KAPITOLA 3

Práce s audiem a videem v HTML5

V této kapitole se seznámíme se dvěma velmi důležitými elementy jazyka HTML5, `audio` a `video`, a ukážeme si jejich použití při tvorbě podmanivých aplikací. Elementy audio a video rozšiřují multimediální možnosti aplikací HTML5 o možnost použití audia a videa, bez potřeby zásuvných modulů, pomocí jednotného, integrovaného rozhraní.

Nejdříve probereme kontejnery pro audio a video a poté si představíme stávající kodeky a vysvětlíme si, proč se používají ty, které se používají. Dále budeme pokračovat problematikou nejednotné podpory kodeků, která dnes představuje asi největší překážku v použití multimediálních elementů, a vysvětlíme si, proč by to v budoucnosti už nemusel být takový problém. Ukážeme si také techniku zajišťující zobrazení toho nejvhodnějšího typu obsahu v prohlížeči.

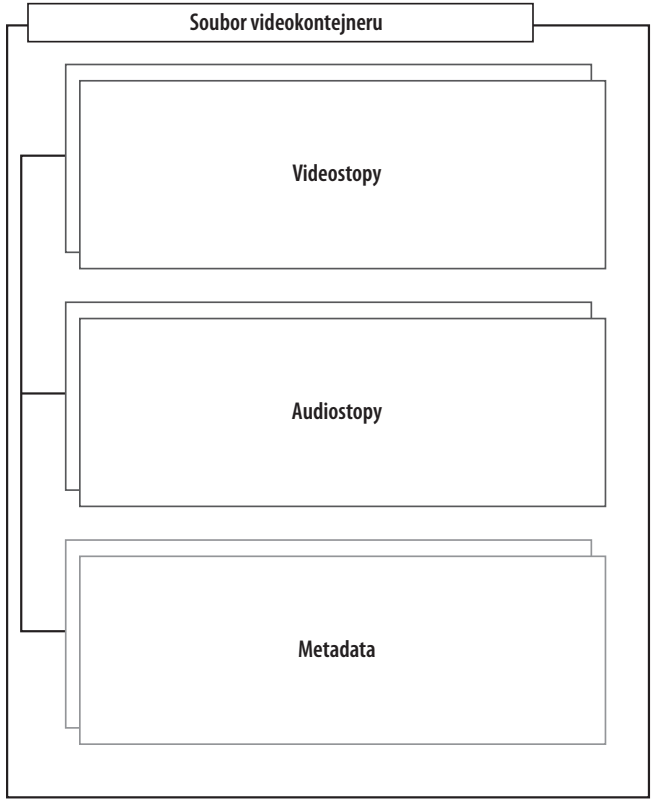
Dále si ukážeme, jak ovládat audio a video pomocí aplikačního rozhraní HTML5 a v neposlední řadě také použití tohoto rozhraní ve vašich aplikacích.

Seznámení s audio- a videorozhraním HTML5

V následující části kapitoly probereme některé z klíčových principů týkajících se použití audio- a videorozhraní HTML5 – konkrétně kontejnery a kodeky.

Kontejnery videa

Audio- či videosoubor jsou ve skutečnosti pouze *kontejnerem*, podobně jako archiv ZIP, který obsahuje další soubory. Obrázek 3.1 ukazuje videosoubor (videokontejner) obsahující audiostopy, videostopy a metadata. Audio- a videostopy se v reálném čase zkombinují při přehrávání videa. Metadata obsahují informace o videu, jako je například jeho náhled, titul, podtitul a další.



Obrázek 3.1: Obsah videokontejneru

Mezi populární formáty videokontejnerů patří například následující:

- Audio Video Interleave (.avi)
- Flash Video (.flv)
- MPEG 4 (.mp4)
- Matroska (.mkv)
- Ogg (.ogv)

Kodeky audia a videa

Kodéry a dekodéry (kodeky) audia a videa jsou algoritmy používané pro kódování a dekódování konkrétní audio- či videostopy nezbytné pro její přehrání. Multimediální soubory v původním stavu bez kódování mají enormní velikost. Audio- či videoklip by představoval ohromné množství dat, která by v přiměřeném čase kvůli jeho velikosti nebylo možné přenášet po Internetu. Bez dekodéru by příjemce nebyl schopen ze zakódované podoby rekonstruovat původní multimediální obsah. Kodek rozumí konkrétnímu formátu kontejneru a dekóduje audio- a videostopy, které obsahuje.

Mezi známé audiokodeky patří například tyto:

- AAC
- MPEG-3
- OggVorbis

Z videokodeků jmenujme následující:

- H.264
- VP8
- OggTheora

VÁLKY KODEKŮ A NEJISTÉ PŘÍMĚŘÍ

Některé z kodeků podléhají patentům, zatímco jiné jsou volně dostupné. Například audiokodek Vorbis a videokodek Theora jsou volně dostupné, zatímco kodeky MPEG-4 a H.264 podléhají licenčním poplatkům.

Specifikace HTML5 původně měla v úmyslu vyžadovat podporu určitých kodeků. Někteří výrobci však nebyli ochotní zahrnout také kodek OggTheora, protože nebyl součástí jejich stávající hardwarové a softwarové výbavy. Například iPhone společnosti Apple obsahuje hardwarový dekodér pro H.264, nikoli však pro kodek Theora. Naproti tomu volně dostupné systémy nemohou podporovat proprietární placené kodeky, aniž by to ovlivnilo jejich distribuci. Výkon některých proprietárních kodeků je navíc faktorem, který výrazně ovlivňuje přijetí volně dostupných kodeků prohlížeči. Tato situace nás nakonec dovedla do slepé uličky. V tuto chvíli neexistuje ani jeden kodek, který by byli ochotní implementovat všichni výrobci prohlížečů.

