Technologia i rozwiązania

Responsive Web Design

Projektowanie elastycznych witryn w HTML5 i CSS3

Wydanie II

Ben Frain

Helion

[PACKT] PUBLISHING

Tytuł oryginału: Responsive Web Design with HTML5 and CSS3, 2nd Edition

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-283-2343-8

First published in the English language under the title 'Responsive Web Design with HTML5 and CSS3 – Second Edition – (9781784398934)'

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/trash2.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/trash2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 2

Zapytania medialne: obsługa zróżnicowanych obszarów roboczych

W poprzednim rozdziale przyjrzeliśmy się podstawowym składnikom responsywnej strony internetowej: płynnemu układowi, płynnym obrazom i zapytaniom medialnym.

Tutaj dokładnie przyjrzymy się zapytaniom medialnym, aby poznać ich wszystkie możliwości oraz elementy składni.

W tym rozdziale:

- dowiesz się, dlaczego zapytania medialne są potrzebne w RWD;
- poznasz składnię zapytań medialnych;
- nauczysz się definiować zapytania medialne za pomocą znacznika `<link>`, dyrektywy `@import` oraz w plikach CSS;
- dowiesz się, jakie właściwości urządzeń można testować za pomocą zapytań medialnych;
- nauczysz się wprowadzać wizualne zmiany na stronie za pomocą zapytań medialnych w zależności od ilości dostępnego miejsca na ekranie;
- poznasz argumenty przedstawiane w dyskusji, czy zapytania medialne powinno się grupować w jednym miejscu, czy wpisywać w tych miejscach, w których są potrzebne;

- nauczysz się korzystać ze znacznika `meta viewport`, dzięki któremu zapytania medialne mogą działać zgodnie z przeznaczeniem także w urządzeniach z systemami iOS i Android;
- poznasz kolejne funkcje proponowane do dodania do następnych wersji specyfikacji zapytań medialnych.

Specyfikacja CSS3 składa się z pewnej liczby modułów. Jednym z nich jest Media Queries (Level 3), czyli zapytania medialne. Za ich pomocą można wybierać, które zestawy reguł CSS mają zostać zastosowane w urządzeniach o określonych możliwościach. Przykładowo: wystarczy tylko kilka wierszy kodu CSS, aby zmienić sposób wyświetlania strony w odpowiedzi na zmianę szerokości obszaru roboczego, współczynnika proporcji ekranu, orientacji (pionowa lub pozioma) itd.

Zapytania medialne są dobrze obsługiwane przez przeglądarki. Obsługują je właściwie wszystkie aplikacje oprócz przedpotopowych potworów (od IE 8 w dół). Krótko mówiąc: nie ma żadnego powodu, aby się przed nimi bronić!

Specyfikacje W3C przechodzą przez proces ratyfikacji w konsorcjum W3C (jeśli masz wolny dzień, zapoznaj się z przebiegiem tego procesu przedstawionym na stronie: *http://www.w3.org/2005/10/Process-20051014/tr*). Mówiąc najkrócej: każda specyfikacja najpierw jest tzw. projektem roboczym (ang. *Working Draft* — *WD*), potem przechodzi w fazę **rekomendacji kandydującej** (ang. *Candidate Recommendation* — CR), następnie zamienia się w **propozycję rekomendacji** (ang. *Proposed Recommendation* — PR), a na koniec, po wielu latach prac, otrzymujemy oficjalną rekomendację W3C (ang. *W3C Recommendation* — REC). Wynika z tego, że bezpieczniejszymi modułami są te, które osiągnęły wyższy stopień dojrzałości. Przykładowo: moduł transformacji CSS poziomu 3. (*http://www.w3.org/TR/css3-3d-transforms/*) osiągnął status projektu roboczego w marcu 2009 r., w związku z czym jest on obsługiwany znacznie gorzej niż moduły będące na etapie CR, takie jak zapytania medialne.

Dlaczego zapytania medialne są potrzebne do budowy układów responsywnych

Zapytania medialne CSS3 umożliwiają tworzenie reguł przeznaczonych tylko dla urządzeń o określonych możliwościach lub do zastosowania wyłącznie w wybranych warunkach. W specyfikacji W3C zapytań medialnych (*http://www.w3.org/TR/css3-mediaqueries/*) można przeczytać następujące wprowadzenie do tej technologii:

> „Zapytanie medialne składa się z części określającej typ medium i opcjonalnie kilku wyrażeń, które sprawdzają, czy zaszły warunki uruchamiające konkretne reguły. Wśród właściwości urządzeń, jakie można sprawdzać za pomocą zapytań medialnych, znajdują się: szerokość, wysokość i kolor. Przy użyciu zapytań medialnych można dostosować prezentację do wybranych rodzajów urządzeń wyjściowych bez zmieniania samej treści".

Bez zapytań medialnych, za pomocą samych arkuszy stylów, nie dałoby się radykalnie zmieniać warstwy wizualnej strony. Dzięki tym zapytaniom możemy pisać reguły CSS przygotowujące naszą stronę na takie ewentualności, jak zmiana orientacji ekranu, zmniejszenie lub powiększenie obszaru roboczego itd.

Choć układ płynny również sprawia, że projekt prezentuje się lepiej przy różnych szerokościach ekranu, to jeśli wziąć pod uwagę różnorodność urządzeń, czasami potrzebne są bardziej kompleksowe rozwiązania. Ich wdrożenie jest możliwe dzięki zapytaniom medialnym. Traktuj je jak podstawową logikę warunkową CSS.

Podstawowa logika warunkowa w CSS

Wszystkie prawdziwe języki programowania zawierają konstrukcje pozwalające wybrać jedną lub więcej gałęzi kodu do wykonania w zależności od pewnych warunków. Najczęściej mają one postać powszechnie używanych instrukcji `if-else`.

Jeśli kod źródłowy języków programowania kłuje Cię w oczy, nie musisz się przejmować. To, o czym teraz piszę, to bardzo prosta koncepcja. Posługujesz się nią np., gdy prosisz kolegę z pracy, aby zamówił Ci coś w kawiarni: „Jeśli mają babeczki z potrójną czekoladą, to weź mi jedną, a jeśli nie, to weź mi kawałek ciasta marchewkowego". Jest to prosta instrukcja warunkowa z dwoma możliwymi wynikami (i każdy jest tak samo przyjemny).

