Na vlastnost se tedy nelze spolehnout na široce dostupných webech, ale pro interní aplikace s omezenou cílovou skupinou se může hodit.

Vlastnosti nezařazené v tomto textu

Většinou jde o technologie, které zatím nemají širokou podporu prohlížečů a hodí se jen pro speciální případy případy. Přidávám i ty, které sice stojí za pozornost, ale zatím jsem je nezdokumentoval.

Filtry

Aplikování grafických filtrů na objekty nebo obrázky. Podpora je zatím horší, v Exploreru to nefunguje a čeká se na zapnutí v produkčním Edge. Raději upozorňuji, že to nemá nic společného s funkcí `filter()` známou z dřívějších Explorerů. Filtry umí rozostření, jas, kontrast, stín a mnoho dalších, které znáte z grafických programů.

- caniuse.com/css-filters
- jecas.cz/filter
- w3.org/TR/filter-effects

Masky

Zobrazení obrázku nebo elementu přes masku tvořenou jiným obrázkem. Hodilo by se, ale podpora je zatím mizerná.

- caniuse.com/masks
- jecas.cz/mask
- w3.org/TR/css-masking

Grid (mřížka)

Layout do mřížky. Zatímco flexbox je vymyšlený pro design komponent uživatelského rozhraní, grid pro layout celých stránek. Podpora je v době psaní jen experimentální. Existuje sice polyfill, pro layout bych ho ovšem používat nedoporučoval. Každopádně grid layout bude po flexboxu další velká věc, takže doporučuji sledovat jeho vývoj.

- caniuse.com/grid
- w3.org/TR/css3-grid-layout

Hyphens (spojovníky)

Definuje, zda budou v případě potřeby slova na konci řádků automaticky rozdělována pomocí spojovníků. Co podpora? Kromě Chrome, Opery a Android Browseru se to už naučily všechny prohlížeče. Ale – vzhledem k povaze vlastnosti – jejímu využití nic nebrání. S českým textem to bude nejlépe fungovat v Exploreru.

- caniuse.com/hyphens
- jecas.cz/hyphens
- w3.org/TR/css-text-3

@supports (detekce podpory vlastností)

Testuje dostupnost CSS vlastností v prohlížeči. Standardizovaná náhrada javascriptové knihovny Modernizr. Dnes už podporují všechny moderní prohlížeče, jen ten Explorer to už nedožene. Doufám, že o `@supports` brzy napíšu více.

- caniuse.com/supports
- jecas.cz/supports
- w3.org/TR/css3-conditional

Na závěr

Kam dál?

- Blog Vzhůru dolů: vzhurudolu.cz
- Facebook blogu: facebook.com/vzhurudolu
- Autor na Twitteru: twitter.com/machal

Autorovy kurzy

- Navazující školení „Dnešní webová kodéřina“: vrdl.cz/kurzy/webova-koderina
- Další školení autora a spolupracujících expertů: vrdl.cz/kurzy

Poděkování

Za grafiku pro přebal knihy děkuji Petru Šťastnému (raist.cz) a za korektury Petru Behúnovi (proofreading.cz). Foto autora je od Jana Forejta (about.me/janforejt).

Připomínkami k verzím 1.1 až 1.4 přispěli: Jan Bien, Jiří Helmich, Bohumil Jahoda, Bea Jakubcová, Milan Lempera, Vladislav Musílek, Martin Kotraš, Jiří Pavlíček, Martin Pešout, Miroslav Pokorný, Luděk Roleček, Michal Staněk, Daniel Střelec, Matúš Vašičkanin, Pavel Vosyka. Děkuji všem v abecedním pořadí.

ISBN: 978-80-260-8438-9