Prozatím byl požadavek na podporu určitých kodeků ze specifikace vypuštěn. Toto rozhodnutí se však může v budoucnu změnit. V tuto chvíli počítejte s rozdílnou podporou ze strany prohlížečů a s nutností změny kódování multimédií pro různá prostředí (což pravděpodobně děláte už nyní).

Časem se podpora různých kodeků zlepší a ustálí, což výrazně usnadní výběr typu kódování. Není vyloučené, že se některý z kodeků stane defacto standardem Internetu. Multimediální elementy disponují mechanizmem přepínání na nejvhodnější typ obsahu pro daný prohlížeč, aby usnadnily podporu různých prostředí.

Na pomoc přichází WebM

Frank říká: „Google představil v květnu 2010 videoformát WebM. WebM je nový formát pro audio a video, který si klade za cíl vyřešit aktuální svízelnou situaci na Webu. Soubory ve formátu WebM mají příponu `.webm` a obsahují videostopy VP8 a audiostopy OggVorbis, v kontejneru podobném Matrosce. Google uvolnil specifikaci WebMa softwaru pod liberální licencí zahrnující zdrojové kódy a patenty. Coby vysoce kvalitní formát, který je pro vývojáře i vydavatele zdarma, představuje WebM výrazný posun kupředu v oblasti kodeků.

Co se podpory ze strany prohlížečů týká, Firefox, Opera a Chrome budou formát WebM podporovat. Opera 10.60 již dokonce podporu WebM nabízí. Mozilla a Google se zavázali k implementaci WebM v dalších verzích svých prohlížečů. Microsoft v Internet Exploreru 9 podporu neimplementoval, ale umožňuje ji po doinstalování zásuvného modulu."

Omezení audia a videa

Několik věcí specifikace audio- a videorozhraní HTML5 nepodporuje:

- *Streaming* audia a videa. V tuto chvíli neexistuje v HTML5 žádný standard pro změnu bitratu videa v HTML5. Aktuální implementace podporuje pouze úplné multimediální soubory. Určité aspekty specifikace však do budoucna počítají s podporou streamování médií, poté co se vyřeší otázka podpory formátů.
- Media jsou omezená zásadami sdílení prostředků různých původů. Více informací o této problematice najdete v kapitole 5.
- Zobrazení videa přes celou obrazovku není možné vyžádat na straně skriptu. Považuje se za bezpečnostní riziko umožnit skriptům převzít kontrolu nad celou obrazovkou. Prohlížeče však mohou nechat uživatele zobrazit video přes celou obrazovku pomocí dodatečných ovládacích prvků.
- Přístupnost není u audio- a videoelementů dosud úplně specifikovaná. Pracuje se na specifikaci s názvem WebSRT, přidávající podporu titulků vycházející z populárního formátu SRT.

Podpora audia a videa HTML5 v prohlížečích

Jak ukazuje tabulka 3.1, elementy `audio` a `video` jazyka HTML5 již v tuto chvíli podporuje řada prohlížečů. Tabulka uvádí také podporované kodeky.

Tabulka 3.1: Podpora HTML5 videa prohlížeči

Prohlížeč	Detaily	Podporované kontejnery a kodeky
Chrome	Verze 3.0 a novější	Theora a Vorbis, Oggkontejner H.264 a AAC, MPEG 4
Firefox	Verze 3.5 a novější	Theora a Vorbis, Oggkontejner
Internet Explorer	Verze 9.0 a novější	H.264 a AAC, MPEG 4 kontejner
Opera	Verze 10.5 a novější	Theora a Vorbis, Oggkontejner (10.5 a novější) VP8 a Vorbis, WebMformát (10.6 a novější)
Safari	Verze 3.2 a novější	H.264 a AAC, MPEG 4 kontejner

Vždy je vhodné nejdříve otestovat, jestli prohlížeč HTML5 audio a video podporuje. Část *Ověření podpory prohlížeče* dále v této kapitole vám ukáže, jak ve vašich aplikacích zkontrolovat podporu tohoto rozhraní prohlížečem.

Použití rozhraní audio a video

V této části kapitoly prozkoumáme možnosti praktického použití rozhraní audio a video HTML5 ve vašich aplikacích. Použití těchto elementů HTML5 namísto dřívějších technik vkládání videa (využívajících zásuvné moduly Flash, QuickTime nebo Windows Media) přináší dvě hlavní výhody, které usnadňují život jak uživatelům, tak vývojářům:

- *Nové elementy pro audio a video eliminují překážky bránící v nasazení kvůli tomu, že jsou součástí prostředí prohlížeče*. Přestože se některé zásuvné moduly těší velké oblibě, v kontrolovaných korporátních prostředích se často blokují. Někteří uživatelé tyto zásuvné moduly deaktivují kvůli jejich častému zneužívání k reklamním účelům, čímž znemožní i přehrávání médií s jejich pomocí. Zásuvné moduly rozšiřují prostor k případnému útoku. Často také mívají problémy s integrací svého výstupu se zbytkem obsahu stránky, čímž u některých typů stránek způsobují problémy s ořezáváním či průhledností. Protože zásuvné moduly používají vlastní vykreslovací jádra, zatímco zbytek webové stránky má na starosti jádro jiné, potýkají se vývojáři s problémy, snaží-li se elementy, jako jsou nabídky nebo jiné vizuální elementy, v rámci stránky překročit hranice zásuvného modulu.
- *Multimediální elementy nabízí v rámci dokumentu jednotné, integrované rozhraní API přístupné ze skriptů*. Jakožto vývojáři vám nové multimediální elementy umožňují ve skriptech velmi snadno řídit přehrávání obsahu. O tom se budete moci přesvědčit v mnoha příkladech dále v této kapitole.

