O'REILLY®

Wydanie IV

CSS
Kaskadowe arkusze stylów

PRZEWODNIK ENCYKLOPEDYCZNY

Helion

Eric A. Meyer, Estelle Weyl

Tytuł oryginału: CSS: The Definitive Guide: Visual Presentation for the Web, 4th Edition

Tłumaczenie: Piotr Cieślak

ISBN: 978-83-283-4083-1

Helion SA
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/kasty4.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/kasty4
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 17.

Przejścia

Przejścia CSS umożliwiają animowanie właściwości CSS — czyli płynną zmianę od wartości początkowej do nowej wartości docelowej z upływem czasu. Powodują one zmianę jednego stanu elementu na drugi w reakcji na jakąś zmianę — zwykle wynikającą z działania użytkownika, ale także z zaprogramowanych w skrypcie zmian klas, identyfikatorów albo innego rodzaju stanów.

W normalnej sytuacji każda zmiana wartości właściwości CSS — chwila, gdy nastąpi „zdarzenie zmiany stylów" — zachodzi natychmiastowo. Nowa wartość właściwości zastępuje starą w ciągu kilku milisekund, jakie zajmuje ponowne narysowanie treści, której dotyczy ta zmiana (albo narysowanie i zmiana układu dokumentu, jeśli zajdzie taka konieczność). Większość zmian wartości wydaje się natychmiastowa, bo ich wyświetlenie trwa poniżej 16 milisekund[1]. Nawet jeśli zmiana trwa dłużej, wciąż jest to tylko jeden krok — przejście od jednej wartości do drugiej. Na przykład zmiana koloru tła po wskazaniu danego elementu kursorem myszy jest bezzwłoczna, bez płynnego przejścia między barwami.

Przejścia CSS

Przejścia (ang. *transitions*) CSS umożliwiają sterowanie sposobem zmiany jednej wartości właściwości na inną w ciągu określonego czasu. Dzięki temu możemy uzyskać płynną zmianę wartości, dającą przyjemny dla oka i (miejmy nadzieję) nieprzeszkadzający w odbiorze treści efekt. Weźmy następujący przykład:

```
button {
    color: magenta;
    transition: color 200ms ease-in 50ms;
}

button:hover {
    color: rebeccapurple;
    transition: color 200ms ease-out 50ms;
}
```

W tym przykładzie, zamiast natychmiastowej zmiany wartości `color` po wskazaniu przycisku kursorem myszy, za pomocą przejść CSS możemy uzyskać płynną zmianę koloru z `magenta` na `rebeccapurple` w ciągu 200 milisekund, z trwającym 50 milisekund opóźnieniem przed rozpoczęciem przejścia.

[1] Zmiana obrazu w tle, wymagająca zdekodowania tego obrazu i wyświetlenia go, może trwać dłużej niż 16 milisekund. To nie jest jednak kwestia samego przejścia, ale niewielkiej wydajności.

Zmiana koloru, bez względu na to, jak długo albo jak krótko trwa, jest pewną formą przejścia. Jednak dzięki zastosowaniu właściwości CSS o nazwie `transition` zmiana koloru może nastąpić stopniowo, w ciągu pewnego czasu, a jej płynność będzie łatwo dostrzegalna dla oka.

Przejściami CSS możesz się posłużyć już dziś, nawet jeśli wciąż dbasz o użytkowników przeglądarek IE9 albo starszych. Jeśli jakaś przeglądarka nie obsługuje właściwości przejść CSS, to zmiana po prostu nastąpi w jednej chwili, a nie płynnie, co jest zupełnie akceptowalne. Poza tym do raptownej (a nie płynnej) zmiany dojdzie także wtedy, gdy jakaś właściwość albo pewne jej wartości nie dają się animować.

Mamy tu na myśli dowolne właściwości, które dają się animować — czy to za pośrednictwem przejść, czy też animacji (będących tematem następnego rozdziału, „Animacje"). Podsumowanie informacji na ten temat znajdziesz w dodatku A.

Czasami zależy nam na natychmiastowej zmianie jakiejś wartości. Choć w poprzedniej części rozdziału posłużyliśmy się przykładem odwołującym się do kolorów odsyłaczy, to kolory te na ogół zmieniają się w sposób natychmiastowy po wskazaniu odsyłacza kursorem myszy, informując osoby widzące o możliwości interakcji oraz o tym, że wskazany fragment treści jest hiperłączem. Na podobnej zasadzie opcje na liście umożliwiającej automatyczne uzupełnianie wpisywanej treści nie powinny się pojawiać stopniowo: chodzi raczej o to, by były wyświetlane od razu, zamiast pojawiać się wolniej, niż pisze użytkownik. Z punktu widzenia wygody użytkownika natychmiastowa zmiana wartości często jest najlepsza.

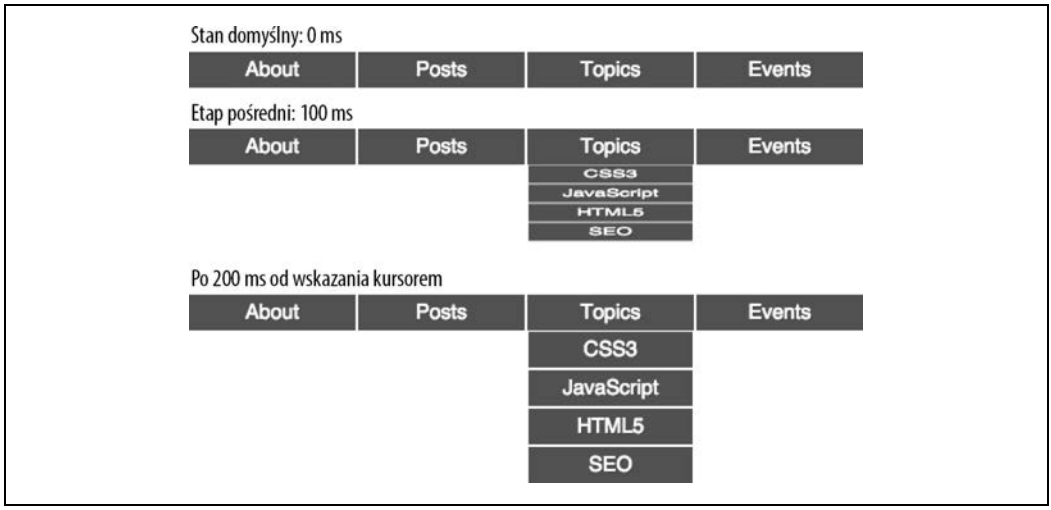
Kiedy indziej z kolei może nam zależeć na tym, by zmiana wartości właściwości następowała bardziej stopniowo i zwracała uwagę na to, co się dzieje. Możemy na przykład nadać grze w karty pozory większego realizmu, wprowadzając animację odwracania karty trwającą 200 milisekund, bo w przypadku braku animacji użytkownik może nawet nie zauważyć, co się zmieniło. ▶

Zwracaj uwagę na symbol odtwarzania ▶, który informuje o możliwości obejrzenia przykładu online. Wszystkie przykłady opisane w tym rozdziale są dostępne pod adresem *https://meyerweb.github.io/csstdg4fi gs/17-transitions/*.

Weźmy inny przykład: może zależeć Ci na tym, by niektóre rozwijane menu pojawiały się albo rozwijały w ciągu 200 milisekund (a nie natychmiast, co może być irytujące). Dzięki przejściom rozwijane menu mogą się wyświetlać powoli. Na rysunku 17.1 ▶ pokazane są etapy przejścia polegającego na skalowaniu wysokości jednego z podmenu. Jest to jedno z typowych zastosowań przejść CSS, które omówimy w dalszej części tego rozdziału.

Właściwości przejść

W CSS przejścia tworzy się za pomocą czterech właściwości przejść: `transition-property`, `transition-duration`, `transition-timing-function` oraz `transition-delay`; istnieje też skrótowa właściwość `transition` obejmująca te cztery właściwości składowe.



Rysunek 17.1. Przejście — etap początkowy, pośredni i końcowy

Do utworzenia rozwijanej nawigacji pokazanej na rysunku 17.1 użyliśmy wszystkich czterech właściwości przejść CSS, a oprócz tego także właściwości niezwiązanych z przejściami, definiujących początkowy i końcowy stan przejścia. Przejście przedstawione na rysunku 17.1 można zdefiniować za pomocą poniższego kodu:

```
nav li ul {
    transition-property: transform;
    transition-duration: 200ms;
    transition-timing-function: ease-in;
    transition-delay: 50ms;
    transform: scale(1, 0);
    transform-origin: top center;
}
nav li:hover ul {
    transform: scale(1, 1);
}
```

Zauważ, że choć w naszych przykładach przejść używamy stanu `:hover` jako zdarzenia wyzwalającego zmianę stylów, to przejścia z użyciem właściwości można stosować także w innych sytuacjach. Możesz na przykład dodać albo usunąć klasę bądź w inny sposób zmienić stan elementu — na przykład poprzez zmianę pola formularza z `:invalid` na `:valid` albo z `:checked` na `:not(:checked)`. Możesz też przy użyciu selektorów `:nth-last-of-type` dodać wiersz do tabeli z naprzemiennie pokolorowanymi wierszami („zebra") albo dołożyć kolejny punkt na końcu jakiejś listy.

W scenariuszu pokazanym na rysunku 17.1 początkowy stan zagnieżdżonych list to `transform: scale(1, 0)` i `transform-origin: top center`. Stan końcowy to `transform: scale(1, 1)` — właściwość `transform-origin` pozostaje bez zmian.

Więcej informacji o właściwościach przekształceń (`transform`) znajdziesz w rozdziale 16.

W tym przykładzie przejście jest zdefiniowane przez właściwość `transform`: po wskazaniu menu kursorem myszy (`:hover`) zagnieżdżona, nienumerowana lista skaluje się do pierwotnego, domyślnego rozmiaru — w ciągu 200 milisekund następuje płynne przejście między starą wartością `transform: scale(1, 0)` a nową wartością `transform: scale(1, 1)`. Przejście rozpoczyna się z 50-milisekundowym opóźnieniem i przyspiesza (odpowiada za to funkcja `ease-in`) — zaczyna się wolno, a potem nabiera tempa.

Przejścia są deklarowane oprócz zwykłych stylów dla elementu. Ilekroć zmienia się właściwość, dla której zdefiniowano przejście, przeglądarka zastosuje to przejście, aby zmiana właściwości odbyła się stopniowo.

Zauważ, że wszystkie właściwości przejść zostały ustalone dla nieaktywnego stanu elementów `ul`. Stan aktywny (po wskazaniu kursorem myszy) został użyty tylko w celu zastosowania przekształcenia, a nie samych parametrów przejścia. I nie bez powodu: dzięki temu bowiem menu nie tylko płynnie otworzy się po wskazaniu kursorem, ale też płynnie zamknie, gdy przestaniemy je wskazywać.

Wyobraźmy sobie, że właściwości przejścia byłyby zamiast tego stosowane za pośrednictwem stanu `:hover`, na przykład tak:

```
nav li ul {
    transform: scale(1, 0);
    transform-origin: top center;
}
nav li:hover ul {
    transition-property: transform;
    transition-duration: 200ms;
    transition-timing-function: ease-in;
    transition-delay: 50ms;
    transform: scale(1, 1);
}
```

Oznaczałoby to, że element, który *nie jest* wskazany kursorem myszy, będzie miał domyślne właściwości przejść — czyli przejścia będą się odbywały natychmiastowo. Menu z poprzedniego przykładu płynnie otwierałoby się po wskazaniu kursorem myszy, ale po zakończeniu stanu `:hover` natychmiast by znikało — bo bez stanu `:hover` właściwości przejść byłyby nieaktywne!

Może się okazać, że właśnie na takim efekcie Ci zależy: płynne wysunięcie elementu, a potem raptowne schowanie go. Jeśli tak, to przypisz przejścia do stanu `:hover`. W przeciwnym razie przypisz je bezpośrednio do elementu, aby miały one zastosowanie zarówno przy rozpoczęciu, jak i przy kończeniu wskazywania elementu kursorem myszy. Zakończenie danego stanu powoduje odwrócenie kierunku przejścia. To domyślne zachowanie możesz zmienić, deklarując różne rodzaje przejść dla stanu początkowego i stanu zmienionego.

Przez „stan początkowy” rozumiemy stan elementu po załadowaniu strony. Może to być stan, który jest dla danego elementu trwały — na przykład określony przez właściwości zdefiniowane za pośrednictwem zwykłego selektora elementu, a nie za pośrednictwem selektora `:hover` dla tego elementu. Może to także być element o edytowalnej treści, który zyskuje stan `:focus`, jak w następującym przykładzie:

```
/* selektor, który zawsze pasuje do danych elementów */
p[contenteditable] {
    background-color: rgba(0, 0, 0, 0);
}
```

```
/* selektor, który pasuje do danych elementów tylko czasami */
p[contenteditable]:focus {
    /* zastępowanie deklaracji */
    background-color: rgba(0, 0, 0, 0.1);
}
```

W tym przykładzie stanem początkowym elementu jest całkowicie przezroczyste tło, które zmienia się tylko wtedy, gdy użytkownik uaktywni (stan `:focus`) ten element. Właśnie to mamy na myśli, mówiąc w tym rozdziale o stanie *początkowym* albo *domyślnym* elementu. Właściwości przejścia podane w selektorze, który pasuje do elementu przez cały czas, będą miały wpływ na ten element przy każdej zmianie jego stanu — na przykład zmianie ze stanu początkowego na docelowy (w poprzednim przykładzie docelowym był stan `:focus`).