W czasie, gdy piszę tę książkę, w CSS nie ma instrukcji logicznych ani żadnych innych konstrukcji typowych dla prawdziwych języków programowania. Pętle, funkcje, iteracja i skomplikowane obliczenia matematyczne nadal należą do świata procesorów CSS (wspominałem już o bardzo ciekawej książce o Sass pt. *Sass i Compass. Praktyczny przewodnik dla projektantów*?). A jednak zapytania medialne są rodzajem mechanizmu umożliwiającego wyrażanie prostej logiki warunkowej w CSS. Zawarte w nich reguły znajdują zastosowanie tylko wtedy, gdy zostaną spełnione określone warunki. **Funkcje programistyczne w przygotowaniu**

> Popularność preprocesorów CSS nie umknęła uwadze ludzi pracujących nad specyfikacjami tej technologii. Trwają już prace nad wstępną wersją roboczą specyfikacji zmiennych CSS: *http://www.w3.org/TR/css-variables/*.
>
> Niestety, obecnie obsługuje je tylko przeglądarka Firefox, więc na razie zmiennych nie można wykorzystywać w realnych projektach.

Składnia zapytań medialnych

Jak wygląda zapytanie medialne i — co ważniejsze — jak ono działa?

Wprowadź poniższy kod na samym dole dowolnego pliku CSS i obejrzyj w przeglądarce stronę internetową, do której jest on dołączony. Ewentualnie otwórz przykładowy plik z folderu *02-01*:

```
body {
   background-color: grey;
}
@media screen and (min-width: 320px) {
   body {
      background-color: green;
   }
}
@media screen and (min-width: 550px) {
   body {
      background-color: yellow;
   }
}
@media screen and (min-width: 768px) {
   body {
      background-color: orange;
   }
}
@media screen and (min-width: 960px) {
   body {
      background-color: red;
   }
}
```

Otwórz teraz stronę internetową w przeglądarce i zmień wielkość okna. Kolor strony zmieni się w zależności od aktualnych wymiarów obszaru roboczego. Bardziej szczegółowy opis składni zapytań medialnych zamieściłem dalej. Najpierw chciałbym Ci pokazać, jak i gdzie można ich używać.

Zapytania medialne w znaczniku <link>

Jeżeli pracowałeś już z arkuszami stylów CSS 2, to wiesz, że można w nich zadeklarować rodzaj urządzenia (używając słów kluczowych `screen` i `print`) w atrybucie `media` znacznika `<link>`. Taką deklarację należy umieścić wewnątrz elementu `<head>`; ma ona następującą postać:

```
<link rel="stylesheet" type="text/css" media="screen" href="screen-styles.css">
```

Zapytania medialne pozwalają na dopasowanie formatowania strony do możliwości i właściwości urządzenia, a nie tylko do jego typu. Można to potraktować jak skierowane do przeglądarki zapytanie. Jeśli przeglądarka „stwierdzi", że podane w zapytaniu warunki zostały spełnione, nada właściwe formatowanie. Jeżeli tak nie będzie, zastosowane zostanie inne. Zamiast wysyłać do przeglądarki zapytanie: „Czy jesteś ekranem?" — co było dawniej szczytem możliwości CSS 2 — zapytania medialne pozwalają nam na większą dociekliwość. Za ich pomocą możemy zapytać: „Czy jesteś ekranem w orientacji pionowej?". Rzuć okiem na poniższy przykład:

```
<link rel="stylesheet" media="screen and (orientation: portrait)"
↪href="portrait-screen.css" />
```

Najpierw zapytanie medialne prosi o potwierdzenie typu urządzenia („Czy jesteś ekranem?"), a następnie właściwości („Czy twój obraz ma orientację pionową?"). Arkusz stylów *portrait--screen.css* zostanie załadowany dla dowolnego urządzenia w trybie pionowym i zignorowany w innych. Istnieje możliwość odwrócenia zapytania poprzez dodanie na początku zapytania medialnego słowa kluczowego `not`. Ilustruje to niżej przedstawiona deklaracja, która odwraca działanie kodu z poprzedniego przykładu i powoduje załadowanie pliku, jeśli tylko dany ekran nie ma orientacji pionowej:

```
<link rel="stylesheet" media="not screen and (orientation: portrait)"
↪href="portrait-screen.css" />
```

Łączenie zapytań medialnych

Istnieje również możliwość połączenia wielu wyrażeń. Rozszerzmy nasz przykład pierwszego zapytania medialnego i ograniczmy stosowanie pliku do urządzeń, które mają obszar operacyjny większy niż 800 pikseli.

```
<link rel="stylesheet" media="screen and (orientation: portrait)
↪and (min-width:  800px)" href="800wide-portrait-screen.css" />
```

Możemy też przygotować całą listę zapytań medialnych. Jeśli dowolne z wypisanych w zapytaniu warunków będą zgodne z prawdą, plik zostanie załadowany. Jeśli żaden z nich nie zostanie spełniony, reguły nie zostaną odczytane:

```
<link rel="stylesheet" media="screen and (orientation: portrait) and
↪(min-width:  800px),  projection" href="800wide-portrait-screen.css" />
```

W tym miejscu należy zwrócić uwagę na dwa ważne aspekty: po pierwsze, każde zapytanie medialne jest oddzielone przecinkiem, po drugie, łatwo zauważyć, że po słowie kluczowym `projection` nie występują żadne uzupełniające słowa `and` i nie ma wartości podanych w nawiasach. Oznacza to, że przy braku zdefiniowanych wartości dane zapytanie medialne będzie dotyczyć wszystkich typów danego urządzenia. W naszym przykładzie formatowanie to będzie dotyczyć wszystkich projektorów.

> W zapytaniach medialnych można używać dowolnej jednostki miary CSS. Do najczęściej stosowanych należą **piksele** (`px`), ale równie dobrze można się posługiwać jednostkami **em** i **rem**. Więcej informacji na ten temat można znaleźć w moim artykule dostępnym pod adresem: *http://benfrain.com/just-use-pixels*.
>
> Jeśli więc chcesz ustawić punkt kontrolny na 800 pikselach (tylko wyrażonych w jednostce `em`), po prostu podziel liczbę pikseli przez 16. Na przykład `800px` jest równe `50em` (800/16=50).