Použití multimediálních elementů s sebou přirozeně přináší jeden zásadní problém, se kterým jsme se seznámili už dříve v této kapitole, a tím je nejednotná podpora kodeků. Dá se však očekávat, že se podpora kodeků do budoucna zlepší a tím se otázka výběru toho správného vyřeší. Multimediální elementy navíc disponují mechanizmem výběru toho nejvhodnějšího typu obsahu pro daný prohlížeč, jak sami brzy uvidíte.

OVĚŘENÍ PODPORY PROHLÍŽEČE

Nejjednodušším způsobem, jak ověřit podporu elementů `audio` a `video` prohlížečem je dynamicky jeden či oba vytvořit a zkontrolovat existenci jedné z jeho metod:

```
var hasVideo = !!(document.createElement('video').canPlayType);
```

Tento jednoduchý kód dynamicky vytvoří element `video` a ověří existenci jeho metody `canPlayType`. Pomocí operátoru `!!` se výsledek převede na pravdivostní hodnotu, která indikuje, jestli se element podařilo vytvořit.

Pokud prohlížeč elementy `audio` a `video` nepodporuje, můžete s použitím specializované knihovny do stránky vložit multimediální elementy s totožným rozhraním, využívající technologie jako je Flash, které podporují i starší prohlížeče.

Do elementů `audio` a `video` můžete také vložit alternativní obsah, který se zobrazí namísto nepodporované značky. Tímto alternativním obsahem může být i volání zásuvného moduluFlash, který zobrazí stejné video v případě, že prohlížeč daný element nepodporuje. Spokojíte-li se se zobrazením textu na nepodporovaných prohlížečích, stačí přidat požadovaný obsah do elementu `video` nebo `audio`, například takto:

```
<video src="video.ogg" controls>
  Váš prohlížeč nepodporuje video HTML5.
</video>
```

Pokud se rozhodnete alternativní obsah použít k zobrazení videa v prohlížečích, které multimediální funkce HTML5 nepodporují, stačí do těla elementu vložit odkaz na externí zásuvný modul zajišťující zobrazení téhož videa:

```
<video src="video.ogg">
  <object data="videoplayer.swf" type="application/x-shockwave-flash">
    <param name="movie" value="video.swf"/>
  </object>
</video>
```

Vložením elementu `object`, který zobrazí video ve formátu Flash, do těla elementu `video` zajistíte preferenci videa HTML5, je-li k dispozici, a zálohu v podobě videa Flash. To naneštěstí vyžaduje, aby bylo na serveru uloženo více verzí videa, dokud se nesjednotí podpora videa HTML5.

Multimédia pro každého

Brian říká: „Zajištěním přístupnosti vaší webové aplikace není jen správné, ale také dobré z obchodního hlediska, a někdy to dokonce vyžaduje zákon. Uživatelům se zrakovým nebo sluchovým postižením by se měl prezentovat alternativní obsah, který vyhovuje jejich potřebám.

Organizace stojící za návrhem HTML5 jsou si velmi dobře vědomy nedostatečné podpory přístupného audia a videa (například titulky pro neslyšící) a právě na ní pracují. V mezičase by měli vývojáři poskytnout přinejmenším odkazy na přepisy a zvážit použití možností rozhraní k zobrazení synchronizovaného textu přímo ve videu anebo v jeho blízkosti.

Mějte na paměti, že se alternativní obsah nacházející se v těle elementu `video`, resp. `audio`, zobrazí pouze tehdy, když prohlížeč tento element vůbec nepodporuje. Nehodí se proto k zajištění přístupnosti v případech, kdy prohlížeč multimédia HTML5 podporuje, ale uživatel ne."

Seznámení s multimediálními elementy

Díky prozíravému návrhu existuje řada podobností mezi elementy `audio` a `video` jazyka HTML5. Oba tyto elementy nabízí mnoho stejných operací – zahájení přehrávání, pauza, ztišení, načtení a další. Z tohoto důvodu popisuje společné chování část specifikace věnovaná elementu `media`. Naše seznámení s multimediálními elementy zahájíme pozorováním toho, co mají společného.

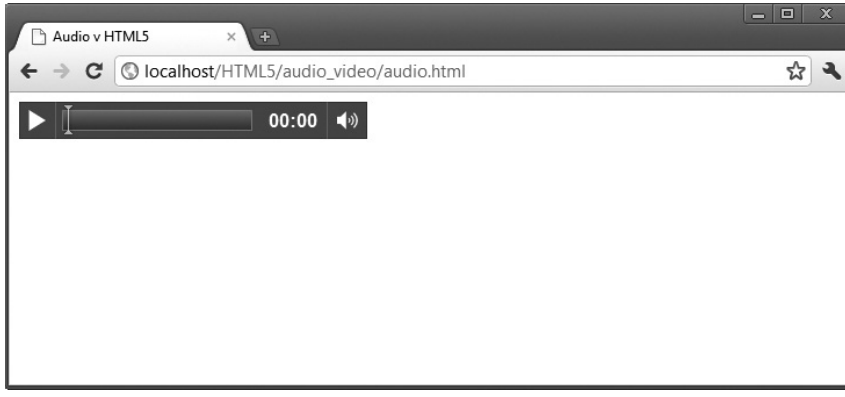
DEKLARACE ELEMENTU

V tomto příkladu použijeme element `audio`, abychom vyzkoušeli běžné vlastnosti médií HTML5. Příklady této části kapitoly budou (nečekaně) plné multimédií, která najdete v ukázkových souborech přiložených k této knize.