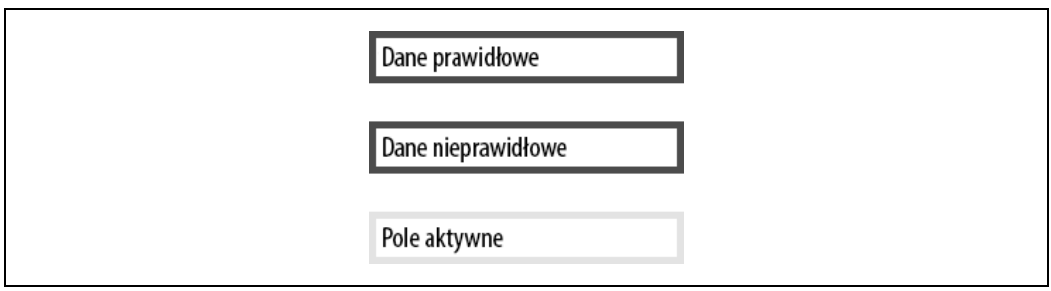
Stan początkowy może też mieć charakter przejściowy — na przykład `:checked` dla pola wyboru albo `:valid` dla kontrolki formularza. Może to być nawet klasa, którą da się włączać i wyłączać.

```
/* selektor, który pasuje do danych elementów tylko czasami */
input:valid {
    border-color: green;
}

/* selektor, który pasuje do danych elementów tylko czasami,
   a konkretnie wtedy, gdy poprzedni selektor NIE pasuje */
input:invalid {
    border-color: red;
}

/* selektor, który pasuje do danych elementów tylko czasami,
   w zależności od prawidłowości danych */
input:focus {
    /* alternatywna deklaracja */
    border-color: yellow;
}
```

W podanym przykładzie do dowolnego elementu może pasować albo selektor `:valid`, albo `:invalid`, ale nie oba naraz. Selektor `:focus`, zgodnie z tym, co zostało pokazane na rysunku 17.2, wybiera element wtedy, gdy pole wprowadzania danych jest aktywne, bez względu na to, czy pole to jest też jednocześnie wybrane przez selektory `:valid` albo `:invalid`.



Rysunek 17.2. Wygląd pola wprowadzania danych w różnych stanach — dane prawidłowe, dane nieprawidłowe, pole aktywne

W takim przypadku, jeśli odwołujemy się do stanu początkowego, mamy na myśli wartość początkową, którą może być zarówno `:valid`, jak i `:invalid`. Zmieniony stan danego elementu jest przeciwieństwem początkowego stanu `:valid` lub `:invalid`.

Należy pamiętać, że da się zastosować różne wartości przejść dla stanu początkowego i zmienionego, zawsze jednak należy zadbać o wprowadzenie wartości związanej z *wejściem* w dany stan. Weźmy na przykład poniższy kod, w którym przejścia zostały skonfigurowane tak, by menu otwierały się przez 2 sekundy, ale zamykały już po 200 milisekundach.

```
nav li ul {
    transition-property: transform;
    transition-duration: 200ms;
    transition-timing-function: ease-in;
    transition-delay: 50ms;
    transform: scale(1, 0);
    transform-origin: top center;
}
nav li:hover ul {
    transition-property: transform;
    transition-duration: 2s;
    transition-timing-function: linear;
    transition-delay: 1s;
    transform: scale(1, 1);
}
```

Z perspektywy funkcjonalności efekt jest straszny, ale dobrze ilustruje omawianą ideę.  Po wskazaniu menu kursorem myszy proces otwierania się tego menu zajmuje bite 2 sekundy. Zamykanie przebiega znacznie szybciej, bo w 0,2 sekundy. Właściwości przejścia w zmienionym stanie (`:hover`) dochodzą do głosu wtedy, gdy wskazujemy pozycję listy kursorem myszy. Wtedy zaczyna działać reguła `transition-duration: 2s`, zdefiniowana dla stanu `:hover`. Po odsunięciu kursora menu wraca do domyślnego, zwiniętego stanu i wykorzystywane są właściwości przejścia zdefiniowane dla stanu początkowego (opisanego regułą `nav li ul`), które powodują trwające 200 milisekund zamknięcie menu.

Przyjrzyjmy się nieco bliżej temu przykładowi, zwłaszcza domyślnym stylom przejść. Kiedy użytkownik przestaje wskazywać kursorem myszy nadrzędny element głównej nawigacji albo element potomny w postaci rozwijanego menu, następuje 50-milisekundowa pauza, poprzedzająca trwające 200 milisekund przejście, które zamyka menu. Akurat ten aspekt funkcjonalności menu jest wygodny, bo daje użytkownikowi szansę (choć krótką) na przesunięcie kursora z powrotem nad menu, zanim zacznie się ono zamykać.

Choć każdą z czterech właściwości przejść można zadeklarować osobno, to zapewne zawsze będziesz się posługiwał skrótem. Najpierw przyjrzymy się czterem właściwościom składowym, abyś mógł lepiej zrozumieć działanie każdej z nich.

Ograniczanie rodzaju przejść do konkretnych właściwości

Właściwość `transition-property` pozwala określić nazwy właściwości CSS, które chcesz poddać działaniu przejść. Umożliwia to ograniczenie rodzaju przejść do konkretnych właściwości, podczas gdy inne będą się zmieniały w sposób natychmiastowy. (I owszem, w dalszej części rozdziału słowo „właściwości” będzie się pojawiało bardzo wiele razy).

transition-property

Wartości:	`none` | [`all` | *`<property-name>`*]#
Wartość początkowa:	`all`
Stosuje się do:	Wszystkich elementów oraz pseudoelementów `:before` i `:after`
Wartość wyliczona:	Jak określono
Dziedziczona:	Nie
Animowana:	Nie

Właściwość `transition-property` przyjmuje wartości w postaci listy rozdzielonych przecinkami właściwości, słowa kluczowego `none` (jeśli nie chcesz dopuścić do przejść jakichkolwiek właściwości) albo domyślnego słowa `all` (które oznacza: „dopuść przejścia wszystkich animowanych właściwości"). Słowo `all` może też wystąpić na liście oprócz właściwości rozdzielonych przecinkami.

Jeśli podasz `all` jako jedyne słowo kluczowe — albo pozwolisz, by właściwość `transition-property` domyślnie przyjęła wartość `all` — wszystkie animowane właściwości zostaną poddane równoczesnemu przejściu. Powiedzmy, że chcesz zmienić wygląd jakiegoś pojemnika po wskazaniu go kursorem myszy:

```
div {
    color: #ff0000;
    border: 1px solid #00ff00;
    border-radius: 0;
    transform: scale(1) rotate(0deg);
    opacity: 1;
    box-shadow: 3px 3px rgba(0, 0, 0, 0.1);
    width: 50px;
    padding: 100px;
}
div:hover {
    color: #000000;
    border: 5px dashed #000000;
    border-radius: 50%;
    transform: scale(2) rotate(-10deg);
    opacity: 0.5;
    box-shadow: -3px -3px rgba(255, 0, 0, 0.5);
    width: 100px;
    padding: 20px;
}
```

Po wskazaniu kursorem elementu `div` wszystkie właściwości, które mają inne wartości w stanie początkowym niż w stanie `:hover` (zmienionym), ulegną zmianie i przyjmą wartości zgodne ze stanem `:hover`. Właściwość `transition-property` służy do określania, które z tych właściwości będą płynnie animowane przez pewien czas (zamiast raptownie się zmienić). Po wskazaniu elementu kursorem myszy wszystkie właściwości zmienią się z wartości domyślnych na wartości `:hover`, ale tylko animowane właściwości zadeklarowane we właściwości `transition-property` zmienią się w sposób płynny, w ciągu całego czasu trwania przejścia. Niedające się animować właściwości, takie jak `border-style`, zmienią swoją wartość w sposób natychmiastowy.

Jeśli jedyną (albo ostatnią na liście wartości rozdzielonych przecinkami) wartością właściwości `transition-property` jest `all`, to wszystkie animowane właściwości dokonają równoczesnego przejścia. Aby uzyskać inny efekt, należy podać rozdzieloną przecinkami listę właściwości, które mają podlegać wpływowi przejść.

Jeśli więc chcemy dopuścić do efektu przejścia dla wszystkich właściwości, to poniższe deklaracje będą niemal równoważne:

```
div {
    color: #ff0000;
    border: 1px solid #00ff00;
    border-radius: 0;
    transform: scale(1) rotate(0deg);
    opacity: 1;
    box-shadow: 3px 3px rgba(0, 0, 0, 0.1);
    width: 50px;
    padding: 100px;
    transition-property: color, border, border-radius, transform, opacity,
    ↪box-shadow, width, padding;
    transition-duration: 1s;
}

div {
    color: #ff0000;
    border: 1px solid #00ff00;
    border-radius: 0;
    transform: scale(1) rotate(0deg);
    opacity: 1;
    box-shadow: 3px 3px rgba(0, 0, 0, 0.1);
    width: 50px;
    padding: 100px;
    transition-property: all;
    transition-duration: 1s;
}
```

Obie powyższe deklaracje `transition-property` zezwalają na animowane przejścia we wszystkich wymienionych właściwościach, ale dla pierwszej deklaracji oznacza to tylko osiem właściwości. Należy pamiętać, że reguły dla tych właściwości mogą też występować w innych blokach reguł. Innymi słowy, akurat w tym przypadku wszystkie osiem animowanych właściwości zostało wymienionych w tym samym bloku, w którym znajduje się deklaracja `transition-property`, ale nie musi tak być.

Deklaracja `transition-property: all` w drugim bloku reguł gwarantuje, że *wszystkie* animowane właściwości zmienią swoją wartość w przypadku zajścia zdarzenia, które wywoła zmianę stylu — bez względu na to, w którym bloku reguł CSS wystąpi właściwość o zmienionej wartości — i zmiana ta będzie trwała dokładnie sekundę. Przejście będzie miało zastosowanie do wszystkich animowanych właściwości wszystkich elementów pasujących do selektora, a nie tylko tych właściwości, które zostały zadeklarowane w tym samym bloku stylów, w którym użyto słowa kluczowego `all`.

W omawianym przypadku pierwsza wersja deklaracji ogranicza rodzaj przejść do ośmiu wymienionych właściwości, ale daje nieco większą kontrolę nad tym, jak te przejścia będą w ich przypadku wyglądały. Zadeklarowanie osobnych właściwości pozwala na określenie różnych prędkości, opóźnień i (lub) czasów trwania przejścia dla każdej z nich, jeśli także właściwości poszczególnych przejść zdefiniujemy osobno:

```
div {
    color: #ff0000;
    border: 1px solid #0f0;
    border-radius: 0;
    transform: scale(1) rotate(0deg);
    opacity: 1;
    box-shadow: 3px 3px rgba(0, 0, 0, 0.1);
    width: 50px;
    padding: 100px;
}

.foo {
    color: #00ff00;
    transition-property: color, border, border-radius, transform, opacity,
        box-shadow, width, padding;
    transition-duration: 1s;
}
<div class="foo">Hello</div>
```

Jeśli chcesz zdefiniować przejścia dla każdej właściwości osobno, wymień je wszystkie i rozdziel poszczególne właściwości przecinkiem. Jeżeli chcesz zastosować animację niemal wszystkich właściwości (powiedzmy, z kilkoma wyjątkami), która nastąpi jednocześnie, z takim samym opóźnieniem i w tym samym tempie, to możesz użyć kombinacji słowa kluczowego `all` oraz nazw tych właściwości, które mają być animowane w innym czasie czy z inną prędkością. Pamiętaj, aby wówczas podać `all` jako pierwszą wartość:

```
div {
    color: #f00;
    border: 1px solid #00ff00;
    border-radius: 0;
    transform: scale(1) rotate(0deg);
    opacity: 1;
    box-shadow: 3px 3px rgba(0, 0, 0, 0.1);
    width: 50px;
    padding: 100px;
    transition-property: all, border-radius, opacity;
    transition-duration: 1s, 2s, 3s;
}
```

Słowo kluczowe `all` występujące na liście właściwości rozdzielonych przecinkami obejmuje wszystkie właściwości podane w powyższym przykładzie, w tym także odziedziczone właściwości CSS oraz wszystkie właściwości zdefiniowane w dowolnym innym bloku reguł CSS, który pasuje do danego elementu bądź jest przez ten element dziedziczony.