Importowanie zapytań medialnych za pomocą dyrektywy @import

Możemy też użyć dyrektywy `@import`, która jeśli zostaną spełnione pewne warunki, wczyta arkusz stylów w innym arkuszu. Następujący kod załadowałby arkusz *phone.css*, ale tylko jeśli urządzenie, w którym strona została otwarta, miałoby obszar roboczy nie szerszy niż 360 pikseli:

```
@import url("phone.css") screen and (max-width:360px);
```

Pamiętaj, że użycie dyrektywy `@import` oznacza dodatkowe żądanie HTTP (które wpływa na szybkość wczytywania strony), polegaj więc na tej metodzie tylko okazyjnie.

Zapytania medialne w arkuszach stylów

Wiesz już, jak stosować zapytania medialne w odnośnikach do plików CSS umieszczonych w nagłówku (między znacznikami `<head>` i `</head>`) dokumentu HTML oraz za pomocą dyrektywy `@import`. Jednak najczęściej zapytania medialne wpisuje się bezpośrednio w arkuszach stylów. Gdybyśmy np. dodali do arkusza stylów poniższy kod, to wszystkie elementy `<h1>` na stronie stałyby się zielone, ale tylko w urządzeniach o szerokości ekranu nieprzekraczającej 400 pikseli.

```
@media screen and (max-device-width: 400px) {
  h1 { color: green }
}
```

Najpierw za pomocą reguły `@media` informujemy przeglądarkę, że tworzymy zapytanie medialne, a następnie określamy, jakie urządzenia nas interesują. W tym przykładzie odnosimy się tylko do ekranów (a więc np. nie do druku — `print`). Później w nawiasie podajemy konkretne właściwości wybranego urządzenia. Dalej znajdują się już normalne reguły CSS.

W tym miejscu powinienem chyba podkreślić, że oznaczanie typu mediów jako `screen` jest opcjonalne. Oto odpowiedni fragment specyfikacji, który o tym mówi:

> „Jeśli zapytanie medialne odnosi się do wszystkich rodzajów mediów, to można w jego definicji posłużyć się składnią skróconą, tzn. można opuścić słowo kluczowe `all` (wraz ze znajdującym się za nim słowem `and`). Innymi słowy: jeśli typ mediów nie zostanie określony, to znaczy, że chodzi o wszystkie typy mediów”.

Jeżeli więc nie piszesz stylów przeznaczonych wyłącznie dla konkretnego rodzaju mediów, możesz opuścić człon `screen and`. Robię to we wszystkich następnych przykładach.

Co można sprawdzać za pomocą zapytań medialnych

Projektanci stron responsywnych najczęściej używają zapytań medialnych do sprawdzania szerokości obszaru roboczego (`width`). Z mojego doświadczenia wynika, że pozostałe parametry (może z wyjątkiem okazjonalnego sprawdzania rozdzielczości ekranu i wysokości obszaru robo-

czego) nie są zbyt użyteczne. Na wszelki wypadek jednak zamieszczam pełną listę parametrów, które można sprawdzać za pomocą zapytań medialnych poziomu 3. Część z nich może Ci się przydać:

- `width` — szerokość obszaru roboczego.
- `height` — wysokość obszaru roboczego.
- `device-width` — szerokość powierzchni wyświetlania (będziemy ją traktować jako szerokość urządzenia).
- `device-height` — wysokość powierzchni wyświetlania (będziemy ją traktować jako wysokość urządzenia).
- `orientation` — ta własność służy do sprawdzenia, czy urządzenie jest ustawione poziomo, czy pionowo.
- `aspect-ratio` — stosunek szerokości obszaru roboczego do jego wysokości. Stosunek 16:9 można zapisać jako `aspect-ratio: 16/9`.
- `device-aspect-ratio` — parametr podobny do `aspect-ratio`, tylko odnoszący się do szerokości i wysokości powierzchni wyświetlania urządzenia, a nie do obszaru roboczego.
- `color` — liczba bitów przypadająca na każdy składnik koloru. Przykładowa własność min-color:16 sprawdza, czy urządzenie wyświetla 16-bitowe kolory.
- `color-index` — liczba wpisów w tabeli przeglądu kolorów urządzenia. Wartości muszą być liczbami i nie mogą być ujemne.
- `monochrome` — parametr służący do sprawdzania, ile bitów przypada na piksel w monochromatycznym buforze klatek. Wartością jest liczba całkowita, np. `2` — ta liczba nie może być ujemna.
- `resolution` — ta opcja służy do sprawdzania rozdzielczości ekranu lub wydruku, np. deklaracja `min-resolution: 300dpi`. Własność ta przyjmuje też wartości wyrażone w liczbie plamek na centymetr (ang. *dots per centimeter*), np. deklaracja `min-resolution: 118dpcm`.
- `scan` — ten parametr odnosi się głównie do telewizorów i służy do sprawdzania, czy stosowane jest skanowanie progresywne, czy przeplatane. Przykładowo: do telewizora 720p HD (litera *p* w 720p oznacza wyświetlanie „progresywne”) można się odwołać za pomocą deklaracji `scan: progressive`, podczas gdy do telewizora 1080i HD (litera *i* w 1080i oznacza wyświetlanie z przeplotem) można się odnieść deklaracją `scan: interlace`.
- `grid` — ta własność pomaga ustalić, czy urządzenie działa na podstawie siatki punktów, czy bitmapy.

Wszystkie te opcje, z wyjątkiem `scan` i `grid`, można poprzedzić przedrostkami `min` i `max`, aby stworzyć deklaracje zakresów. W ramach przykładu rozważmy następujący fragment kodu:

```
@import url("tiny.css") screen and (min-width:200px) and (max-width:360px);
```

Jak widać, przedrostki minimum (`min`) i maksimum (`max`) zostały użyte z własnością `width`, tym samym został zdefiniowany zakres. Plik *tiny.css* zostanie odczytany tylko w przypadku urządzeń wyposażonych w ekrany z obszarem roboczym nie węższym niż 200 pikseli i nie szerszym niż 360 pikseli.