Začneme velmi jednoduchým příkladem (soubor s názvem `audio.html`) stránky s audiopřehrávačem, který přehrává uklidňující a velmi dobře známou skladbu od Johanna Sebastiana Bacha s názvem Air.

```
<!DOCTYPE html>
<html>
  <meta charset="utf-8" />
  <title>Audio v HTML5</title>
  <audio controls src="johann_sebastian_bach_air.ogg">
    Skladba od Johanna Sebastiana Bacha.
  </audio>
</html>
```

Tento příklad předpokládá, že se dokument HTML i audiosoubor (v tomto případě `johann_sebastian_bach_air.ogg`) nachází ve stejné složce. Jak ukazuje obrázek 3.2, otevření této stránky v prohlížeči, který podporuje element `audio`, zobrazí jednoduchý ovládací panel s posuvníkem pro přehrávání audia. Když uživatel klepne na tlačítko přehrávání, skladba se podle očekávání začne přehrávat.



Obrázek 3.2: Jednoduchý přehrávač audia

Atribut `controls` říká prohlížeči, aby zobrazil běžné ovládací prvky pro zahájení a ukončení přehrávání a posun ve skladbě, a samozřejmě také nastavení hlasitosti. Vynechání atributu `controls` ovládací prvky skryje, a neumožní tak uživateli zahájit přehrávání skladby.

Obsah těla elementu `audio` představuje text, který se v prohlížeči zobrazí, pokud prohlížeč nepodporuje tento multimediální element, tedy právě to, co vy i vaši uživatelé uvidíte, použijete-li zastaralý prohlížeč. Pomocí něho můžete také specifikovat alternativní způsob zobrazení multimediálního obsahu jako například v přehrávači Flashe anebo poskytnutím přímého odkazu na multimediální soubor.

POUŽITÍ ZDROJE

Konečně se dostáváme k tomu nejdůležitějšímu z atributů – k atributu `src`. V nejjednodušším případě odkazuje atribut `src` na soubor obsahující multimediální obsah. Co když ale prohlížeč nepodporuje použitý kontejner anebo kodek (Ogg a Vorbis v tomto případě)? V takovém případě je možné použít alternativní způsob nastavení zdroje umožňující specifikovat více zdrojů, ze kterých si může prohlížeč vybírat (viz ukázkový soubor `audio_viceZdroju.html`):

```
<audio controls>
  <source src="johann_sebastian_bach_air.ogg">
  <source src="johann_sebastian_bach_air.mp3">
  Skladba od Johanna Sebastiana Bacha.
</audio>
```

V tomto případě namísto atributu `src` elementu `audio` přidáváme rovnou dva nové elementy `source`. Ty dávají prohlížeči možnost vybrat si zdroj, který nejlépe vyhovuje možnostem přehrávání, jimiž oplývá. Zdroje se zpracovávají v pořadí, v jakém jsou zadané, takže pokud prohlížeč dokáže přehrát více uvedených zdrojových souborů, použije se ten uvedený v pořadí jako první.

Poznámka

Multimediální soubory, které slibují maximální zážitek pro uživatele anebo představují nejmenší zátěž pro server, by měly být na čelních pozicích každého seznamu elementů `source`.

Otevření této stránky v podporovaném prohlížeči nemusí, v porovnání s předchozím příkladem, přinést žádné vizuální změny. Pokud však prohlížeč nepodporuje formát OggVorbis, ale formát MP3 ano, bude nyní přehrání multimediálního obsahu možné. Kouzlo tohoto modelu spočívá v tom, že při zadávání kódu, který pracuje s multimediálním souborem, nezáleží na tom, jaký kontejner anebo kodek používá. Prohlížeč nabízí jednotné rozhraní pro manipulaci s multimédii, bez ohledu na to, který ze zdrojů se použil.

Existuje ještě další způsob, jak prohlížeči poradit, který zdroj použít. Jak si jistě vzpomínáte, může kontejner podporovat mnoho různých typů kodeků. Prohlížeč tedy může mylně vyhodnotit, které ze zadaných souborů podporuje, na základě jejich přípony. Zadáte-li však atribut `type`, který neodpovídá zdrojovému souboru, může prohlížeč odmítnout soubor přehrát. Je proto moudré atribut `type` používat pouze v případech, kdy jste si typem obsahu naprosto jistí. Jinak je lepší tento atribut vynechat a nechat detekci kódování na prohlížeči. Je vhodné poznamenat, že formát WebMnabízí pouze jeden kodek pro audio a jeden pro video. To znamená, že vám přípona `.webm` anebo typ obsahu `video/webm` řekne vše, co potřebujete o souboru vědět. Pokud prohlížeč podporuje formát WebM, měl by být schopný přehrát jakýkoli soubor `.webm`, tak jak demonstruje následující ukázka kódu (kterou najdete v souboru `audio_type.html`):

```
<audio controls>
  <source src="johann_sebastian_bach_air.ogg"
   type="audio/ogg; codecs=vorbis">
  <source src="johann_sebastian_bach_air.mp3" type="audio/mpeg">
  Skladba od Johanna Sebastiana Bacha.
</audio>
```

Jak je z ukázky patrné, může atribut `type` deklarovat jak kontejner, tak typ kodeku – v tomto případě OggVorbis, resp. MP3. Kompletní seznam najdete v dokumentu RFC 4281, který spravuje IETF (Internet EngineeringTaskForce), některé z běžných kombinací však uvádí tabulka 3.2.