W poprzednim przykładzie wszystkie właściwości, którym nadane zostaną nowe wartości, zostaną poddane przejściu o takim samym czasie trwania, opóźnieniu i tempie — z wyjątkiem właściwości `border-radius` oraz `opacity`, które w sposób jawny wymieniliśmy osobno. Ponieważ umieściliśmy je na rozdzielonej przecinkami liście wartości, po wartości `all`, możemy albo poddać je przejściu jednocześnie, z tym samym opóźnieniem i tempem, albo podać dla nich dwóch inne wartości czasu, opóźnień i tempa. W tym konkretnym przypadku wszystkie właściwości będą trwały 1 sekundę, oprócz właściwości `border-radius` i `opacity`, których przejście będzie trwało, odpowiednio, 2 i 3 sekundy. (Właściwość `transition-duration` została omówiona w dalszej części tego rozdziału).

Pamiętaj o wymienieniu słowa kluczowego `all` na początku listy wartości rozdzielonej przecinkami, bo właściwości zadeklarowane przed słowem `all` będą podlegały tym samym regułom co `all`. W rezultacie parametry przejść tych właściwości, które zamierzałeś podać osobno, zostaną nadpisane parametrami dotyczącymi zbioru właściwości opisanego słowem kluczowym `all`.

Wyłączanie przejść poprzez ograniczanie puli właściwości

Choć przejścia inne niż natychmiastowe są domyślnie wyłączone, to jeśli zdefiniujesz jakieś przejście CSS i będziesz chciał je wyłączyć w jakiejś konkretnej sytuacji, możesz w tym celu użyć deklaracji `transition-property: none`. Dzięki temu zyskasz gwarancję wyłączenia przejść dla wszystkich właściwości.

Słowo kluczowe `none` może być użyte tylko jako pojedyncza wartość tej właściwości — nie może być jedną z rozdzielonych przecinkami wartości na liście. Jeśli chcesz wyłączyć przejścia dla pewnego ograniczonego zbioru właściwości, musisz postąpić niejako na odwrót i wymienić wszystkie właściwości, których przejścia nadal chcesz zostawić włączone. Właściwości `transition-property` nie da się użyć do wykluczania właściwości; służy ona raczej do uwzględniania ich.

Inna metoda polega na ustawieniu dla danej właściwości opóźnienia i czasu trwania przejścia na `0s`. Dzięki temu zmiany wartości właściwości będą zachodzić natychmiastowo, tak jakby nie było dla nich zdefiniowanego przejścia.

Zdarzenia związane z przejściami

W modelu DOM na końcu każdego przejścia — w obu kierunkach i dla każdej właściwości, która podlega przejściu w ciągu dowolnej ilości czasu *albo* po dowolnym opóźnieniu — wyzwalane jest zdarzenie `transitionend`. Dzieje się tak bez względu na to, czy dana właściwość została zadeklarowana osobno, czy stanowi część deklaracji związanej ze słowem kluczowym `all`. W przypadku niektórych na pozór pojedynczych deklaracji właściwości może zajść kilka zdarzeń `transitionend`, bo każda animowana składowa w ramach właściwości skrótowej wywoła własne zdarzenie tego typu. Rozważmy następujący przykład:

```
div {
    color: #f00;
    border: 1px solid #00ff00;
    border-radius: 0;
    transform: scale(1) rotate(0deg);
    opacity: 1;
    box-shadow: 3px 3px rgba(0, 0, 0, 0.1);
    width: 50px;
    padding: 100px;
    transition-property: all, border-radius, opacity;
    transition-duration: 1s, 2s, 3s;
}
```

Po zakończeniu przejść dojdzie do co najmniej ośmiu zdarzeń `transitionend`. Na przykład przejście samej tylko właściwości `border-radius` generuje cztery zdarzenia `transitionend`, po jednym dla:

- `border-bottom-left-radius`,
- `border-bottom-right-radius`,
- `border-top-right-radius`,
- `border-top-left-radius`.

Właściwość `padding` też jest wszak skrótem dla czterech pełnych właściwości:

- `padding-top`,
- `padding-right`,
- `padding-bottom`,
- `padding-left`.

Skrótowa właściwość `border` generuje osiem zdarzeń `transitionend`: czterokrotnie dla czterech właściwości odpowiadających skrótowi `border-width` i czterokrotnie dla właściwości reprezentowanych przez `border-color`:

- `border-left-width`,
- `border-right-width`,
- `border-top-width`,
- `border-bottom-width`,
- `border-top-color`,
- `border-left-color`,
- `border-right-color`,
- `border-bottom-color`.

Zdarzenia `transitionend` nie zajdą jednak dla właściwości `border-style`, bo nie jest ona właściwością animowaną.

Skąd wiemy, że właściwość `border-style` nie jest animowana? Możemy tak założyć, bo nie ma sensownego punktu pośredniego między jej dwiema wartościami, takimi jak `solid` i `dashed`. Możemy to sprawdzić na liście animowanych właściwości w dodatku A albo w specyfikacji każdej z nich z osobna.

W naszym scenariuszu z wymienionymi ośmioma konkretnymi właściwościami zajdzie 21 zdarzeń `transitionend`, bo wśród tych ośmiu są właściwości skrótowe, które mają różne wartości w stanach przed przejściem i po nim. W przypadku słowa kluczowego `all` będziemy mieli do czynienia z co najmniej 21 zdarzeniami `transitionend`: po jednym dla każdej ze składowych 8 właściwości, które mają zdefiniowane stany „przed i po” oraz potencjalnie inne, wynikające z dziedziczenia albo deklaracji właściwości w innych blokach stylów mających wpływ na animowany element.

Zdarzeń `transitionend` można nasłuchiwać w następujący sposób:

```
document.querySelector('div').addEventListener('transitionend',
    function (e) {
      console.log(e.propertyName);
});
```

Zdarzenie `transitionend` zawiera trzy atrybuty, które to zdarzenie opisują:

1. `propertyName` — jest to nazwa właściwości CSS, której przejście właśnie się zakończyło.
2. `pseudoElement` — jest to poprzedzony dwoma dwukropkami pseudoelement, którego dotyczyło przejście. Może to być pusty łańcuch znaków, jeśli przejście dotyczyło zwykłego węzła DOM.
3. `elapsedTime` — jest to ilość czasu, jaką zajęło przejście (w sekundach). Zwykle jest to czas podany jako wartość właściwości `transition-duration`.

Zdarzenie `transitionend` zachodzi tylko wtedy, gdy zmiana wartości właściwości w ramach przejścia zakończy się powodzeniem. Zdarzenie `transitionend` nie zachodzi, jeśli przejście zostało przerwane — na przykład przez kolejną zmianę tej samej właściwości tego samego elementu.

Przywrócenie wartości początkowej właściwości powoduje zajście kolejnego zdarzenia `transitionend`. To drugie zdarzenie zachodzi zawsze, jeśli tylko przejście się rozpocznie — nawet jeśli nie zostanie ono dokończone w pierwotnym kierunku.

Ustalanie czasu trwania przejścia

Właściwość `transition-duration` przyjmuje wartość w postaci rozdzielonej przecinkami listy wartości określających interwały czasowe, w sekundach (`s`) albo w milisekundach (`ms`). Wartości te opisują czas trwania przejścia z jednego stanu do drugiego.

transition-duration

Wartości:	*<time>#*
Wartość początkowa:	`0s`
Stosuje się do:	Wszystkich elementów oraz pseudoelementów `:before` i `:after`
Wartość wyliczona:	Jak określono
Dziedziczona:	Nie
Animowana:	Nie

Jeśli mamy do czynienia z przejściem w tę i z powrotem między dwoma stanami, a w deklaracji został podany czas trwania dotyczący tylko jednego z tych stanów, to ów czas będzie rzutował tylko na przejście *do* tego stanu. Rozważmy:

```
input:invalid {
    transition-duration: 1s;
    background-color: red;
}

input:valid {
    transition-duration: 0.2s;
    background-color: green;
}
```

Jeśli dla właściwości `transition-duration` zadeklarowane zostaną różne wartości, to czas trwania danego przejścia będzie odpowiadał wartości `transition-duration` zadeklarowanej w bloku reguł opisującym docelowe właściwości tego przejścia. W poprzednim przykładzie zmiana koloru tła pola

wprowadzania danych na czerwone, jeśli w polu tym zostanie wprowadzona nieprawidłowa wartość, potrwa 1 sekundę. Przejście do zielonego tła po wprowadzeniu prawidłowej wartości będzie zaś trwało tylko 200 milisekund. ▶

Właściwość `transition-duration` przyjmuje wartości dodatnie w sekundach (`s`) albo w milisekundach (`ms`). Jednostka czasu — `ms` albo `s` — jest wymagana w specyfikacji nawet w przypadku czasu ustawionego na `0s`. Domyślnie zmiana właściwości z jednej wartości na drugą zachodzi błyskawicznie, bez widocznej animacji — właśnie dlatego domyślna wartość czasu trwania przejścia wynosi `0s`.

Jeśli pominie się deklarację właściwości `transition-duration`, to (chyba że właściwość `transition--delay` ma wartość dodatnią) efekt jest taki, jakby właściwość `transition-property` też nie została zadeklarowana — nie zachodzi wtedy także zdarzenie `transitionend`. Dopóki łączny czas przejścia jest większy niż zero sekund (czas zerowy może wynikać z jawnej deklaracji `0s` albo z pominięcia właściwości `transition-duration`, co skutkuje nadaniem jej domyślnej wartości `0s`), przejście nastąpi, a po jego zakończeniu zajdzie zdarzenie `transitionend`.

Ujemne wartości właściwości `transition-duration` są nieprawidłowe, a jeśli wartość taka zostanie podana, unieważni całą deklarację właściwości.

Korzystając z tej samej co wcześniej, bardzo długiej formy deklaracji `transition-property`, możemy określić wspólny czas trwania dla wszystkich właściwości bądź osobne czasy trwania dla każdej z nich. Możemy też zadecydować, by na przykład co druga z podanych na liście właściwości była animowana przez taki sam czas. Jeśli chcemy zadeklarować pojedynczy czas trwania dla wszystkich właściwości biorących udział w przejściu, wystarczy, że podamy jedną wartość `transition-duration`:

```
div {
    color: #ff0000;
    ...
    transition-property: color, border, border-radius, transform, opacity,
    ↪box-shadow, width, padding;
    transition-duration: 200ms;
}
```

Dla właściwości `transition-duration` możemy też podać tyle samo rozdzielonych przecinkami wartości czasu, ile właściwości CSS zostało wymienionych w charakterze wartości właściwości `transition-property`. Jeśli chcemy, aby przejście dla każdej z zadeklarowanych właściwości zajmowało różny czas, musimy dla każdej z osobna podać wartość czasu na liście:

```
div {
    color: #ff0000;
    ...
    transition-property: color, border, border-radius, transform, opacity,
    ↪box-shadow, width, padding;
    transition-duration: 200ms, 180ms, 160ms, 140ms, 120ms, 100ms, 1s, 2s;
}
```

Jeśli liczba zadeklarowanych właściwości jest inna niż liczba zadeklarowanych czasów trwania, przeglądarka radzi sobie z tą różnicą, kierując się ściśle określonymi zasadami. Otóż jeżeli czasów trwania jest więcej niż właściwości, zbędne czasy trwania są ignorowane. Jeśli z kolei właściwości jest więcej niż czasów, wartości czasów są powtarzane od początku. W poniższym przykładzie przejścia właściwości `color`, `border-radius`, `opacity` i `width` będą trwały po 100 milisekund, zaś przejścia właściwości `border`, `transform`, `box-shadow` i `padding` będą trwały po 200 milisekund.

```
div {
    ...
    transition-property: color, border, border-radius, transform, opacity,
    ↪box-shadow, width, padding;
    transition-duration: 100ms, 200ms;
}
```

Jeśli zadeklarujemy dwa czasy trwania oddzielone przecinkiem, każda nieparzysta właściwość będzie animowana zgodnie z pierwszą podaną wartością czasu, a każda parzysta zgodnie z drugą.

Należy pamiętać o wygodzie obsługi. Jeśli przejście jest zbyt wolne, strona internetowa będzie sprawiała wrażenie ociężałej albo reagującej z opóźnieniem i skupi uwagę użytkownika na czymś, co powinno być jedynie subtelnym efektem. Jeśli z kolei przejście jest za szybkie, może być trudne do dostrzeżenia. Choć da się podać dowolną dodatnią wartość czasu trwania przejść, to Twoim celem powinno być raczej uatrakcyjnienie odbioru strony, a nie poirytowanie użytkownika. Efekty powinny trwać wystarczająco długo, by dało się je zauważyć, ale nie aż tak długo, by przyciągały uwagę. Na ogół najlepsze rezultaty dają czasy rzędu 100 – 200 milisekund, które przekładają się na przejścia zauważalne, ale nie dekoncentrujące.