Elementy wycofywane w Media Queries CSS poziomu 4.

Warto wiedzieć, że w szkicu specyfikacji Media Queries poziomu 4. kilka własności oznaczono do usunięcia (*http://dev.w3.org/csswg/mediaqueries-4/#mf-deprecated*). Najważniejsze z nich to: `device-height`, `device-width` oraz `device-aspect-ratio`. Przeglądarki nadal będą je obsługiwać, ale zaleca się nie używać ich już w nowych arkuszach stylów.

Modyfikowanie projektu strony za pomocą zapytań medialnych

Arkusze stylów zostały zaprojektowane tak, aby style zadeklarowane niżej zastępowały style zdefiniowane wyżej (chyba że te znajdujące się wyżej dotyczą bardziej konkretnych fragmentów strony). Możemy bez trudu ustanowić na samym początku arkusza podstawowe style, które będą dotyczyć wszystkich wersji naszego projektu, a następnie nadpisać interesujące nas fragmenty za pomocą zapytań medialnych umieszczonych w dolnej części dokumentu. Przykładowo: najpierw możemy zdefiniować odnośniki czysto tekstowe (albo o zmniejszonym rozmiarze pisma) w nawigacji przeznaczonej dla urządzeń o wąskim obszarze roboczym, a następnie nadpisać te reguły zapytaniem medialnym, które powiększy elementy nawigacji i doda do nich ikony w urządzeniach o większym ekranie.

Zobaczmy, jak to może wyglądać w praktyce (przykład *02-02*). Najpierw spójrz na kod HTML:

```
<a href="#" class="CardLink CardLink_Hearts">Kier</a>
<a href="#" class="CardLink CardLink_Clubs">Trefl</a>
<a href="#" class="CardLink CardLink_Spades">Pik</a>
<a href="#" class="CardLink CardLink_Diamonds">Karo</a>
```

Teraz przyjrzyj się regułom CSS:

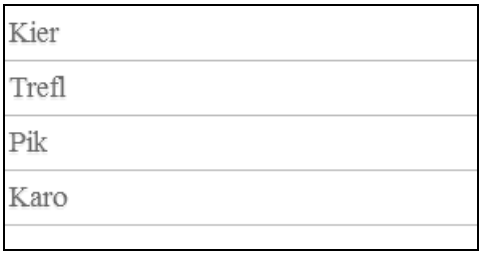
```
.CardLink {
    display: block;
    color: #666;
    text-shadow: 0 2px 0 #efefef;
    text-decoration: none;
    height: 2.75rem;
    line-height: 2.75rem;
    border-bottom: 1px solid #bbb;
    position: relative;
}
```

```
@media (min-width: 300px) {
    .CardLink {
        padding-left: 1.8rem;
        font-size: 1.6rem;
    }
}

.CardLink:before {
    display: none;
    position: absolute;
    top: 50%;
    transform: translateY(-50%);
    left: 0;
}

.CardLink_Hearts:before {
    content: " ";
}

.CardLink_Clubs:before {
    content: " ";
}

.CardLink_Spades:before {
    content: " ";
}

.CardLink_Diamonds:before {
    content: " ";
}

@media (min-width: 300px) {
    .CardLink:before {
        display: block;
    }
}
```

Pobieranie przykładów kodu

Wszystkie pliki z przykładami kodu źródłowego można pobrać z serwera FTP wydawnictwa Helion pod adresem: *ftp://ftp.helion.pl/przyklady/trash2.zip*.

Tak te odnośniki będą wyglądać w urządzeniu o niewielkiej szerokości (patrz rysunek na następnej stronie).



A taki wygląd przybiorą w szerszych urządzeniach:

W zapytaniu medialnym można wpisać każdą regułę CSS

Należy pamiętać, że w zapytaniu medialnym można wpisać wszystko, co wpisałoby się w zwykłym arkuszu stylów. Dzięki temu możliwe jest całkowite zmienienie układu i wyglądu strony w zależności od warunków jej wyświetlania (np. zmiennych rozmiarów obszaru roboczego).

Zapytania medialne dla urządzeń o dużej gęstości pikseli

Zapytań medialnych często używa się też do zmieniania stylów w przypadkach, gdy strona jest oglądana w urządzeniach o bardzo wysokiej rozdzielczości. Spójrz na poniższy kod:

```
@media (min-resolution: 2dppx) {
  /* Style. */
}
```

To zapytanie stanowi, że zawarte w nim reguły mają zostać zastosowane tylko w urządzeniach o rozdzielczości ekranu wynoszącej 2 plamki na piksel (2dppx). Dotyczy to iPhone'a 4 (urządzenia firmy Apple z ekranami o dużej gęstości pikseli nazywają się Retina) i wielu urządzeń z systemem Android. W razie potrzeby można zmniejszyć wartość `dppx`, aby zapytanie odnosiło się do większej liczby urządzeń.

Jeśli przy definiowaniu zapytania z minimalną rozdzielczością zależy Ci na obsłudze jak największej liczby urządzeń, dodaj przedrostki producentów za pomocą jakiegoś służącego do tego narzędzia. Nie przejmuj się, jeśli nie wiesz, co to są przedrostki producentów, ponieważ wkrótce wszystko wyjaśnię.

Metody organizacji i pisania zapytań medialnych

Na chwilę zmienimy temat i przyjrzymy się innym metodom pisania i organizacji zapytań medialnych w arkuszach stylów. Każda z nich ma pewne zalety i wady. Warto o nich wiedzieć, nawet jeśli teraz wydaje Ci się, że nie są potrzebne w Twoich projektach.

Dołączanie różnych plików CSS za pomocą zapytań medialnych

Dla przeglądarki internetowej arkusze stylów są zasobami „blokującymi renderowanie strony". Przeglądarka musi je pobrać z serwera i przeanalizować, aby móc wyświetlić stronę na ekranie.

Nowoczesne przeglądarki internetowe są przynajmniej wystarczająco „bystre", że rozróżniają, które style (dołączone w nagłówku przy użyciu zapytań medialnych) trzeba przetworzyć natychmiast, a które można odłożyć na później, gdy zostanie zakończona wstępna faza renderowania.