Tabulka 3.2: Typy multimédií a odpovídající hodnoty atributu `type`

Typ	**Hodnota atributu** `type`
Theora video a Vorbis audio v kontejneru Ogg	type='video/ogg; codecs="theora, vorbis"'
Vorbis audio v kontejneru Ogg	type='audio/ogg; codecs=vorbis'
Video H.264 v profilu Baselinea audio AAC v profilu LC (LowComplexity) v kontejneru MP4	type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'
MPEG-4 v profilu Visuala audio AAC v profilu LC v kontejneru MP4	type='video/mp4; codecs="mp4v.20.8, mp4a.40.2"'

OVLÁDACÍ PRVKY

Již jste měli možnosti vidět výchozí ovládací prvky elementů `audio` a `video` zobrazené pomocí atributu`controls`. Jak asi tušíte, vynechání tohoto atributu způsobí nejen nezobrazení ovládacích prvků u přehrávaného multimédia, ale v případě *audio* souborů úplně všeho, vzhledem k tomu, že jedinou vizuální reprezentací elementu `audio` jsou jeho ovládací prvky (element `video` bez ovládacích prvků stále zobrazí svůj multimediální obsah). Vynechání atributu `controls` by mělo skrýt jakýkoli obsah ovlivňující normální vykreslení stránky. Jedním ze způsobů, jak v takovém případě zahájit přehrávání obsahu, je atribut `autoplay` (viz ukázkový soubor `audio_autoplay.html`):

```
<audio autoplay>
  <source src="johann_sebastian_bach_air.ogg"
   type="audio/ogg; codecs=vorbis">
  <source src="johann_sebastian_bach_air.mp3" type="audio/mpeg">
  Skladba od Johanna Sebastiana Bacha.
</audio>
```

Přidání atributu `autoplay` zajistí, aby se soubor začal přehrávat hned po svém načtení, bez jakékoli interakce s uživatelem. Většině uživatelů se toto chování nebude ani trochu líbit, a proto používejte atribut `autoplay` s rozvahou. Automatické zahájení přehrávání audia může mít za cíl navodit tu správnou atmosféru pro uživatele nebo ještě hůře k reklamním účelům. Může však narušovat jiné, právě přehrávané audio na stroji uživatele a také velmi komplikovat život uživatelům, kteří při procházení webovým obsahem spoléhají na čtečku obrazovky.

Pokud nabízené ovládací prvky nevyhovují vzhledu vašeho webu anebo potřebujete řídit multimediální element na základě výpočtů a chování, kterými výchozí ovládací prvky nedisponují, pomůže vám řada připravených metod JavaScriptu. Tabulka 3.3 uvádí některé z těch nejběžnějších.

Tabulka 3.3: Běžné ovládací metody

Metoda	Popis
load	Načte multimediální soubor a připraví ho k přehrávání. Tuto metodu běžně není zapotřebí volat, pokud se samotný element dynamicky nevytváří. Metoda se hodí, je-li zapotřebí načítat data předem, ještě před přehráváním.
play	Načte (je-li to zapotřebí) a přehraje multimediální soubor. Soubor se začne přehrávat od začátku, není-li jeho přehrávání už pozastavené na jiné pozici.
pause	Pozastaví přehrávání, probíhá-li právě.
canPlayType	Ověří, zda element `video` dokáže přehrát hypotetický soubor se zadaným typem MIME, který metoda bere jako parametr.

Metoda `canPlayType` má i jeden ne příliš zjevný způsob využití: zadáním typu MIME videoklipu přehrávaného v dynamicky vytvořeném elementu `video` je možné snadno určit, jestli prohlížeč daný typ podporuje. Následující kód ověří podporu videí s typem MIME `fooType` prohlížečem, aniž by se cokoli zobrazilo v okně prohlížeče:

```
var supportsFooVideo =
!!(document.createElement('video').canPlayType('fooType'));
```

Tabulka 3.4 ukazuje několik vlastností multimediálních elementů určených pouze pro čtení.

Tabulka 3.4: Vlastnosti multimediálních elementů pouze pro čtení

Vlastnost	Popis
duration	Délka trvání multimediálního obsahu v sekundách. Pokud není délka známá, obsahuje atribut hodnotu `NaN`.
paused	Obsahuje hodnotu `true`, pokud je přehrávání klipu právě pozastavené. Výchozí hodnota před zahájením přehrávání je `true`.
ended	Obsahuje hodnotu `true`, pokud se již dokončilo přehrávání klipu.
startTime	Vrací údaj o čase, na kterém může začít přehrávání. Zpravidla to bude hodnota 0.0, nejedná-li se o streamovaný klip a dřívější obsah již není ve vyrovnávací paměti.
error	Chybový kód, dojde-li k chybě.
currentSrc	Vrací název právě načítaného nebo přehrávaného souboru odpovídající elementu `source`, který prohlížeč vybral.

Tabulka 3.5 uvádí některé z vlastností multimediálních elementů, které je možné ve skriptech také upravovat, a tím okamžitě ovlivňovat přehrávání obsahu.

Tabulka 3.5: Vlastnosti multimediálních elementů pro čtení i zápis

Vlastnost	Popis
autoplay	Udává, jestli se má multimediální klip začít automaticky přehrávat po svém vytvoření.
look	Udává, jestli se má klip začít znovu přehrávat od začátku, poté co skončí (hodnota `true`).
currentTime	Udává dobu v sekundách, která uplynula od začátku přehrávání. Nastavením této hodnoty je možné přeskočit na konkrétní pozici v klipu.
controls	Udává, jestli jsou ovládací prvky přehrávače právě viditelné.
volume	Udává hlasitost přehrávaného klipu v podobě relativní hodnoty v rozsahu 0.0 až 1.0.
muted	Udává ztišení právě přehrávaného klipu.
autobuffer	Udává, jestli se má prohlížeč pokoušet načíst multimediální soubor ještě před zahájením jeho přehrávání. U klipů s nastaveným automatickým přehráváním se tato vlastnost ignoruje.

Pomocí nejrůznějších vlastností a metod mohou vývojáři vytvářet libovolná uživatelská rozhraní pro přehrávání libovolných multimédií podporovaných prohlížečem.