W przypadku naszego rozwijanego menu też zależy nam na wygodzie użytkownika, czas przejścia obu animowanych właściwości ustaliliśmy więc na 200 milisekund.

```
nav li ul {
    transition-property: transform, opacity;
    transition-duration: 200ms;
    ...
}
```

Zmiana tempa przejść

Chciałbyś, aby przejście zaczynało się wolno i przyspieszało, zaczynało się szybko i zwalniało, odbywało się w jednostajnym tempie, polegało na zmianach skokowych, a może nawet ulegało „sprężystemu" wytłumianiu? Nie ma problemu — do sterowania tempem przejścia służy właściwość `transition-timing-function`.

transition-timing-function

Wartości:	*<timing-function>#*
Wartość początkowa:	`Ease`
Stosuje się do:	Wszystkich elementów oraz pseudoelementów `:before` i `:after`
Wartość wyliczona:	Jak określono
Dziedziczona:	Nie
Animowana:	Nie

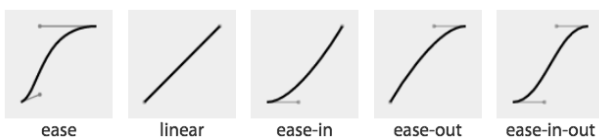
Właściwość `transition-timing-function` przyjmuje następujące wartości: `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out`, `step-start`, `step-end`, `steps(n, start)` — gdzie `n` jest liczbą kroków — `steps(n, end)` oraz `cubic-bezier(x1, y1, x2, y2)`. (Wartości te są też przyjmowane przez właściwość `animation-timing-function` i zostały bardziej szczegółowo omówione w rozdziale 18.).

Słowa kluczowe inne niż `step-` są funkcjami płynnej zmiany prędkości, służącymi jako wygodne zamienniki konkretnych funkcji Béziera drugiego stopnia, których wykresy mają postać łagodnych krzywych. Specyfikacja obejmuje definicje pięciu takich funkcji, zebranych w tabeli 17.1.

Tabela 17.1. Słowa kluczowe odpowiadające zmianom prędkości opisanym krzywymi Béziera

Funkcja czasu	Opis	Odpowiednik w postaci cubic-bezier
`cubic-bezier()`	Określa krzywą Béziera drugiego stopnia.	`cubic-bezier(x1, y1, x2, y2)`
`ease`	Przejście zaczyna się wolno, następnie przyspiesza, potem spowalnia i kończy się bardzo wolno.	`cubic-bezier(0.25, 0.1, 0.25, 1)`
`linear`	Przejście przez cały czas odbywa się w tym samym tempie.	`cubic-bezier(0, 0, 1, 1)`
`ease-in`	Przejście zaczyna się wolno, potem przyspiesza.	`cubic-bezier(0.42, 0, 1, 1)`
`ease-out`	Przejście zaczyna się szybko, potem zwalnia.	`cubic-bezier(0, 0, 0.58, 1)`
`ease-in-out`	Podobnie jak `ease` — przejście jest szybsze w połowie, zaczyna się wolno, ale kończy się nie aż tak wolno jak `ease`.	`cubic-bezier(0.42, 0, 0.58, 1)`

Krzywe Béziera, także te, które odpowiadają za pięć predefiniowanych funkcji regulacji tempa zebranych w tabeli 17.1 i przedstawionych na wykresach na rysunku 17.3, przyjmują cztery parametry liczbowe. Na przykład wartość `linear` odpowiada wyrażeniu `cubic-bezier(0, 0, 1, 1)`. Pierwszy i trzeci parametr funkcji Béziera muszą się zawierać w przedziale od 0 do +1.

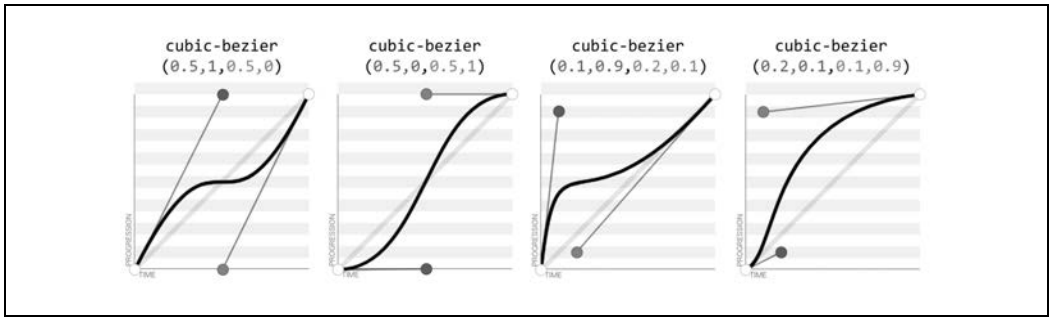


Rysunek 17.3. Krzywe odpowiadające predefiniowanym funkcjom Béziera

Cztery parametry funkcji `cubic-bezier()` definiują współrzędne *x* oraz *y* dwóch *uchwytów* na diagramie funkcji. Uchwyty te są powiązane z końcami wykresu krzywej, który rozciąga się od lewego dolnego do prawego górnego rogu diagramu. Krzywe są konstruowane na podstawie tych dwóch rogów oraz współrzędnych uchwytów z wykorzystaniem funkcji Béziera.

Aby się przekonać, na czym to polega, przyjrzyj się krzywym oraz odpowiadającym im wartościom, pokazanym na rysunku 17.4.

Rozważmy pierwszy przykład. Dwie pierwsze wartości, odpowiadające parametrom *x1* i *y1*, to `0.5` oraz 1. Położenie pierwszego uchwytu wyznaczamy, idąc najpierw do połowy diagramu (*x1* = 0,5), a potem na samą górę (*y1* = 1). Na tej samej zasadzie współrzędne `0.5, 0` dla parametrów *x2*, *y2* wyznaczają położenie drugiego uchwytu — na dole diagramu, pośrodku. Pokazana krzywa wynika z położenia obu uchwytów.

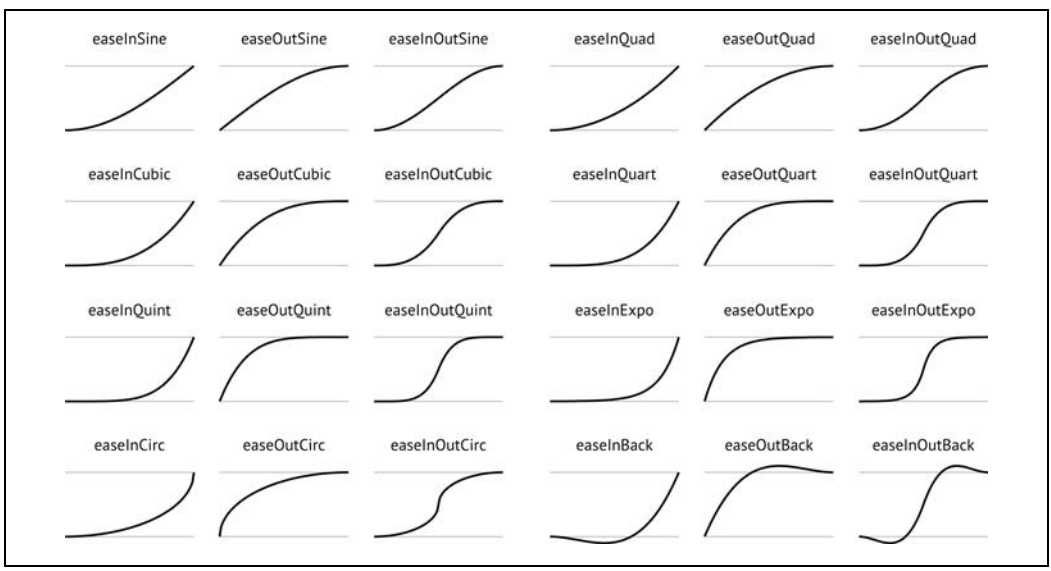


Rysunek 17.4. Cztery krzywe Béziera oraz wartości funkcji cubic-bezier() (na podstawie http://cubic-bezier.com*)*

Na drugim przykładzie współrzędne uchwytów zostały zamienione miejscami, co poskutkowało zmianą przebiegu krzywej. Na tej samej zasadzie zmienione zostały przykłady trzeci i czwarty, które są niejako swoimi przeciwieństwami. Zwróć uwagę na zmianę w kształcie otrzymanej krzywej, wynikającą z zamiany położenia uchwytów.

Predefiniowane słowa kluczowe mają dość ograniczone możliwości. Jeśli chciałbyś uzyskać efekty lepiej odzwierciedlające prawidła animacji, to zamiast predefiniowanych słów być może powinieneś użyć funkcji Béziera drugiego stopnia opisanych czterema parametrami w postaci liczb rzeczywistych. Jeśli masz algebrę w małym palcu albo ogromne doświadczenie w obsłudze programów takich jak FreeHand czy Illustrator, to być może bez trudu umiesz sobie wyobrazić potrzebne krzywe; w przeciwnym razie skorzystaj z narzędzi online umożliwiających eksperymentowanie z różnymi wartościami (na przykład *http://cubic-bezier.com*). Pozwalają one na porównywanie różnych krzywych — zarówno tych odpowiadających predefiniowanym słowom kluczowym, jak i samodzielnie zdefiniowanych funkcji Béziera.

Na rysunku 17.5 zostały pokazane różne inne przebiegi funkcji Béziera, dostępne na stronie *http://easings.net*. Umożliwiają one uzyskanie bardziej realistycznych i estetycznych animacji.



Rysunek 17.5. Przydatne funkcje Béziera opracowane przez innych autorów (ze strony http://easings.net*)*

Choć autorzy wspomnianej strony nazwali opracowane przez siebie rodzaje animacji, ich nazwy nie wchodzą w skład specyfikacji CSS. Należy je zapisać następująco:

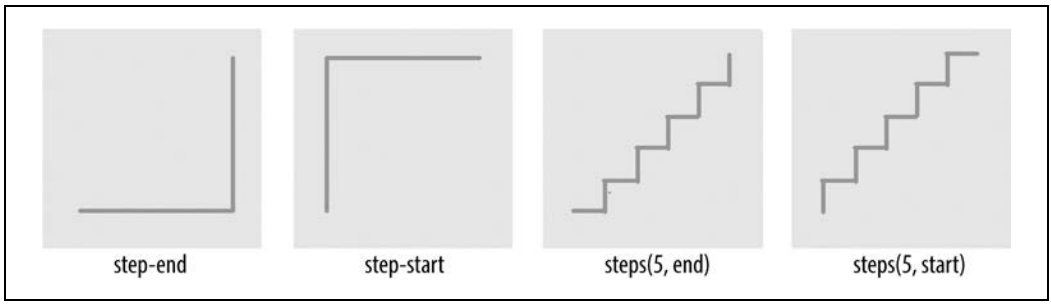
Nieoficjalna nazwa	Parametry funkcji Béziera drugiego stopnia
`easeInSine`	`cubic-bezier(0.47, 0, 0.745, 0.715)`
`easeOutSine`	`cubic-bezier(0.39, 0.575, 0.565, 1)`
`easeInOutSine`	`cubic-bezier(0.445, 0.05, 0.55, 0.95)`
`easeInQuad`	`cubic-bezier(0.55, 0.085, 0.68, 0.53)`
`easeOutQuad`	`cubic-bezier(0.25, 0.46, 0.45, 0.94)`
`easeInOutQuad`	`cubic-bezier(0.455, 0.03, 0.515, 0.955)`
`easeInCubic`	`cubic-bezier(0.55, 0.055, 0.675, 0.19)`
`easeOutCubic`	`cubic-bezier(0.215, 0.61, 0.355, 1)`
`easeInOutCubic`	`cubic-bezier(0.645, 0.045, 0.355, 1)`
`easeInQuart`	`cubic-bezier(0.895, 0.03, 0.685, 0.22)`
`easeOutQuart`	`cubic-bezier(0.165, 0.84, 0.44, 1)`
`easeInOutQuart`	`cubic-bezier(0.77, 0, 0.175, 1)`
`easeInQuint`	`cubic-bezier(0.755, 0.05, 0.855, 0.06)`
`easeOutQuint`	`cubic-bezier(0.23, 1, 0.32, 1)`
`easeInOutQuint`	`cubic-bezier(0.86, 0, 0.07, 1)`
`easeInExpo`	`cubic-bezier(0.95, 0.05, 0.795, 0.035)`
`easeOutExpo`	`cubic-bezier(0.19, 1, 0.22, 1)`
`easeInOutExpo`	`cubic-bezier(1, 0, 0, 1)`
`easeInCirc`	`cubic-bezier(0.6, 0.04, 0.98, 0.335)`
`easeOutCirc`	`cubic-bezier(0.075, 0.82, 0.165, 1)`
`easeInOutCirc`	`cubic-bezier(0.785, 0.135, 0.15, 0.86)`
`easeInBack`	`cubic-bezier(0.6, -0.28, 0.735, 0.045)`
`easeOutBack`	`cubic-bezier(0.175, 0.885, 0.32, 1.275)`
`easeInOutBack`	`cubic-bezier(0.68, -0.55, 0.265, 1.55)`

Tempo krokowe

Dostępne są też funkcje umożliwiające zdefiniowanie tempa krokowego, a także dwa predefiniowane rodzaje takiego przejścia:

Funkcja tempa	Definicja
`step-start`	Przez cały czas przejścia wyświetlany jest ostatni krok animacji. Równoważne `steps(1, start)`.
`step-end`	Przez cały czas przejścia wyświetlany jest ostatni krok animacji. Równoważne `steps(1, end)`.
`steps(n, start)`	Wyświetlanych jest *n* pojedynczych kroków, przy czym pierwszy krok pojawia się w chwili odpowiadającej n/100 procent całkowitego czasu przejścia.
`steps(n, end)`	Wyświetlanych jest *n* pojedynczych kroków, przy czym wartości początkowe tych kroków są widoczne przez n/100 procent całkowitego czasu przejścia.