Przeglądarki mogą odłożyć na później wczytywanie stylów dołączonych do strony przy użyciu niemających zastosowania zapytań medialnych (np. jeśli ekran jest zbyt mały, by dane zapytanie zostało zastosowane) i wrócić do nich po zakończeniu wstępnego wczytywania strony. W ten sposób zyskuje się nieco na wydajności.

Więcej informacji na ten temat można znaleźć w portalu dla programistów Google na stronie: *https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-blocking-css*.

Chciałbym jednak zwrócić Twoją uwagę na następujący fragment tekstu:

> „Pamiętaj, że pojęcie zasób blokujący renderowanie odnosi się tylko do faktu, że przeglądarka wstrzymuje pierwsze renderowanie strony z powodu tego zasobu. Niezależnie od tego, czy następuje czy nie, przeglądarka zawsze pobiera zasób CSS, chociaż w przypadku zasobów nieblokujących z niższym priorytetem".

Powtórzmy zatem: wszystkie dołączone do strony pliki zostaną pobrane, ale przeglądarka nie musi wstrzymywać renderowania, jeśli niektóre z nich nie są natychmiast potrzebne.

Zatem nowoczesna przeglądarka wczytująca responsywną stronę internetową (zobacz przykład z folderu *02-03*) z czterema arkuszami stylów dołączonymi przy użyciu czterech różnych zapytań medialnych (zawierającymi różne style dla poszczególnych przedziałów rozmiarowych urządzeń) pobierze wszystkie te pliki, ale najprawdopodobniej najpierw przetworzy ten, który jest natychmiast potrzebny do renderowania strony.

Zasadność dzielenia zapytań medialnych na wiele plików

Choć, jak już wiesz, dzielenie zapytań medialnych ma pewne zalety, nie zawsze korzyści te są warte zachodu (oczywiście nie licząc preferencji projektanta w zakresie dzielenia kodu na osobne pliki).

Tak czy inaczej, każdy dodatkowy plik oznacza konieczność wysłania do serwera jednego żądania HTTP więcej, co w niektórych sytuacjach spowalnia proces ładowania strony. W internecie nic nigdy nie jest łatwe! Dlatego zanim zdecydujesz się na zastosowanie tej metody, zawsze dokładnie wszystko przemyśl i przetestuj wydajność swojej strony w różnych urządzeniach.

Sam postępuję w ten sposób, że jeśli nie mam nadmiaru czasu do wykorzystania, jest to ostatnia technika, za pomocą której próbuję przyspieszyć działanie swoich stron. Najpierw wykonuję następujące czynności:

- kompresuję wszystkie obrazy;
- łączę i minimalizuję wszystkie skrypty;
- kompresuję algorytmem gzip wszystkie serwowane zasoby;
- buforuję statyczną treść w sieciach CDN;
- usuwam wszystkie niepotrzebne reguły CSS.

Dopiero potem mogę rozważyć możliwość wydzielenia zapytań medialnych do osobnych plików w celu zyskania na wydajności.

> Algorytm gzip służy do kompresowania i dekompresowania plików. Każdy szanujący się serwer powinien umożliwiać kompresowanie za jego pomocą plików CSS, dzięki czemu z serwera do przeglądarki (w której następuje dekompresja) przesyłane są mniejsze ilości danych. Dobre wprowadzenie do gzip można znaleźć w Wikipedii: *https://pl.wikipedia.org/wiki/Gzip*.

Śródliniowe zagnieżdżanie zapytań medialnych

Prawie zawsze, nie licząc pewnych ekstremalnych przypadków, zalecam wstawianie zapytań medialnych do istniejących arkuszy stylów, obok „normalnych" reguł.

Jeśli Tobie również to odpowiada, musisz rozważyć jeszcze jedną kwestię: czy zapytania medialne powinno się definiować bezpośrednio pod związanymi z nimi selektorami, czy w całkiem osobnych blokach na końcu pliku? Cieszę się, że pytasz.

Łączyć zapytania medialne w bloki czy rozpraszać je w różnych miejscach pliku

Jestem zwolennikiem pisania zapytań medialnych bezpośrednio pod „normalnymi" regułami. Powiedzmy np., że chcemy modyfikować szerokość kilku elementów w kilku miejscach arkusza stylów w zależności od szerokości obszaru roboczego. Ja zastosowałbym takie rozwiązanie:

```
.thing {
    width: 50%;
}

@media screen and (min-width: 30rem) {
    .thing {
        width: 75%;
    }
}

/* Kilka innych reguł. */

.thing2 {
    width: 65%;
}
@media screen and (min-width: 30rem) {
    .thing2 {
        width: 75%;
    }
}
```

Na pierwszy rzut oka ten kod wygląda makabrycznie. Są w nim dwa zapytania medialne odnoszące się do minimalnej szerokości ekranu `30rem`. Przecież takie wielokrotne powtarzanie deklaracji `@media` to czyste marnotrawstwo. Czy nie powinienem więc zachęcać do grupowania zapytań medialnych w pojedyncze bloki, jak poniżej?

```
.thing {
    width: 50%;
}

.thing2 {
    width: 65%;
}
```

```
@media screen and (min-width: 30rem) {
    .thing {
        width: 75%;
    }
    .thing2 {
        width: 75%;
    }
}
```

Niewątpliwie tak też można, ale według mnie taka organizacja kodu utrudnia jego modyfikowanie i poprawianie. Oczywiście nie ma jedynego poprawnego sposobu zapisywania zapytań medialnych, ale ja preferuję wstawianie ich bezpośrednio pod regułami, których dotyczą, dzięki czemu wszystkie warianty danej reguły znajdują się w jednym miejscu. Gdy zechcę znaleźć deklarację odpowiadającą wybranej regule, nie muszę potem przeszukiwać całego arkusza stylów.

> Procesory CSS mają jeszcze wygodniejsze funkcje, ponieważ pozwalają na zagnieżdżanie zapytań medialnych w regułach. W książce *Sass i Compass. Praktyczny przewodnik dla projektantów* poświęciłem temu tematowi cały rozdział.