Práce s audiem

Po seznámení se společnými vlastnostmi multimediálních elementů `audio` a `video` již v zásadě znáte vše, co může element `audio` nabídnout. Podívejme se na jednoduchý příklad ukazující ovládání klipu v praxi.

PŘEHRÁNÍ AUDIA

Pokud vaše uživatelské rozhraní potřebuje přehrát audioklip, ale nechcete narušit jeho vzhled časovou osou ani žádnými jinými ovládacími prvky, můžete vytvořit neviditelný element `audio`, tedy element bez atributu `controls`, případně s tímto atributem nastaveným na hodnotu `false`, a vytvořit své vlastní ovládací prvky pro přehrávání. Podívejte se na následující ukázkový kód, který najdete také v souboru `audio_prehravac.html`:

```
<!DOCTYPE html>
<html>
  <meta charset="utf-8" />
  <title>Audio přehrávač</title>

  <audio id="clickSound">
    <source src="johann_sebastian_bach_air.ogg">
    <source src="johann_sebastian_bach_air.mp3">
  </audio>

  <button id="toggle" onclick="toggleSound()">Přehrát</button>

  <script type="text/javascript">
    function toggleSound() {
        var music = document.getElementById("clickSound");
        var toggle = document.getElementById("toggle");

        if (music.paused) {
          music.play();
          toggle.innerHTML = "Zastavit";
        }
        else {
          music.pause();
          toggle.innerHTML ="Přehrát";
        }
    }
  </script>

</html>
```

Opět zde přichází ke slovu element `audio` a naše oblíbená Bachova skladba. V tomto příkladě jsme však skryli ovládací prvky a nenechali klip přehrát automaticky. Místo toho jsme pro ovládání přehrávání skladby vytvořili následující tlačítko:

```
<button id="toggle" onclick="toggleSound()">Přehrát</button>
```

Toto jednoduché tlačítko ve výchozím stavu informuje uživatele o tom, že klepnutím na něj zahájí přehrávání skladby. Po každém stisknutí tlačítka se volá funkce `toggleSound`. Tato funkce nejdříve získá přístup k elementům `audio` a `button` stránky:

```
if (music.paused) {
    music.play();
    toggle.innerHTML = "Zastavit";
}
```

Na základě hodnoty vlastnosti `paused` elementu `audio` určíme, jestli uživatel pozastavil přehrávání. Výchozí hodnota této vlastnosti, když ještě nezačalo přehrávání, je `true`. Proto bude tato podmínka platit hned po prvním stisknutí tlačítka. V takovém případě se zavolá metoda `play` klipu a změní se text tlačítka, tak aby bylo zřejmé, že jeho další stisknutí pozastaví přehrávání:

```
else {
    music.pause();
    toggle.innerHTML ="Přehrát";
}
```

V opačném případě, pokud není přehrávání skladby pozastavené (tj. skladba hraje), zavoláme metodu `pause` a text tlačítka změníme tak, aby z něj bylo patrné, že jeho další stisknutí obnoví přehrávání. Vypadá to jednoduše, co říkáte? To je také smysl multimediálních elementů HTML5 – umožnit jednoduché přehrávání a ovládání multimédií tam, kde dříve kraloval bezpočet zásuvných modulů. Jednoduchost nadevše.

Práce s videem

Dost už bylo jednoduchých příkladů, je na čase udělat něco trochu složitějšího. Element `video` HTML5 se velice podobá elementu `audio`, nabízí však několik dodatečných vlastností. Tabulka 3.6 ukazuje některé z těchto vlastností.

Tabulka 3.6: Vlastnosti elementu `video`

Vlastnost	Popis
poster	Adresa URL obrázku, který reprezentuje obsah videa před jeho načtením. Můžete na něj pohlížet jako na plakát filmu. Hodnotu této vlastnosti je možné číst i měnit, a tím upravovat plakát.
width, height	Tyto vlastnosti umožňují čtení i změnu velikosti přehrávače. Zadané hodnoty se mohou lišit od velikosti samotného videa, čímž je možné docílit vycentrování, orámování a dalších efektů.
videoWidth, videoHeight	Tyto vlastnosti udávají skutečnou šířku, resp. výšku, videa. Hodnoty těchto vlastností jsou pouze pro čtení.

Element `video` má ještě jednu klíčovou funkci, kterou element `audio` nenabízí – je možné ho předat mnohým z metod HTML5 rozhraní Canvas (více informací o tomto rozhraní najdete v kapitole 2).

VYTVOŘENÍ PŘEHRÁVAČE VIDEA S ČASOVOU OSOU S NÁHLEDY

V této (trochu komplexnější) ukázce si představíme, jak získat jednotlivé snímky obsahu elementu `video` a zobrazit je v dynamickém plátně. V rámci demonstrace těchto možností vytvoříme jednoduchý přehrávač videa s časovou osou s náhledy. Zatímco se video přehrává, budou se na plátně, nacházejícím se poblíž, pravidelně zobrazovat snímky z něj. Klepnutím na kterýkoli ze snímků se ve videu přesunete na daný okamžik. Jak se sami přesvědčíte, je s použitím jen několika řádků kódu možné vytvořit přehrávač videa oplývající časovou osou s náhledy, jež mohou uživatelé používat k navigaci v rámci delšího videa.

Jako ukázkový videoklip nám poslouží lákavá reklama na občerstvení promítaná v kinech v polovině 20. stolení. Pojďme tedy do bufetu dát si něco dobrého (viz obrázek 3.3).