Jak wynika z rysunku 17.6, funkcje krokowe przedstawiają przejście od wartości początkowej do wartości końcowej w sekwencji kroków, a nie na podstawie płynnej krzywej.



Rysunek 17.6. Funkcje opisujące tempo krokowe

Funkcje opisujące tempo krokowe umożliwiają podzielenie przejścia na równe kroki. Funkcje te definiują liczbę i kierunek kroków. Są dwie opcje kierunku: `start` i `end`. W przypadku opcji `start` pierwszy krok odpowiada samemu początkowi animacji. W przypadku opcji `end` ostatni krok odpowiada samemu końcowi animacji. Na przykład wyrażenie `steps(5, end)` spowodowałoby wyświetlenie następujących równych kroków przejścia: 0%, 20%, 40%, 60% i 80%. Z kolei wyrażenie `steps(5, start)` spowodowałoby wyświetlenie następujących równych kroków: 20%, 40%, 60%, 80% i 100%.

Funkcja `step-start` jest odpowiednikiem wyrażenia `steps(1, start)`. W przypadku jej zastosowania przejście właściwości polega na wyświetleniu *końcowej* wartości tego przejścia, od początku do samego końca jego trwania. Funkcja `step-end` jest odpowiednikiem wyrażenia `steps(1, end)` i powoduje wyświetlanie *początkowej* wartości przejścia, od początku do samego końca jego trwania.

Tempo krokowe, ze szczególnym uwzględnieniem znaczenia wartości `start` i `end`, zostało omówione w rozdziale 18.

Wróćmy do wspomnianej wcześniej bardzo długiej deklaracji właściwości `transition-property`. Za jej pomocą możemy zadeklarować jedną funkcję tempa dla wszystkich właściwości albo zdefiniować różne funkcje tempa dla każdej właściwości z osobna i tak dalej. W poniższym przypadku wszystkie właściwości ze zdefiniowanymi przejściami będą trwały tyle samo i przebiegną w tym samym tempie:

```
div {
    transition-property: color, border-width, border-color, border-radius, transform,
    ↪opacity, box-shadow, width, padding;
    transition-duration: 200ms;
    transition-timing-function: ease-in;
}
```

Na własną prośbę możemy też utrudnić użytkownikowi korzystanie ze strony, animując przejścia różnych właściwości w zupełnie innym tempie. (Jak w kolejnym przykładzie).

Pamiętaj, że właściwość `transition-timing-function` nie zmienia czasu trwania przejścia — od tego mamy właściwość `transition-duration`. Właściwość ta zmienia tylko tempo przejścia w ramach określonego czasu. Rozważmy następujący przykład:

```
div {
    ...
    transition-property: color, border-width, border-color, border-radius, transform,
    ↪opacity, box-shadow, width, padding;
    transition-duration: 200ms;
    transition-timing-function: ease, ease-in, ease-out, ease-in-out, linear,
    ↪step-end, step-start, steps(5, start), steps(3, end);
}
```

Nawet jeśli podamy dziewięć różnych funkcji tempa dla dziewięciu różnych właściwości, to dopóki przejścia będą miały ten sam czas trwania i to samo opóźnienie, wszystkie właściwości rozpoczną i zakończą przejście jednocześnie. Funkcje tempa decydują tylko o przebiegu przejścia w czasie jego trwania, ale nie mają wpływu na czas, w jakim to przejście się zakończy. (Jak już wspomnieliśmy, przejścia takie jak pokazane przed chwilą byłyby bardzo uciążliwe pod względem użytkowym. Wystrzegaj się tego rodzaju metod).

Najlepszym sposobem obeznania się z różnymi funkcjami tempa jest poeksperymentowanie z nimi i przekonanie się, które najlepiej nadają się do uzyskania potrzebnego Ci w danej sytuacji efektu. Podczas testowania nadaj właściwości `transition-duration` stosunkowo dużą wartość, aby lepiej uwidocznić różnice między funkcjami. ▶ Przy szybszych animacjach możesz nie być w stanie dostrzec różnic między niektórymi funkcjami tempa; pamiętaj tylko, by skrócić czas trwania przejść przed umieszczeniem projektu w internecie!

Opóźnianie przejść

Właściwość `transition-delay` umożliwia wprowadzenie opóźnienia między zmianą inicjującą przejście zdefiniowane dla danego elementu a rzeczywistym rozpoczęciem tego przejścia.

transition-delay

Wartości:	*<time>*#
Wartość początkowa:	`0s`
Stosuje się do:	Wszystkich elementów oraz pseudoelementów `:before` i `:after`
Wartość wyliczona:	Jak określono
Dziedziczona:	Nie
Animowana:	Nie

Wartość `0s` właściwości `transition-delay` (wartość domyślna) oznacza, że przejście rozpocznie się natychmiast — zacznie się ono w chwili, gdy stan elementu ulegnie zmianie. Takie podejście znamy na przykład z natychmiastowej zmiany typu `a:hover`.

W przypadku wartości innej niż `0s` parametr *<time>* właściwości `transition-delay` określa czas opóźnienia od chwili zmiany danej właściwości — przez czas opóźnienia żadne zmiany określone przez właściwości `transition` albo `transition-property` nie będą stosowane — aż do chwili, w której rozpocznie się animacja zadeklarowana we właściwościach `transition` albo `transition-property` (i będzie trwała aż do osiągnięcia wartości docelowych).

Co ciekawe, w przypadku tej właściwości można stosować ujemne wartości czasu. Efekty możliwe do uzyskania za pomocą ujemnych wartości `transition-delay` zostały opisane w dalszej części tego rozdziału, w podpunkcie „Ujemne wartości opóźnień”.

Wróćmy do przykładu z deklaracją `transition-property` obejmującą 8 (albo 21) właściwości. Aby wszystkie właściwości rozpoczęły przejścia od razu, bez opóźnienia, możemy pominąć deklarację właściwości `transition-delay` albo uwzględnić ją i nadać jej wartość `0s`. Równie dobrze można też rozpocząć animację połowy przejść od razu, a reszty — 200 milisekund później, jak w poniższym przykładzie:

```
div {
    transition-property: color, border, border-radius, transform, opacity,
        box-shadow, width, padding;
    transition-duration: 200ms;
    transition-timing-function: linear;
    transition-delay: 0s, 200ms;
}
```

Zastosowanie reguły `transition-delay: 0s, 200ms` dla sekwencji kilku właściwości sprawia, że (w tym przypadku) `color`, `border-radius`, `opacity` i `width` rozpoczną przejście od razu. Dla reszty właściwości animacja rozpocznie się w chwili, gdy parzyste właściwości zakończą przejście, bo wartość `transition-delay` dla tej reszty jest równa wartości czasu `transition-duration` dla wszystkich animowanych właściwości.

Podobnie jak ma to miejsce w przypadku właściwości `transition-duration` i `transition-timing-function`, jeśli liczba rozdzielonych przecinkami wartości `transition-delay` przewyższa liczbę rozdzielonych przecinkami wartości `transition-property`, nadmiarowe wartości opóźnień są ignorowane. Jeśli liczba rozdzielonych przecinkami wartości `transition-property` przewyższa liczbę rozdzielonych przecinkami wartości `transition-delay`, po wyczerpaniu dostępnych wartości opóźnień ich lista jest przeglądana od początku.

Możliwe jest zadeklarowanie dziewięciu różnych wartości `transition-delay` w taki sposób, by każda właściwość zaczęła przejście dopiero w chwili, gdy poprzednia je ukończy:

```
div {
    ...
    transition-property: color, border-width, border-color, border-radius, transform,
    ↪opacity, box-shadow, width, padding;
    transition-duration: 200ms;
    transition-timing-function: linear;
    transition-delay: 0s, 0.2s, 0.4s, 0.6s, 0.8s, 1s, 1.2s, 1.4s, 1.6s;
}
```

W tym przykładzie za pomocą właściwości `transition-duration` zadeklarowaliśmy, że każde przejście ma trwać 200 milisekund. Następnie przy użyciu właściwości `transition-delay` podaliśmy listę rozdzielonych przecinkami wartości — każda kolejna jest większa od poprzedniej o 200 milisekund (0,2 sekundy), czyli tyle samo, ile trwa przejście poszczególnych właściwości. W ten sposób kolejna właściwość rozpocznie przejścia w chwili, gdy poprzednia je ukończy.

Możemy też wyliczyć czas trwania i opóźnienia tak, by wszystkie animowane właściwości zakończyły przejście równocześnie:

```
div {
    ...
```

```
    transition-property: color, border-width, border-color, border-radius, transform,
    ↪opacity, box-shadow, width, padding;
    transition-duration: 1.8s, 1.6s, 1.4s, 1.2s, 1s, 0.8s, 0.6s, 0.4s, 0.2s;
    transition-timing-function: linear;
    transition-delay: 0s, 0.2s, 0.4s, 0.6s, 0.8s, 1s, 1.2s, 1.4s, 1.6s;
}
```

W podanym przykładzie wszystkie właściwości kończą przejścia po upłynięciu 1,8 sekundy, ale każde z przejść ma inny czas trwania i inne opóźnienie. Dla każdego jednak suma wartości właściwości `transition-duration` oraz `transition-delay` wynosi 1,8 sekundy.

Zasadniczo należy dbać o to, by wszystkie przejścia zaczynały się jednocześnie. Możesz to osiągnąć, podając pojedynczą wartość dla właściwości `transition-delay` — wartość ta zostanie przypisana wszystkim animowanym właściwościom. W rozwijanym menu z rysunku 17.1 zastosowaliśmy opóźnienie o wartości 50 milisekund. To opóźnienie nie jest na tyle duże, by użytkownik zwrócił na nie uwagę, i nie spowoduje subiektywnego spowolnienia aplikacji. Takie trwające 50 milisekund opóźnienie może za to pomóc w uniknięciu niepożądanego, nagłego otwierania się menu, gdy podczas przesuwania kursora z jednej strony serwisu WWW (lub aplikacji) na drugą użytkownik przypadkiem wskaże którąś z pozycji menu.

Ujemne wartości opóźnień

Ujemna wartość właściwości `transition-delay`, mniejsza od wartości `transition-duration`, spowoduje natychmiastowe rozpoczęcie przejścia, ale nie od początku, lecz od pewnego momentu w trakcie tego przejścia. Na przykład:

```
div {
  transform: translateX(0);
  transition-property: transform;
  transition-duration: 200ms;
  transition-delay: -150ms;
  transition-timing-function: linear;
}
div:hover {
  transform: translateX(200px);
}
```

Biorąc pod uwagę fakt, że wartość `transition-delay` wynosi `–150ms`, a całe przejście trwa `200ms`, w tej sytuacji przejście rozpocznie się w trzech czwartych całego czasu jego trwania i będzie trwało 50 milisekund. W tym scenariuszu, przy założeniu liniowej funkcji tempa, od razu po wskazaniu elementu kursorem myszy element ten „przeskoczy” o `150px` wzdłuż osi x, a pozostałą część drogi od 150. piksela do 200. piksela pokona już płynnie w czasie 50 milisekund.

Jeśli wartość bezwzględna ujemnej wartości `transition-delay` jest równa lub większa od wartości `transition-duration`, to zmiana wartości właściwości nastąpi w sposób natychmiastowy — tak jakby w ogóle nie zadeklarowano reguły `transition`. W tej sytuacji zdarzenie `transitionend` *nie zachodzi*.

Domyślnie przy przejściu powrotnym od stanu `:hover` do stanu początkowego ma zastosowanie ta sama wartość właściwości `transition-delay`. W poprzednim scenariuszu, ponieważ wartość `transition-delay` dla stanu `:hover` nie została zmodyfikowana, w drodze powrotnej dojdzie najpierw do przeskoku o 75% całości ruchu wstecz (czyli licząc w kierunku pierwotnego przejścia, do pozycji znajdującej się w 25% drogi), a potem do płynnego przejścia do punktu początkowego. Innymi słowy,

po odsunięciu kursora myszy znad elementu przeskoczy on wzdłuż osi x do położenia odległego o 50 pikseli od położenia początkowego, a potem w ciągu 50 milisekund płynnie powróci do punktu wyjścia (czyli znajdzie się w miejscu o zerowym przesunięciu względem osi x).

Skrótowa właściwość transition

Skrótowa właściwość `transition` łączy cztery omówione dotąd właściwości: `transition-property`, `transition-duration`, `transition-timing-function` oraz `transition-delay`.

transition

Wartości:	*<single-transition>#*
Wartość początkowa:	`all 0s ease 0s`
Stosuje się do:	Wszystkich elementów oraz pseudoelementów `:before` i `:after`
Wartość wyliczona:	Jak określono
Dziedziczona:	Nie
Animowana:	Nie

<single-transition> = [[*none* | *<transition-property>*] || *<time>* || *<transition-timing-function>* || *<time>*]#

Właściwość `transition` przyjmuje wartość `none` lub dowolną liczbę rozdzielonych przecinkami zbiorów wartości opisujących *pojedyncze przejścia*. Pojedyncze przejście składa się z nazwy właściwości, która ma zostać mu poddana (bądź słowa kluczowego `all` w celu zastosowania przejścia dla wszystkich właściwości), czasu trwania przejścia, funkcji tempa oraz wartości opóźnienia.