Jednym z logicznych argumentów przeciw preferowanej przeze mnie metodzie jest kwestia objętości kodu. Ale czy na pewno sam rozmiar pliku jest ważny do tego stopnia, by zniechęcić nas do organizowania arkuszy stylów w taki sposób? Wiadomo, że nikt nie chce wysyłać swoim użytkownikom rozdętych do gigantycznych rozmiarów plików. Z drugiej strony, kompresja gzip (kompresowane powinny być wszystkie zasoby, jakie da się skompresować) sprawia, że wzrost rozmiaru pliku CSS staje się nieistotny. Sprawdziłem to na wiele sposobów, więc jeśli ten temat szczególnie Cię interesuje, możesz przeczytać mój artykuł: *http://benfrain.com/inline-or--combined-media-queries-in-sass-fight/*. Ogólnie chodzi w nim o to, że moim zdaniem kwestia rozmiaru pliku w dyskusji o tym, jaką metodę definiowania zapytań medialnych stosować, jest nieistotna.

> Jeśli chcesz wpisywać zapytania medialne bezpośrednio pod regułami, których dotyczą, i jednocześnie wszystkie reguły związane z danym zapytaniem mieć w jednym bloku, to możesz skorzystać z narzędzi kompilacyjnych (w tej chwili Grunt i Gulp mają odpowiednie wtyczki).

Znacznik meta viewport

Jeśli chcesz maksymalnie wykorzystać możliwości zapytań medialnych, musisz sprawić, aby urządzenia z mniejszymi ekranami wyświetlały strony w odpowiednich dla siebie wymiarach (a nie np. prezentowały strony o szerokości 980 pikseli, które trzeba by było powiększać i pomniejszać).

Kiedy w 2007 r. firma Apple wyprodukowała iPhone'a, wprowadziła wraz z nim własny znacznik `meta viewport`, który jest już obsługiwany przez system Android i wiele innych platform. Znacznik ten służy do sygnalizowania przeglądarkom urządzeń przenośnych przez strony internetowe, w jaki sposób należy je renderować.

W przyszłości każda responsywna strona, która ma dobrze prezentować się także na małych ekranach, będzie musiała mieć zdefiniowany ten znacznik.

Testowanie projektów responsywnych w emulatorach

Choć nic nie zastąpi testów witryn na właściwych urządzeniach, warto skorzystać z emulatorów dla systemów Android i iOS.

Gwoli ścisłości: symulator tylko symuluje dane urządzenie, podczas gdy emulator naprawdę próbuje interpretować pierwotny kod urządzenia.

Emulator Androida dla systemów Windows, Linux i Mac jest darmowy — wystarczy ściągnąć i zainstalować zestaw narzędzi Android **Software Development Kit (SDK)** z witryny: *http://developer.android.com/sdk/*.

Symulator iOS dostępny jest tylko dla użytkowników systemu Mac OS X — jest częścią pakietu Xcode (dostępnego za darmo w Mac App Store).

Także same przeglądarki internetowe zawierają coraz to lepsze emulatory różnych urządzeń. Zarówno Firefox, jak i Chrome posiadają specjalne ustawienia pozwalające emulować wiele różnych urządzeń przenośnych i obszarów roboczych.

Znacznik `meta viewport` wstawia się w elemencie `<head>` dokumentu HTML. Można go ustawić na konkretną szerokość (wyrażoną np. w pikselach) lub na współczynnik skalowania, np. `2.0` (dwukrotność rzeczywistego rozmiaru). Poniżej znajduje się przykładowa definicja z ustawioną skalą na 200% (`2.0`):

```
<meta name="viewport" content="initial-scale=2.0,width=device-width" />
```

Przeanalizujmy składnię tego znacznika, aby dokładnie zrozumieć zasadę jego działania. Łatwo rozszyfrować znaczenie atrybutu `name="viewport"`. Atrybut `content="initial-scale=2.0"` powoduje dwukrotne powiększenie treści na stronie (wartość `0.5` spowodowałaby dwukrotne zmniejszenie, a `3.0` trzykrotne zwiększenie itd.). Natomiast fragment `width=device-width` mówi przeglądarce, że szerokość strony powinna być równa szerokości urządzenia.

Za pomocą znacznika `meta` można też kontrolować maksymalny stopień powiększania i zmniejszania strony przez użytkownika. Poniższy prosty przykład umożliwia użytkownikom powiększenie obrazu na szerokość trzykrotnie większą niż szerokość ekranu i zmniejszenie go na szerokość dwukrotnie mniejszą od szerokości ekranu.

```
<meta name="viewport" content="width=device-width, maximum-scale=3, minimum-scale=0.5" />
```

Istnieje też możliwość wyłączenia funkcji zmniejszania i powiększania (choć trzeba pamiętać, że jest ona bardzo ważnym mechanizmem dostępności i dlatego w praktyce rzadko kiedy zaleca się ten krok):

```
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
```

Tutaj najważniejszy jest fragment `user scalable=no`.

Zmienimy wartość współczynnika skalowania na `1.0`, dzięki czemu przeglądarka dopasuje wielkość strony do 100% wielkości obszaru roboczego. Ustawienie tego parametru na szerokość urządzenia sprawia, że strona będzie renderowana na całej szerokości obsługujących te ustawienia przeglądarek. W większości przypadków definicja tego znacznika powinna wyglądać następująco:

```
<meta name="viewport" content="width=device-width,initial-scale=1.0" />
```

> Konsorcjum W3C zauważyło, że znacznik `meta viewport` jest w coraz powszechniejszym użyciu, więc stara się przenieść jego funkcjonalność do CSS. Na stronie: *http://dev.w3.org/csswg/css-device-adapt/* znajdziesz informacje o deklaracji `@viewport`. Chodzi o to, by zamiast wprowadzać oddzielny znacznik `<meta>` w sekcji `<head>` witryny, można było deklarować wyrażenia typu `@viewport { width: 320px; }` w arkuszu stylów. Deklaracja ta ustawiłaby szerokość przeglądarki na 320 pikseli. Niestety, przeglądarki internetowe słabo obsługują tę technikę, więc dla pewności można się posługiwać kombinacją znacznika `meta` i deklaracji `@viewport`.

Myślę, że już dość dobrze orientujesz się, do czego służą i jak działają zapytania medialne. Zanim przejdziemy do całkiem nowego tematu, chciałbym jeszcze zrobić przegląd tego, co szykuje nam przyszłość w następnej wersji specyfikacji. Zobaczmy!