Obrázek 3.3: Přehrávač s časovou osou s náhledy

PŘIDÁNÍ ELEMENTŮ VIDEA A PLÁTNA

Začneme jednoduchou deklarací zajišťující zobrazení našeho videoklipu:

```
<video id="movies" autoplay oncanplay="startVideo()"
 onended="stopTimeline()" autobuffer="true" width="400px" height="300px">
    <source src="klip.ogv"
     type='video/ogg; codecs="theora, vorbis"'>
    <source src="klip.mp4"
     type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
</video>
```

Většina kódu vám bude povědomá z příkladu audiopřehrávače, takže se zaměříme pouze na odlišnosti. Element `audio` přirozeně nahradil element `video` a elementy `source` odkazují na videa ve formátech Ogg a MPEG, mezi kterými může prohlížeč vybírat.

Element `video` v tomto případě obsahuje atribut `autoplay`. Video se tedy začne přehrávat, hned jak se stránka načte.Všimněte si registrace obsluhy dvou událostí. Poté, co se video načte a je připravené k přehrávání, zavolá se naše obsluha události `canplay`. Podobně se poté, co video skončí, zavolá naše obsluha události `ended`, která zastaví vytváření snímků videa.

Následně přidáme plátno s identifikátorem `timeline`, na kterém budeme v pravidelných intervalech zobrazovat snímky z přehrávaného videa.

```
<canvas id="timeline" width="400px" height="300px">
```

PŘIDÁNÍ PROMĚNNÝCH

V následující části příkladu začneme s tvorbou jeho programového kódu definicí několika proměnných, které usnadní další úpravy ukázky a zlepší čitelnost kódu.

```
// doba v milisekundách mezi aktualizacemi snímků
var updateInterval = 5000;

// velikost snímků
var frameWidth = 100;
var frameHeight = 75;

// počet snímků
var frameRows = 4;
var frameColumns = 4;
var frameGrid = frameRows * frameColumns;
```

Proměnná `updateInterval` určuje, jak často se budou vytvářet náhledy snímků videa – v tomto případě každých pět sekund. Proměnné `frameWidth` a `frameHeight` určují, jak velké budou náhledy snímků zobrazené na plátně. Proměnné `frameRows`, `frameColumns` a `frameGrid` určují, kolik náhledů se zobrazí v časové ose:

```
// aktuální snímek
var frameCount = 0;

// ke zrušení časovače po skončení přehrávání
var intervalId;

var videoStarted = false;
```

Abychom věděli, který snímek videa zobrazujeme, je zde proměnná `frameCount` (náhled snímku se v tomto příkladě vytváří každých pět vteřin). Proměnná `intervalId` slouží pro zastavení časovače používaného při vytváření náhledů snímků. Na závěr vytvoříme ještě proměnnou `videoStarted`, která slouží jako příznak zajišťující vytvoření pouze jednoho časovače.

FUNKCE PRO AKTUALIZACI SNÍMKU

Klíčovou funkcí našeho příkladu a místem, kde se video potkává s plátnem, je funkce `updateFrame`, ve které získáme snímek videa a nakreslíme ho na plátno.

```
// zobrazí snímek na plátně
fiction updateFrame() {
    var video = document.getElementById("movies");
    var timeline = document.getElementById("timeline");

    var ctx = timeline.getContext("2d");

    // výpočet aktuální pozice na základě počtu snímků
    // následně se do plátna přidá obrázek - jako jeho zdroj slouží video
    var framePosition = frameCount % frameGrid;
    var frameX = (framePosition % frameColumns) * frameWidth;
    var frameY = (Math.floor(framePosition / frameRows)) * frameHeight;
    ctx.drawImage(video, 0, 0, 400, 300, frameX, frameY, frameWidth,
      frameHeight);

    frameCount++;
}
```

Jak už víte z kapitoly 2, je v prvé řadě třeba získat dvourozměrný kontext plátna.

```
var ctx = timeline.getContext("2d");
```

Abychom mohli vyplnit plátno náhledy snímků ve směru zleva doprava a odshora dolů, musíme určit, na jaké pozici v plátně se má nový náhled umístit. Učiníme tak na základě pořadí právě vytvářeného snímku. Podle výšky a šířky snímku můžeme následně určit přesné souřadnice, na kterých obrázek v plátně zobrazit.

```
var framePosition = frameCount % frameGrid;
var frameX = (framePosition % frameColumns) * frameWidth;
var frameY = (Math.floor(framePosition / frameRows)) * frameHeight;
```

Konečně se dostáváme ke klíčovému volání, které zajistí vlastní zobrazení obrázku na plátně. Parametry metody `drawImage` jako je pozice nebo velikost, již známe z předchozích příkladů práce s plátnem, tentokrát však namísto obrázku předáváme video:

```
ctx.drawImage(video, 0, 0, 400, 300, frameX, frameY, frameWidth,
frameHeight);
```

Metody plátna akceptují jako parametr mimo obrázků také videa, což umožňuje snadnou práci s videem a jeho zobrazení v jiném umístění.

Poznámka

Pokud se jako zdroj obrazu používá v plátně video, nakreslí se v něm pouze právě zobrazený snímek videa. Obsah plátna se nebude dynamicky aktualizovat společně s přehrávaným videem. Pokud si přejete obsah plátna aktualizovat, musíte sami zajistit překreslování obrázků při přehrávání videa.