Jeśli w skrótowej właściwości `transition` zadeklarujemy jeden zbiór parametrów przejścia i pominiemy w nim właściwość do animowania, to zamiast nazwy przyjęta zostanie domyślna wartość `all`. Jeśli pominięta zostanie wartość składowej `transition-timing-function`, przyjmie ona domyślną wartość `ease`. Jeśli podasz tylko jedną wartość czasu, będzie ona potraktowana jako czas trwania, a opóźnienie zostanie wyzerowane — tak jakby wartość właściwości `transition-delay` została ustawiona na `0s`.

W ramach każdego zbioru parametrów przejścia istotna jest kolejność czasu trwania i czasu opóźnienia: pierwsza wartość, którą da się zinterpretować jako czas, będzie potraktowana jako czas trwania. Jeśli przed przecinkiem albo przed końcem deklaracji wystąpi jeszcze jedna wartość czasu, zostanie ona potraktowana jako opóźnienie.

Oto trzy równoważne sposoby na uzyskanie tego samego efektu przejścia:

```
nav li ul {
    transition: transform 200ms ease-in 50ms,
                  opacity 200ms ease-in 50ms;
}

nav li ul {
    transition: all 200ms ease-in 50ms;
}
nav li ul {
    transition: 200ms ease-in 50ms;
}
```

W pierwszym przykładzie mamy skrót obejmujący dwie właściwości. Ponieważ przejście dotyczy wszystkich podanych właściwości, które zmieniają się w stanie `:hover`, moglibyśmy użyć słowa kluczowego `all`, tak jak w drugim przykładzie. A ponieważ `all` jest wartością domyślną, równie dobrze możemy zapisać całość w skrócie obejmującym tylko czas trwania, funkcję tempa oraz opóźnienie. Gdybyśmy użyli funkcji tempa `ease` zamiast `ease-in`, moglibyśmy także i ją pominąć w deklaracji, bo funkcja `ease` jest wybierana domyślnie.

Musimy jednak podać czas trwania, bo w przeciwnym razie przejście będzie niewidoczne. Innymi słowy, jedyna składowa właściwości `transition`, jaką możemy uznać za absolutnie niezbędną, to `transition-duration`.

Jeżeli zależałoby nam tylko na tym, by zmiana polegająca na otwarciu zamkniętego menu była po prostu opóźniona o pewien czas, ale przebiegała bez stopniowego przejścia, to i tak musielibyśmy podać wartość czasu trwania równą `0s`. Pamiętaj, że pierwsza wartość dająca się zinterpretować jako czas będzie potraktowana jako czas trwania, druga zaś jako opóźnienie:

```
nav li ul {
    transition: 0s 200ms; ...
```

Ta animacja będzie opóźniona o 200 milisekund, a potem na ekranie nagle, bez płynnego przejścia, pojawi się otwarte, w pełni nieprzezroczyste menu. Z perspektywy funkcjonalności jest to fatalne rozwiązanie. Choć jeśli zmienisz selektor z `nav li ul` na `*`, to może to być niezły pomysł na primaaprilisowy żart…

Jeśli mamy do czynienia z rozdzieloną przecinkami listą przejść (a nie z pojedynczą deklaracją), w której występuje słowo kluczowe `none`, cała deklaracja przejścia zostanie uznana za nieprawidłową i zignorowana.

```
div {
    transition-property: color, border-width, border-color, border-radius, transform,
    ↪opacity, box-shadow, width, padding;
    transition-duration: 200ms, 180ms, 160ms, 140ms, 120ms, 100ms, 1s, 2s, 3s;
    transition-timing-function: ease, ease-in, ease-out, ease-in-out, linear, step-end,
    ↪step-start, steps(5, start), steps(3, end);
    transition-delay: 0s, 0.2s, 0.4s, 0.6s, 0.8s, 1s, 1.2s, 1.4s, 1.6s;
}

div {
    transition:
        color 200ms,
        border-width 180ms ease-in 200ms,
        border-color 160ms ease-out 400ms,
        border-radius 140ms ease-in-out 600ms,
        transform 120ms linear 800ms,
        opacity 100ms step-end 1s,
        box-shadow 1s step-start 1.2s,
        width 2s steps(5, start) 1.4s,
        padding 3s steps(3, end) 1.6s;
}
```

Dwa poprzednie bloki reguł CSS są funkcjonalnymi odpowiednikami: możesz zadeklarować rozdzielone przecinkami wartości dla czterech osobnych składowych właściwości przejść bądź sporządzić jedną, rozdzieloną przecinkami listę wielu kompletnych przejść i przypisać ją skrótowej właściwości

transition. Nie da się jednak połączyć obu tych podejść: deklaracja transition: transform, opacity 200ms ease-in 50ms spowoduje płynną (w tempie ease-in) zmianę przezroczystości w ciągu 200 milisekund po trwającym 50 milisekund opóźnieniu, ale zmiana wynikająca z właściwości transform będzie natychmiastowa i nie wygeneruje zdarzenia transitionend.

Na odwrót: przejście do początku

W poprzednich przykładach deklarowaliśmy pojedyncze przejścia. Wszystkie przejścia zaczynały się od stanu domyślnego i były inicjowane przez wskazanie kursorem myszy (:hover). W przypadku takich deklaracji po odsunięciu kursora myszy znad elementu właściwości wracają do stanu początkowego z użyciem tego samego przejścia, odwrotnej funkcji tempa i z takim samym opóźnieniem.

W przypadku globalnych deklaracji przejść zarówno stan domyślny, jak i stan :hover bazują na tej samej regule transition — selektor pasuje do obydwu tych stanów. Możemy jednak uniknąć powielania całego przejścia (albo niektórych jego właściwości) w drodze powrotnej, podając inne wartości właściwości przejścia dla stanu globalnego i inne dla stanu :hover.

W przypadku deklaracji przejść dla różnych stanów każda z nich odnosi się do przejścia *do* danego stanu:

```
a {
    background: yellow;
    transition: 200ms background-color linear 0s;
  }
a:hover {
    background-color: orange;
    /* opóźnienie przy przejściu DO stanu :hover */
    transition-delay: 50ms;
  }
```

W tym przypadku, gdy użytkownik wskaże łącze kursorem myszy, kolor tła zacznie się zmieniać na pomarańczowy po opóźnieniu wynoszącym 50 milisekund. Kiedy użytkownik odsunie kursor znad łącza, kolor tła zacznie się zmieniać natychmiast. Przejście zajmuje 200 milisekund w każdą stronę, a zmiana następuje w sposób liniowy. Ponieważ opóźnienie trwające 50 milisekund zostało podane tylko dla stanu :hover („pomarańczowego”), opóźnienie to zachodzi tylko przed zmianą koloru tła na pomarańczowy.

W naszym przykładzie z rozwijanym menu w stanie :hover menu pojawia się i wydłuża w ciągu 200 milisekund w tempie ease-in po opóźnieniu wynoszącym 50 milisekund. Przejście zostało zdefiniowane za pomocą właściwości transition dla stanu domyślnego (element nie jest wskazany kursorem myszy). Gdy użytkownik odsunie kursor, wartości właściwości wracają do stanu pierwotnego w ciągu 200 milisekund, w tempie ease-out, po upływie trwającego 50 milisekund opóźnienia. Ów odwrotny efekt bazuje na wartości właściwości transition dla stanu, w którym element nie jest wskazany kursorem myszy. Jest to zachowanie domyślne, ale możemy je zmienić. Wprawdzie to domyślne zachowanie zapewnia największą wygodę obsługi, więc raczej nie należy go modyfikować, warto jednak wiedzieć, że da się to zrobić.

Jeśli zależałoby nam na tym, by zamykanie menu przebiegało skokowo i wolno (*nie* powinno nam na tym zależeć, bo będzie to uciążliwe dla użytkownika; na potrzeby tego przykładu powiedzmy jednak, że potrzebujemy takiego efektu), to możemy zadeklarować dwa różne przejścia:

```
nav ul ul {
  transform: scale(1, 0);
  opacity: 0;
  ...
  transition: all 4s steps(8, start) 1s;
}
nav li:hover ul {
  transform: scale(1, 1);
  opacity: 1;
  transition: all 200ms linear 50ms;
}
```

Przypominamy, że przejścia następują *do* stanu: kiedy dochodzi do zmiany stylu, w grę wchodzą nowe wartości właściwości biorących udział w przejściu, a nie stare. W podanym przykładzie zdecydowaliśmy się na płynną, liniową animację w stanie `:hover`. Przejście, które ma wtedy zastosowanie, jest przejściem do stanu docelowego. W przykładzie tym, kiedy użytkownik wskaże kursorem myszy element rodzica rozwijanego menu, czyli element `li`, nastąpi stopniowe, ale szybkie otwarcie tego menu — potrwa ono 200 milisekund, z opóźnieniem wynoszącym 50 milisekund. Kiedy użytkownik odsunie kursor myszy znad rozwijanego menu albo znad elementu rodzica (`li`), po opóźnieniu wynoszącym 1 sekundę nastąpi czterosekundowe przejście wstecz, podzielone na osiem kroków.

Jeśli mamy tylko jedno przejście, umieszczamy je w globalnym stanie, *od* którego wychodzimy (źródłowym), bo chcemy, aby przejście to nastąpiło *do* jakiegoś stanu — na przykład związanego ze wskazaniem elementu kursorem myszy albo ze zmianą klasy. Ponieważ chcemy, aby przejście nastąpiło w przypadku dowolnej zmiany, zwykle umieszczamy deklarację tego przejścia w początkowym, domyślnym (najmniej sprecyzowanym) bloku reguł. Jeśli zależy Ci na pełniejszej kontroli nad sytuacją i chciałbyś uzyskać różne efekty, w zależności od kierunku przejścia, pamiętaj o umieszczeniu deklaracji przejść we wszystkich potrzebnych klasach i stanach interfejsu.

Wystrzegaj się definiowania przejść jednocześnie na elementach przodków i elementach potomnych. Deklarowanie przejść mających się zacząć wkrótce po zajściu jakiejś zmiany, która spowoduje innego rodzaju przejście na węzłach przodków albo węzłach potomnych, może mieć nieoczekiwane rezultaty. Jeżeli przejście na elementach potomnych zakończy się przed dokończeniem przejścia na elementach przodków, element potomny ponownie odziedziczy (wciąż będącą w trakcie przejścia) wartość właściwości rodzica. I niekoniecznie musi to być spodziewany efekt.

Zmiana kierunku przerwanych przejść

Jeśli przejście zostanie przerwane przed jego ukończeniem (na przykład w wyniku odsunięcia kursora myszy znad przykładowego rozwijanego menu, zanim skończy się ono otwierać), właściwości są zerowane do wartości, które miały one przed rozpoczęciem przejścia, i następuje przejście powrotne do tych wartości. Ponieważ powtórzenie tego samego czasu trwania i funkcji tempa w odniesieniu do powrotnego, częściowego przejścia może prowadzić do dziwacznych rezultatów i być niewygodne dla użytkownika, specyfikacja przejść CSS przewiduje skrócenie przerwanych animacji powrotnych.

W naszym przykładowym menu mamy zadeklarowaną właściwość `transition-delay` o wartości `50s` dla stanu domyślnego i żadnych właściwości przejść dla stanu `:hover`. To oznacza, że przeglądarka odczeka 50 milisekund przed rozpoczęciem odwrotnego (zamykającego) przejścia.

Kiedy przejście biegnące do przodu zakończy się osiągnięciem wartości końcowych i wyzwolone zostanie zdarzenie `transitionend`, wszystkie przeglądarki powielą właściwość `transition-delay` dla stanu odwrotnego.

Jak widać w tabeli 17.2, jeśli przejście się nie skończy — na przykład użytkownik odsunie kursor myszy znad nawigacji, zanim przejście to zostanie dokończone — wszystkie przeglądarki oprócz Microsoft Edge powtórzą opóźnienie w przeciwnym kierunku. Niektóre przeglądarki powielą też właściwość `transition-duration`, ale w Edge i w Firefoxie zostały zaimplementowane zgodne ze specyfikacją współczynniki skracające animację powrotną.

Tabela 17.2. Obsługa niedokończonych przejść w różnych przeglądarkach

Przeglądarka	Opóźnienie przejścia powrotnego	Czas przejścia	Czas łączny
Chrome 37	Tak	200 ms	0,200 s
Chrome 42	Tak	200 ms	0,250 s
Safari 8	Tak	200 ms	0,200 s
Firefox 41	Tak	38 ms	0,038 s
Opera 32	Tak	200 ms	0,250 s
Edge 12	Nie	38 ms	0,038 s

Przypuśćmy, że użytkownik odsunie kursor znad menu po 75 milisekundach od chwili rozpoczęcia przejścia. To oznacza, że rozwijane menu zacznie się zamykać, jeszcze zanim będzie w pełni otwarte i nieprzezroczyste. Przeglądarka powinna zastosować trwające 50 milisekund opóźnienie przed zamknięciem menu, na tej samej zasadzie, na jakiej odczekała 50 milisekund przed jego otwarciem.