Zapytania medialne — poziom 4.

W czasie, gdy piszę tę książkę, specyfikacja CSS Media Queries Level 4 jest na etapie szkicu (*http://dev.w3.org/csswg/mediaqueries-4/*) i opisane w niej nowości nie są zaimplementowane w wielu przeglądarkach. Dlatego też, choć poniżej robię przegląd najważniejszych elementów tej specyfikacji, należy pamiętać, że wiele jeszcze może się zmienić. Zawsze upewnij się, czy to, czego chcesz użyć, jest już obsługiwane, i sprawdź, czy nie nastąpiły jakieś zmiany w składni.

Omawiana specyfikacja zawiera wiele nowości, ale ja skupiam się tylko na składnikach dotyczących skryptów, urządzeń wskazujących, kwestii wskazywania elementów kursorem oraz światła otoczenia.

Obsługa skryptów

W dokumentach HTML często stosuje się technikę polegającą na domyślnym przypisaniu elementu do klasy oznaczającej, że nie ma żadnego skryptu JavaScript, a następnie zmianie tej klasy na inną, gdy zostanie uruchomiony skrypt. Jest to prosty sposób na rozgałęzienie kodu (wliczając CSS) na podstawie tej nowej klasy HTML. Technika ta służy przede wszystkich do pisania reguł przeznaczonych dla użytkowników, których przeglądarka ma wyłączoną obsługę JavaScript.

Nie wiem, czy powyższy akapit jest zrozumiały, więc dodam prosty przykład. Domyślnie w dokumencie HTML znajdowałby się taki element:

```
<html class="no-js">
```

Gdy na tej stronie zostanie uruchomiony skrypt JavaScript, jego pierwszym zadaniem będzie zamiana klasy `no-js` na `js`:

```
<html class="js">
```

Teraz możemy napisać reguły CSS przeznaczone do zastosowania tylko wtedy, gdy działają skrypty JavaScript, np. `.js .header {display: block;}`.

W nowej specyfikacji zapytań medialnych podjęto próbę ustandaryzowania tej metody tak, aby test obsługi JavaScript można było wykonywać wprost z poziomu CSS:

```
@media (scripting: none) {
  /* Reguły do zastosowania, gdy brakuje obsługi JavaScript. */
}
```

Można też sprawdzać, czy JavaScript jest włączony:

```
@media (scripting: enabled) {
  /* Reguły do zastosowania, gdy JavaScript jest włączony. */
}
```

Istnieje też możliwość sprawdzenia, czy obsługa JavaScript jest włączona tylko początkowo. W specyfikacji W3C podano przykład drukowanej strony, której układ jest najpierw budowany z użyciem skryptów, ale później JavaScript staje się niedostępny. Ten test zapisuje się następująco:

```
@media (scripting: initial-only) {
  /* Reguły do zastosowania, gdy JavaScript działa tylko początkowo. */
}
```

Szerszy opis wstępnej wersji tej funkcji można znaleźć pod adresem: *https://drafts.csswg.org/mediaqueries-4/#mf-scripting*.

Urządzenia wskazujące

Na stronach W3C znajduje się następujący opis funkcji dotyczącej urządzeń wskazujących:

> „Funkcja ta służy do sprawdzania obecności i dokładności urządzeń wskazujących, takich jak mysz. Jeżeli urządzenie dysponuje kilkoma mechanizmami wskazywania, funkcja musi odzwierciedlać właściwości podstawowego z nich zgodnie z ustaleniami klienta użytkownika".

Urządzenie wskazujące może mieć jedną z trzech właściwości: `none`, `coarse` i `fine`.

Urządzeniem wskazującym o właściwości `coarse` (gruby) może być palec w przypadku ekranu dotykowego albo kursor z konsoli do gier, w której nie ma tak precyzyjnego wskaźnika jak mysz.

```
@media (pointer: coarse) {
  /* Reguły do zastosowania dla niezbyt precyzyjnego wskaźnika. */
}
```

Urządzenia o właściwości `fine` (precyzyjny) to np.: mysz, rysik lub jakiś inny wynalazek do precyzyjnego wskazywania elementów na ekranie.

```
@media (pointer: fine) {
  /* Style do zastosowania dla precyzyjnych wskaźników. */
}
```

Moim zdaniem im szybciej przeglądarki zaczną obsługiwać te własności, tym lepiej. Obecnie wszyscy mają problemy ze stwierdzeniem, czy użytkownik dysponuje myszą, ekranem dotykowym, czy jednym i drugim, oraz z czego aktualnie korzysta.

> Najbezpieczniej jest zakładać, że użytkownik korzysta z ekranu dotykowego i zgodnie z tym założeniem ustawiać odpowiednie rozmiary elementów. Dzięki temu, nawet jeśli ktoś używa myszy, bez problemu będzie mógł się posługiwać naszym interfejsem. Gdybyśmy natomiast przyjęli, że użytkownik używa myszy, a ten posługiwałby się ekranem dotykowym, to korzystanie z naszej strony byłoby dla niego utrudnione.
>
> Jeśli chcesz się dowiedzieć, jak wielkie trudności trzeba przezwyciężyć przy projektowaniu stron dla urządzeń z myszą i ekranem dotykowym, polecam zestaw slajdów pt. *Getting touchy* Patricka H. Lauke'a: *https://patrick ↪hlauke.github.io/getting-touchy-presentation/*.

Szerszy opis wstępnej wersji tej funkcji można znaleźć pod adresem: *http://dev.w3.org/csswg/ mediaqueries-4/#mf-interaction*.

Funkcja sprawdzania obsługi efektu hover

Nietrudno zgadnąć, że funkcja sprawdzania obsługi efektu hover dotyczy testowania, czy użytkownik jest w stanie umieścić kursor nad elementem na stronie. Jeśli obecnych jest kilka urządzeń wejściowych (np. ekran dotykowy i mysz), pod uwagę brane są właściwości tego podstawowego. Poniżej przedstawiam możliwe wartości i przykłady kodu.