Akurat to zachowanie można uznać za poprawne, bo daje ono ułamek sekundy opóźnienia przed zamknięciem, pozwalający uniknąć skokowej animacji menu, jeśli użytkownik odsunie kursor tylko przypadkiem. Jak widać w tabeli 17.2, postępują tak wszystkie przeglądarki oprócz Microsoft Edge.

Choć daliśmy przeglądarce tylko 75 milisekund na częściowe otwarcie rozwijanego menu przed jego zamknięciem, przywrócenie stanu początkowego w niektórych przeglądarkach potrwa 200 milisekund — pełną wartość właściwości `transition-duration`. Inne przeglądarki, w tym Firefox i Edge, mają zaimplementowany zgodny ze specyfikacją CSS mechanizm skracania przejścia zwrotnego i korekty wartości początkowej tego przejścia. Jeśli mechanizm ten działa prawidłowo, czas potrzebny do ukończenia częściowego przejścia powrotnego będzie podobny do początkowej wartości czasu, ale niekoniecznie taki sam.

W przypadku krokowej funkcji tempa przeglądarki Firefox i Edge zastosują taki czas przejścia powrotnego, jaki wynika z zaokrąglenia czasu potrzebnego do wykonania tylu kroków, ile zostało wyświetlonych przed przerwaniem animacji. Przypuśćmy, że przejście miało trwać 10 sekund i składać się z 10 kroków, a przerwa (i związana z nią zmiana wartości właściwości na początkowe) nastąpiła po 3,25 sekundy — w efekcie przejście zakończyło się w jednej czwartej etapu między trzecim a czwartym krokiem (3 kroki ukończone, co odpowiada 30% całego przejścia). W tej sytuacji powrót do pierwotnych wartości zajmie 3 sekundy. W poniższym, ilustrującym tę sytuację przykładzie szerokość elementu `div` wzrosłaby do 130 pikseli, zanim (po odsunięciu kursora myszy) nastąpiłby powrót do szerokości 100 pikseli:

```
div {
    width: 100px;
    transition: width 10s steps(10, start);
}
div:hover {
    width: 200px;
}
```

O ile jednak czas powrotnego przejścia zostanie zaokrąglony do czasu, jaki upłynął do osiągnięcia ostatniego zakończonego kroku, to przejście w tym kierunku zostanie podzielone na *początkowo* zadeklarowaną liczbę kroków, a nie liczbę kroków zakończonych. W naszym przykładzie z przerwaniem przejścia po 3,25 sekundy przejście powrotne będzie trwało 3 sekundy, ale obejmie wszystkie 10 kroków. Te powrotne kroki będą krótsze — każdy z nich będzie trwał 300 milisekund i każdy spowoduje zwężenie elementu o 3 piksele, a nie o 10.

Jeśli animowalibyśmy „duszka" przy użyciu przejścia właściwości `background-position` ▶, to takie rozwiązanie wyglądałoby fatalnie. Być może specyfikacja oraz jej implementacje ulegną zmianie, aby przejście w przeciwnym kierunku składało się z tej samej liczby kroków co niepełne przejście w przód.

W niektórych przeglądarkach w opisanej sytuacji przejście powrotne trwałoby 10 sekund — przez ten czas nastąpiłoby cofnięcie 3 wykonanych kroków, ale podzielone na 10 kroków. To oznacza, że przykładowy element zwężałby się w 3-pikselowych krokach, ale co sekundę. Chodzi tu o przeglądarki, w których nie został zaimplementowany mechanizm skracania przejścia powrotnego — przejście to potrwa wówczas pełne 10 sekund zamiast 3 sekund, a powrót z 30% początkowej zmiany zostanie podzielony na 10 kroków. Bez względu na to, czy początkowe przejście zostało ukończone, czy nie, w przeglądarkach tych zostanie zastosowany cały czas trwania przejścia początkowego, ewentualnie pomniejszony o wartość bezwzględną ujemnej wartości `transition-delay`, jeśli została ona zadeklarowana. Stanie się tak bez względu na rodzaj funkcji tempa. W przypadku tempa krokowego w opisanym przed chwilą przykładzie przejście powrotne zajmie więc 10 sekund. W przykładzie z rozwijanym menu przejście powrotne potrwa 200 milisekund, bez względu na to, czy menu zostało w pełni rozwinięte, czy nie.

W przeglądarkach, w których mechanizmy korygowania czasu przejścia powrotnego funkcjonują prawidłowo, jeśli wybrana została liniowa funkcja tempa, czas trwania przejścia będzie taki sam w obu kierunkach. Jeśli wybrana została krokowa funkcja tempa, czas przejścia powrotnego będzie równy czasowi, jaki zajęło osiągnięcie ostatniego ukończonego kroku. W przypadku wszystkich pozostałych funkcji `cubic-bezier` czas trwania będzie proporcjonalny do postępu początkowego przejścia przed jego przerwaniem (*https://drafts.csswg.org/css-transitions/transition-reversing-demo*). Ujemne wartości właściwości `transition-delay` również są proporcjonalnie skracane. Dodatnie wartości opóźnień pozostają bez zmian w obu kierunkach.

W przypadku niedokończonego przejścia w żadnej przeglądarce nie zostanie wyzwolone zdarzenie `transitionend` dla stanu `:hover`, ale we wszystkich zdarzenie `transitionend` zostanie wyzwolone dla stanu powrotnego — w chwili, gdy menu skończy się zwijać. Rzeczywisty czas trwania przejścia powrotnego (`elapsedTime`) będzie uzależniony od tego, czy przeglądarka zamknie menu w czasie pełnych 200 milisekund, czy raczej potrwa to tyle, ile zajęło częściowe otwarcie tego menu.

Aby zmodyfikować te wartości, zadeklaruj właściwości przejścia zarówno dla stanu początkowego, jak i końcowego (na przykład dla stanów wskazania i niewskazania elementu kursorem myszy). Choć nie będzie to miało wpływu na skrócenie przejścia powrotnego, da nieco większą kontrolę nad jego przebiegiem.

Animowane właściwości i wartości

Zanim zajmiesz się tworzeniem przejść i animacji, powinieneś zwrócić uwagę na fakt, że nie wszystkie właściwości da się animować. Przejścia (i animacje) można tworzyć tylko w przypadku dających się animować właściwości CSS… tylko które to są właściwości?

Listę tych właściwości zamieściliśmy w dodatku A. CSS jednak nieustannie się rozwija i lista animowanych właściwości (*https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animated_properties*) zapewne się rozszerzy.

Jednym ze sposobów na domyślenie się, które właściwości można animować, jest sprawdzenie, które z nich przyjmują wartości dające się interpolować. *Interpolacja* to proces wyliczania punktów danych wypadających pomiędzy znanymi punktami danych. Najważniejszym warunkiem decydującym o zdolności danej właściwości do animacji jest możliwość interpolacji *wartości wyliczonej*. Jeśli wartością wyliczoną właściwości są słowa kluczowe, to nie dadzą się one interpolować; chyba że słowa te mogą być zamienione na jakiegoś rodzaju wartości numeryczne. Podstawowa zasada jest taka, że jeżeli da się wyznaczyć wartość pośrednią między dwiema wartościami jakiejś właściwości, to właściwość ta zapewne daje się animować.

Na przykład wartości właściwości `display`, takie jak `block` i `inline-block`, nie są numeryczne i nie można określić między nimi wartości pośredniej — a to oznacza, że nie da się ich animować. Wartości właściwości `transform`, takie jak `rotate(10deg)` oraz `rotate(20deg)`, mają wartość pośrednią — tu w postaci `rotate(15deg)` — a zatem da się je animować.

Właściwość `border` jest skrótem właściwości `border-style`, `border-width` i `border-color` (które z kolei same są skrótami właściwości decydujących o obramowaniach czterech boków elementu). Choć nie da się wyznaczyć wartości pośredniej między dwiema dowolnymi wartościami `border-style`, to wartości długości `border-width` są numeryczne, da się je więc animować. Słowa kluczowe `medium`, `thick` i `thin` mają odpowiedniki liczbowe i da się je interpolować: wyliczona wartość właściwości `border-width` jest w istocie wartością długości odpowiadającą tym słowom.

Kolory w przypadku właściwości `border-color` są wartościami liczbowymi (nawet kolory nazwane mają odpowiedniki w postaci liczb szesnastkowych), a zatem kolory także da się animować. Jeśli zadeklarujesz przejście od `border: red solid 3px` do `border: blue dashed 10px`, szerokość obramowania i jego kolory będą animowane zgodnie z podaną prędkością, ale właściwość `border-style` przeskoczy z wartości `solid` na `dashed` od razu po rozpoczęciu przejścia (po ewentualnym opóźnieniu).

Jak już wspomnieliśmy (zob. też dodatek A), wartości numeryczne na ogół dają się animować. Słowa kluczowe, których nie da się przełożyć na wartości numeryczne, zwykle nie dają się animować. Funkcje CSS, które jako parametry przyjmują wartości liczbowe, zazwyczaj także mogą być animowane. Jedynym wyjątkiem od tej reguły jest `visibility` — choć nie ma punktu pośredniego między

wartościami `visible` i `hidden`, to wartości właściwości `visibility` dają się interpolować pomiędzy „widocznością" a „niewidocznością". W przypadku właściwości `visibility` albo wartość początkowa, albo końcowa musi być jednak równa `visible`, bo w przeciwnym razie interpolacja nie będzie możliwa. W przypadku przejścia z wartości `visible` na `hidden` zmiana dokona się na końcu animacji. W przypadku przejścia z `hidden` na `visible` zmiana nastąpi na początku animacji.

Wartość `auto` na ogół należy traktować jako nieanimowaną i powinno się jej unikać w przypadku animacji oraz przejść. Zgodnie ze specyfikacją nie jest to wartość animowana, ale niektóre przeglądarki interpretują bieżącą wartość numeryczną `auto` (taką jak `height: auto`) jako `0px`. Wartości `auto` nie da się animować w przypadku właściwości takich jak `height`, `width`, `top`, `bottom`, `left`, `right` i `margin`.

Często da się jednak zastosować alternatywną właściwość albo wartość. Na przykład zamiast tworzyć przejście z `height: 0` na `height: auto` zrób przejście z `max-height: 0` na `max-height: 100vh`, co na ogół powinno dać oczekiwany rezultat. Wartość `auto` da się animować w przypadku właściwości `min-height` i `min-width`, bo `min-height: auto` ma wyliczoną wartość 0.

Na czym polega interpolacja wartości właściwości?

Interpolacja jest możliwa wtedy, gdy da się wyliczyć wartości pośrednie między dwiema lub większą liczbą znanych wartości. Wartości dające się interpolować można stosować w przejściach i animacjach.

Liczby rzeczywiste są interpolowane jako wartości zmiennoprzecinkowe. Liczby całkowite są po interpolacji zaokrąglane do pełnych wartości (wzrost albo spadek następuje w postaci wartości całkowitych).

W CSS jednostki długości i wartości procentowe są przeliczane na liczby rzeczywiste. W przypadku przejść albo animacji z użyciem funkcji `calc()` bądź przejść z jakiejś wartości długości na wartość procentową (lub na odwrót) wartości te zostaną przekształcone na funkcję `calc()` i interpolowane jako liczby rzeczywiste.

Kolory — zarówno w postaci HSLA, RGB, jak i kolorów nazwanych (jak `aliceblue`) — są na potrzeby przejść przeliczane na odpowiedniki RGBA i interpolowane w przestrzeni kolorów RGBA.

W przypadku animowania grubości znaków, jeśli użyjesz słów kluczowych takich jak `bold`, słowa te zostaną przekształcone na wartości numeryczne i animowane w krokach będących wielokrotnościami wartości 100. Być może zmieni się to w przyszłości, jeśli grubości znaków będzie się dało wyrażać w dowolnych liczbach całkowitych — w takim przypadku będą one interpolowane jako wartości całkowite, a nie wielokrotności 100.

Jeśli zastosujesz animowaną właściwość składającą się z kilku komponentów, to każdy z nich będzie interpolowany zgodnie ze specyfiką tego komponentu. Na przykład właściwość `text-shadow` może się składać z czterech komponentów: koloru oraz wartości `x`, `y` i `blur`. Kolor jest interpolowany jako właściwość `color`, składniki `x`, `y` oraz `blur` są interpolowane jako wartości długości. Cienie typu `box-shadow` mają dwie dodatkowe, opcjonalne właściwości: akcent `inset` (lub jego brak) oraz `spread`. Właściwość `spread`, jako wartość długości, jest interpolowana właśnie jako długość. Słowo kluczowe `inset` nie ma odpowiednika liczbowego: możesz dokonać przejścia między jednym cieniem z właściwością `inset` a innym albo między jednym wieloskładnikowym cieniem a drugim, ale zasadniczo nie ma sposobu na uzyskanie płynnego przejścia między cieniem `inset` a zwykłym.

Podobnie jak ma to miejsce w przypadku wartości wieloskładnikowych, przejścia z użyciem gradientów są możliwe tylko w przypadku gradientów tego samego typu (liniowych lub kołowych), o równej liczbie znaczników koloru. Kolory każdego znacznika są interpolowane jako wartości koloru, a położenie każdego znacznika jest interpolowane jako długość albo wartość procentowa.