Dla użytkowników bez możliwości umieszczania kursora nad elementami możemy pisać reguły z wartością `none`:

```
@media (hover: none) {
  /* Style do zastosowania, gdy użytkownik nie może umieszczać kursora nad elementami. */
}
```

Dla użytkowników, którzy mogą posługiwać się kursorem, ale muszą w tym celu wykonać jakąś znaczącą czynność, należy napisać deklarację z wartością `on-demand`:

```
@media (hover: on-demand) {
  /* Style do zastosowania, gdy użytkownik może używać kursora, ale wymaga to od niego
  /* konkretnego wysiłku. */
}
```

Aby wyznaczyć reguły dla użytkowników mogących bez żadnych przeszkód posługiwać się kursorem, wystarczy użyć wartości `hover`.

```
@media (hover) {
  /* Style do zastosowania, gdy użytkownik może bez przeszkód posługiwać się kursorem. */
}
```

Istnieją jeszcze opcje `any-pointer` i `any-hover`. Działają podobnie jak opisane powyżej ustawienia `pointer` i `hover`, z tym że sprawdzają, czy daną możliwość ma którekolwiek z dostępnych urządzeń.

Zapytania dotyczące otoczenia

Czy nie byłoby przyjemnie, gdybyśmy mogli zmieniać nasze projekty na podstawie warunków panujących w otoczeniu użytkownika, np. natężenia światła? Dzięki temu, gdyby użytkownik znajdował się w ciemnym pokoju, moglibyśmy zastosować nieco przytłumione kolory. Albo odwrotnie: gdyby oświetlenie otoczenia miało duże natężenie, moglibyśmy zwiększać kontrast kolorów. Podjęto już próby rozwiązania tego typu problemów za pomocą zapytań medialnych. Spójrz na poniższe przykłady:

```
@media (light-level: normal) {
    /* Style dla standardowych warunków oświetlenia. */
}
@media (light-level: dim) {
    /* Style dla słabego światła otoczenia. */
}
@media (light-level: washed) {
    /* Style dla jasnego światła otoczenia. */
}
```

Warto wiedzieć, że jest kilka implementacji tych zapytań medialnych. Poza tym można się spodziewać zmian w specyfikacji, zanim będzie można bezpiecznie z niej korzystać. Jednak już teraz dobrze jest się orientować, jakie zmiany mogą nastąpić w ciągu kilku najbliższych lat.

Szerszy opis wstępnej wersji tej funkcji można znaleźć pod adresem: *http://dev.w3.org/ csswg/ mediaqueries-4/#mf-environment*.

Podsumowanie

W tym rozdziale dowiedziałeś się, czym są zapytania medialne CSS3, jak je zadeklarować w plikach CSS i jak mogą Ci one pomóc w stworzeniu projektu RWD witryny. Dowiedziałeś się też, jak za pomocą znacznika `meta` zmusić nowoczesne przeglądarki przenośne, aby wczytywały strony w taki sposób, w jaki chcesz.

Ostatecznie przekonałeś się jednak, że za pomocą samych zapytań medialnych można tworzyć projekty adaptacyjne, czyli takie, które zmieniają skokowo układ w zależności od warunków. Prawdziwie responsywna strona gładko przechodzi od jednego układu do innego. Do osiągnięcia naszego celu potrzebna będzie więc też znajomość układów płynnych, które mogą się elastycznie zmieniać między kolejnymi punktami kontrolnymi. Tworzenie układów płynnych ułatwiających harmonijne przejście między stanami narzucanymi przez nasze zapytania medialne jest tematem kolejnego rozdziału.

Skorowidz

F

G

H

I

Ś

T

U

V

W

Z

Ż

Responsive Web Design. Projektowanie elastycznych witryn w HTML5 i CSS3. Wydanie II

Profesjonalnie wykonana strona internetowa powinna działać bez zarzutu nie tylko na komputerze stacjonarnym, lecz także na tablecie czy telefonie. Użytkownicy chcą korzystać z aplikacji sieciowych w każdych warunkach i uruchamiać je na urządzeniach o najprzeróżniejszych rozmiarach czy parametrach. Należy się spodziewać, że wkrótce katalog urządzeń podłączanych do internetu znacznie się poszerzy. W takich warunkach projektant koniecznie musi zadbać o responsywność, aby tworzone przez niego strony internetowe dostosowywały się do zmiennych warunków i zachowywały funkcjonalność.

Niniejsza książka stanowi kompletne źródło informacji potrzebnych do napisania responsywnej strony internetowej. Jeśli znasz HTML i CSS, możesz z pomocą tego podręcznika zbudować taką aplikację. Znajdziesz tu opis wszystkich podstawowych aspektów responsywnych projektów stron i dowiesz się, jak korzystać z najbardziej przydatnych technik w technologiach HTML5 i CSS3. Co więcej, odkryjesz najlepsze metody pisania i dostarczania kodu, obrazów i plików. Dzięki licznym przykładom i opisom bez trudu dostosujesz swój projekt do wymagań telefonów komórkowych i wielu innych urządzeń. Wydanie drugie uzupełniono o opis prawie wszystkich najnowszych technik i narzędzi potrzebnych do budowy responsywnych aplikacji internetowych.

Responsywne strony internetowe — nawet po latach będą równie piękne.

W tej książce znajdziesz:

- opis elementów potrzebnych do zbudowania responsywnych stron internetowych
- informacje o zapytaniach medialnych, ich składni i sposobach wykorzystania
- omówienie projektowania struktury i układów płynnych oraz wykorzystania modelu Flexbox
- wskazówki co do wykorzystania niezwykłych możliwości CSS3 i HTML5
- spis dobrych praktyk kodowania responsywnych stron internetowych

Ben Frain — projektant stron internetowych, starszy programista frontendowy w firmie Bet365. Kiedyś był niedocenionym (i skromnym) aktorem telewizyjnym. Napisał cztery równie niedocenione (jego zdaniem) scenariusze i wciąż żywi (słabnącą) nadzieję, że w końcu uda mu się sprzedać choćby jeden z nich. W oczekiwaniu na ten dzień pisze książki o tajnikach projektowania stron WWW. Warto przeczytać również jego drugą książkę: *Sass i Compass. Praktyczny przewodnik dla projektantów.*

[PACKT] PUBLISHING

Helion

44540 numer katalogowy

księgarnia internetowa
http://helion.pl

zamówienia telefoniczne
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-283-2343-8

Informatyka w najlepszym wydaniu

cena: 49,00 zł