Interpolowanie powtarzających się właściwości

W przypadku list zawierających różnego rodzaju właściwości każda pozycja na liście jest interpolowana w sposób właściwy dla jej rodzaju — jeśli tylko na listach jest ta sama liczba pozycji (lub dające się cyklicznie powtórzyć pozycje), a każdą parę wartości można interpolować:

```
.img {
    background-image:
        url(1.gif), url(2.gif), url(3.gif), url(4.gif),
        url(5.gif), url(6.gif), url(7.gif), url(8.gif),
        url(9.gif), url(10.gif), url(11.gif), url(12.gif);
    background-size: 10px 10px, 20px 20px, 30px 30px, 40px 40px;
    transition: background-size 1s ease-in 0s;
}
.img:hover {
    background-size: 25px 25px, 50px 50px, 75px 75px, 100px 100px;
}
```

Weźmy przykładowe przejście dotyczące czterech rozmiarów obrazu w tle (`background-size`). Każda para wartości na obu listach została podana w pikselach, można więc na przykład dokonać płynnego przejścia od trzeciej wartości `background-size` sprzed przekształcenia do trzeciej wartości `background-size` z listy po przekształceniu. W podanym przykładzie szerokość i wysokość obrazów numer 1, 6 i 10 po wskazaniu tych obrazów kursorem myszy zostanie zmieniona z `10px` na `25px`. Na tej samej zasadzie obrazy numer 3, 7 i 11 zmienią rozmiar z `30px` na `75px` i tak dalej.

To oznacza, że wartości właściwości `background-size` zostaną powtórzone trzykrotnie, tak jakbyśmy napisali kod CSS w następujący sposób:

```
.img {
    ...
    background-size: 10px 10px, 20px 20px, 30px 30px, 40px 40px,
                     10px 10px, 20px 20px, 30px 30px, 40px 40px,
                     10px 10px, 20px 20px, 30px 30px, 40px 40px;
    ...
}
.img:hover {
    background-size: 25px 25px, 50px 50px, 75px 75px, 100px 100px,
                     25px 25px, 50px 50px, 75px 75px, 100px 100px,
                     25px 25px, 50px 50px, 75px 75px, 100px 100px;
}
```

Jeśli liczba rozdzielonych przecinkami wartości jest mniejsza niż liczba obrazów w tle, to lista wartości jest powtarzana od początku, aż wartości wystarczy — nawet jeśli lista dla stanu `:hover` nie jest zgodna z listą dla stanu początkowego:

```
.img:hover {
    background-size: 33px 33px, 66px 66px, 99px 99px;
}
```

Jeśli zażyczylibyśmy sobie przejścia od czterech deklaracji `background-size` dla stanu początkowego do trzech deklaracji `background-size` dla stanu `:hover` (wszystkie wartości w pikselach) i nadal mieli-

byśmy 12 obrazów w tle do przetworzenia, to wartości dla stanu `:hover` i dla stanu początkowego zostałyby powtórzone (odpowiednio, trzykrotnie i czterokrotnie) aż do uzyskania 12 potrzebnych wartości. Byłoby to odpowiednikiem poniższej deklaracji:

```
.img {
    ...
    background-size: 10px 10px, 20px 20px, 30px 30px,
                     40px 40px, 10px 10px, 20px 20px,
                     30px 30px, 40px 40px, 10px 10px,
                     20px 20px, 30px 30px, 40px 40px;
    ...
}
.img:hover {
    background-size: 33px 33px, 66px 66px, 99px 99px,
                     33px 33px, 66px 66px, 99px 99px,
                     33px 33px, 66px 66px, 99px 99px,
                     33px 33px, 66px 66px, 99px 99px;
}
```

Jeśli dla jakiejś pary wartości nie da się określić wartości pośrednich — na przykład jeżeli właściwość `background-size` zmienia się z `contain` (w stanie domyślnym) na `cover` (w stanie `:hover`) — to zgodnie ze specyfikacją list z takimi wartościami nie da się interpolować. Niektóre przeglądarki ignorują jednak tę konkretną parę wartości na potrzeby przejścia i wciąż animują pozostałe wartości — te, które da się interpolować.

Niektóre właściwości da się animować, jeśli przeglądarka może uzupełnić brakujące, niezadeklarowane jawnie wartości. Na przykład w przypadku cieni przeglądarka zastosuje regułę `box-shadow: transparent 0 0 0` albo `box-shadow: inset transparent 0 0 0`, zastępując dowolne wartości, które nie zostaną w sposób jawny zadeklarowane dla stanu przed przejściem albo po nim. Przykłady tego rodzaju znajdziesz na stronie z materiałami pomocniczymi do tej książki (*https://meyerweb.github.io/csstdg4figs/*).

Tylko dające się interpolować wartości wyzwalają zdarzenia `transitionend`.

Jak już wspomnieliśmy, właściwość `visibility` jest animowana inaczej od pozostałych: jeśli animujemy lub tworzymy przejście do stanu `visible` (albo od niego), to proces ten jest traktowany jako pojedynczy krok. Element jest zawsze widoczny w trakcie przejścia albo animacji, jeśli rezultat funkcji tempa mieści się w zakresie między 0 a 1. W przypadku przejścia z `hidden` na `visible` do przełączenia wartości dojdzie na początku tego przejścia. W przypadku przejścia z `visible` na `hidden` do przełączenia wartości dojdzie na końcu tego przejścia. Należy pamiętać, że procesem tym da się kierować za pomocą funkcji tempa.

Jeśli przez przypadek zastosujesz w przejściu właściwość, której nie da się animować, nie przejmuj się. Cała deklaracja nie zostanie zignorowana. Przeglądarka po prostu nie dokona przejścia właściwości, której nie da się animować. Warto przy tym podkreślić, że niedające się animować (albo nieistniejące) właściwości CSS nie są wtedy zupełnie pomijane. Przeglądarka uwzględnia nierozpoznane albo nieanimowane właściwości, aby zachować ich miejsce na liście i dzięki temu zadbać o to, by kolejne rozdzielone przecinkami parametry przejść nie zostały powiązane z nieodpowiednimi właściwościami[2].

[2] Może się to zmienić. CSS Working Group rozważa umożliwienie animowania wszystkich właściwości. W razie braku możliwości określenia punktu pośredniego między wartościami sprzed przejścia i po nim miałoby to polegać na jednorazowym przeskoku między tymi wartościami w połowie przebiegu funkcji tempa.

Przejścia mogą dotyczyć tylko tych właściwości, które w danej chwili nie są animowane z poziomu CSS. Jeśli dany element jest animowany, to związane z nim przejścia mogą nastąpić, ale tylko wtedy, gdy nie dotyczą one właściwości sterowanych w danej chwili przez animację. O animacjach CSS przeczytasz w rozdziale 18.

Wyjścia awaryjne: przejścia to tylko ozdobniki

Obsługa przejść w przeglądarkach stoi na znakomitym poziomie. Wszystkie przeglądarki, w tym Safari, Chrome, Opera, Firefox, Edge i Internet Explorer (od wersji IE10), obsługują przejścia CSS.

Przejścia są ozdobnikami interfejsu użytkownika, ewentualny brak pełnej obsługi przejść nie powinien Cię więc zniechęcać do ich stosowania. Nawet jeśli jakaś przeglądarka nie obsługuje przejść CSS, to zmiany, które chcesz wprowadzić za pomocą przejścia, i tak zajdą — po prostu „przejście" ze stanu początkowego do końcowego nastąpi raptownie, w chwili przeliczenia parametrów stylu.

Użytkownicy mogą nie zobaczyć atrakcyjnego (a czasami irytującego…) efektu, ale nie odbierze im to możliwości zapoznania się z treścią strony.

Ponieważ przejścia są zasadniczo ozdobnikami, nie ma potrzeby sztucznego implementowania ich w „zabytkowych" wersjach przeglądarki IE. Choć za pomocą JavaScriptu da się zaimplementować przejścia w IE9 lub starszych wersjach Internet Explorera, a w przypadku Androida 4.3 (i starszych) używać prefiksów właściwości przejść, potrzeba stosowania tego rodzaju obejść jest znikoma.

Drukowanie przejść

Przy drukowaniu strony albo aplikacji internetowej wykorzystywany jest arkusz stylów przystosowany do druku. Jeśli zastosowałeś style obsługujące jedynie atrybut `screen`, to CSS nie będzie miał żadnego wpływu na wygląd wydruku.

Często atrybuty mediów są pomijane, co stanowi odpowiednik domyślnej deklaracji `media="all"`. W zależności od przeglądarki próba wydrukowania elementu, dla którego zastosowano przejście, spowoduje albo zignorowanie interpolowanych wartości, albo wydruk elementu zgodnie z bieżącym stanem wartości jego właściwości.

Oczywiście na wydruku nie da się zobaczyć przejścia elementu, ale w niektórych przeglądarkach, takich jak Chrome, jeśli dany element jest w trakcie przejścia z jednego stanu do drugiego, to na wydruku zostanie pokazany bieżący stan tego elementu w chwili wywołania funkcji `print` (jeśli tylko dana właściwość jest drukowalna). Na przykład jeśli zmienia się kolor tła, to na wydruku nie będzie widać ani koloru sprzed przejścia, ani po przejściu, bo kolory tła zasadniczo nie są drukowane w ogóle. Jeśli jednak to kolor tekstu uległ zmianie z jednej wartości na inną, to na kolorowej drukarce albo w pliku PDF zobaczymy barwę zgodną z bieżącą wartością właściwości `color`.

W innych przeglądarkach, takich jak Firefox, o tym, czy na wydruku pojawi się wartość sprzed rozpoczęcia przejścia, czy wartość po zakończeniu przejścia, decyduje sposób inicjalizacji przejścia. Jeśli jest ono inicjowane przez stan `:hover`, to na wydruku pojawi się wartość nieuwzględniająca tego stanu, bo podczas korzystania z okna dialogowego wydruku kursor myszy nie będzie już raczej

wskazywał aktywnego elementu. Jeśli przejście polega na dodaniu klasy, to na wydruku pojawi się wartość po przejściu, nawet jeśli nie zostało ono ukończone. W takich sytuacjach wydruk następuje z pominięciem właściwości przejścia.

Ponieważ możemy zastosować osobny arkusz stylów lub reguł `@media` dla druku, przeglądarki przeliczają każdy arkusz osobno. W przypadku reguł przeznaczonych do druku style się nie zmieniają, nie ma więc mowy o przejściach. W takich przypadkach drukowanie działa tak, jakby wartości zmieniły się w sposób nagły, a nie stopniowo, z upływem czasu.

Skorowidz

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

Ż

O'REILLY®

CSS. Poznaj i stosuj najnowsze specyfikacje!

CSS służy do opisywania sposobu prezentowania treści internetowych wyświetlanych na ekranach, na potrzeby druku czy syntezatorów mowy. Jest obsługiwany przez wszystkie przeglądarki i urządzenia, takie jak smartfony, komputery, gry wideo, telewizory, zegarki, kioski multimedialne czy konsole samochodowe. CSS pozwala zwiększyć wygodę użytkownika, przyspieszyć proces projektowania, uniknąć potencjalnych błędów, a także urozmaicić i ożywić aplikacje. Język ten wciąż się rozwija i od wielu lat stanowi niezbędny element warsztatu każdego profesjonalnego projektanta stron internetowych.

Ta książka to kompleksowy i zaktualizowany przewodnik po implementacji CSS. Zawiera obszerną analizę najnowszych specyfikacji CSS. Przedstawiono tu szereg istotnych zagadnień i wyrafinowanych technik stylizowania stron oraz poprawiania dostępności treści — wykorzystanie tych sposobów pomaga zaoszczędzić czas i wysiłek. Ta publikacja powstała z myślą o profesjonalnych projektantach stron, niemniej jednak bardzo przyda się osobom, które zaczynają naukę CSS i chcą od razu wykorzystywać jego najlepsze cechy. Znalazł się tu szczegółowy opis wszystkich funkcji CSS powszechnie obsługiwanych przez przeglądarki, w tym funkcji, które w czasie pisania tej książki były przygotowywane do wprowadzenia.

Wybrane zagadnienia:

- Selektory, specyficzność i kaskada
- Właściwości tekstu, dopełnienia, marginesy, tła i gradienty
- Układy, w tym *flexbox* i *grid*
- Przekształcenia 2D i 3D, przejścia i animacje
- Filtry: mieszanie, przycinanie i maskowanie

Eric A. Meyer — to niekwestionowany autorytet w dziedzinie HTML, CSS i standardów internetowych. Założył firmę Complex Spiral Consulting. Do najważniejszych inicjatyw, w które się zaangażował, należą ruch na rzecz promowania mikroformatów oraz seria konferencji An Event Apart.

Estelle Weyl — jest promotorką otwartych standardów internetowych i inżynierem społeczności. Od 1999 roku zajmuje się projektowaniem stron internetowych, zwracając uwagę na ich zgodność ze standardami i dostępność. Często zabiera głos na konferencjach na całym świecie.

ISBN 978-83-283-4083-1