FUNKCE PRO ZAHÁJENÍ PŘEHRÁVÁNÍ VIDEA

V závěru ještě aktualizujeme hodnotu proměnné `frameCount`, tak aby reflektovala skutečnost, že jsme přidali nový snímek do naší časové osy. Zbývá vytvořit funkci, která bude pravidelně aktualizovat snímky v časové ose:

```
function startVideo() {

    // časovač se nastaví pouze při prvním spuštění videa
    if (videoStarted)
      return;

      videoStarted = true;

      // zobrazení úvodního snímku a nastavení časovače pro pravidelné
      // zobrazování dalších snímků
      updateFrame();
      intervalId = setInterval(updateFrame, updateInterval);
```

Připomeňme, že se funkce `startVideo` volá, jakmile je video dostatečně načtené k tomu, aby se mohlo začít s jeho přehráváním. V prvé řadě se ujistíme, že se tato událost zpracuje v rámci daného načtení stránky pouze jednou – pro případ, že by došlo k restartování videa:

```
// časovač se nastaví pouze při prvním spuštění videa
if (videoStarted)
  return;

  videoStarted = true;
```

Po spuštění videa zachytíme jeho první snímek. Poté nastavíme časovač, který bude v pravidelných intervalech volat naši aktualizační funkci `updateFrame`. V konečném důsledku to znamená zachycení nového snímku každých pět sekund:

```
// zobrazení úvodního snímku a nastavení časovače pro pravidelné
// zobrazování dalších snímků
updateFrame();
intervalId = setInterval(updateFrame, updateInterval);
```

ZPRACOVÁNÍ VSTUPU UŽIVATELE

Zbývá zajistit obsluhu události klepnutí na některý ze snímků v časové ose:

```
// nastavení obsluhy klepnutí na snímek
var timeline = document.getElementById("timeline");
timeline.onclick = function(evt) {
    var offX = evt.layerX - timeline.offsetLeft;
    var offY = evt.layerY - timeline.offsetTop;

    // určení pořadí snímku, na který uživatel klepnul (počítá se od nuly)
    var clickedFrame = Math.floor(offY / frameHeight) * frameRows;
    clickedFrame += Math.floor(offX / frameWidth);

    // určení snímku hledaného ve videu
    var seekedFrame = (((Math.floor(frameCount / frameGrid)) *
                        frameGrid) + clickedFrame);

    // pokud uživatel klepnul na snímek, který se v časové ose nachází
    // za aktuálním snímkem, jedná se o snímek z předchozího cyklu snímků
    if (clickedFrame > (frameCount % 16))
        seekedFrame -= frameGrid;

        // není možné posunout se před začátek videa
        if (seekedFrame < 0)
            return;
```

Zde už je to trochu složitější. Získáme element plátna a nastavíme obsluhu události klepnutí myší. Obsluha použije informace o události k určení souřadnic pozice, na kterou uživatel klepnul myší:

```
var timeline = document.getElementById("timeline");
timeline.onclick = function(evt) {
    var offX = evt.layerX - timeline.offsetLeft;
    var offY = evt.layerY - timeline.offsetTop;
```

Na základě rozměrů snímku následně určíme, na který ze 16 snímků uživatel klepnul.

```
// určení pořadí snímku, na který uživatel klepnul (počítá se od nuly)
var clickedFrame = Math.floor(offY / frameHeight) * frameRows;
clickedFrame += Math.floor(offX / frameWidth);
```

Snímkem, na který uživatel klepnul, by měl být některý z nedávných snímků videa, a proto určíme nejaktuálnější snímek se zadaným pořadím:

```
// určení snímku hledaného ve videu
var seekedFrame = (((Math.floor(frameCount / frameGrid)) *
                    frameGrid) + clickedFrame);
```

Pokud uživatel klepne na snímek nacházející se v časové ose za aktuálním snímkem, vrátíme se zpět o jeden celý cyklus snímků:

```
// pokud uživatel klepnul na snímek, který se v časové ose nachází
// za aktuálním snímkem, jedná se o snímek z předchozího cyklu snímků
if (clickedFrame> (frameCount % 16))
    seekedFrame -= frameGrid;
```

Na závěr ještě bezpečnostní opatření zabraňující jakýmkoli pokusům uživatele o přechod na snímek nacházející se před začátkem videa:

```
// není možné posunout se před začátek videa
if (seekedFrame< 0)
    return;
```

Nyní když víme, jaký bod v čase uživatel hledá, můžeme tuto informaci použít k posunu právě přehrávaného videa. Přestože se jedná o zcela klíčovou část této ukázky, je samotný kód poměrně jednoduchý:

```
// přesunutí na daný snímek videa (v sekundách)
var video = document.getElementById("movies");
video.currentTime = seekedFrame * updateInterval / 1000;

// nastavení aktuálního počtu snímků na cílový snímek
frameCount = seekedFrame;
```

Nastavením hodnoty vlastnosti `currentTime` elementu `video` se ve videu přesuneme na zadaný čas. Právě vybraný snímek následně nastavíme jako hodnotu proměnné udávající aktuální počet snímků.

Poznámka

Na rozdíl od mnohých funkcí JavaScriptu, které pracují s milisekundami, se hodnota vlastnosti `currentTime` zadává v sekundách.

FUNKCE PRO ZASTAVENÍ VIDEA

Vše, co zbývá udělat, je zastavit vytváření náhledů snímků po skončení videa. Přestože to není nezbytně nutné, bez této úpravy by naše ukázková aplikace pokračovala v zobrazování snímků videa, které již skončilo, čímž by po chvíli vymazala celou časovou osu.

```
// ukončení zobrazování snímků
functionstopTimeline() {
    clearInterval(intervalId);
}
```

Funkce `stopTimeline` se zavolá při výskytu události `ended`, ke které dochází při ukončení přehrávání videa.

Náš přehrávač s časovou osou s náhledy pravděpodobně neuspokojí náročné uživatele, obsahuje však pouze minimální množství kódu. Pokračujeme s praktickými příklady.

Praktické doplňky

Některé techniky jednoduše nezapadají do tradičních ukázkových příkladů, přesto najdou využití v mnoha aplikacích HTML5. V této části kapitoly najdete některé z těchto jednoduchých, přesto praktických a běžných doplňků.

HUDBA NA POZADÍ

Mnoho webových stránek se snaží zabavit své návštěvníky přehráváním nějaké hudby na pozadí. Přestože nic takového neschvalujeme, s použitím HTML5 a elementu `audio` je snadné toho docílit:

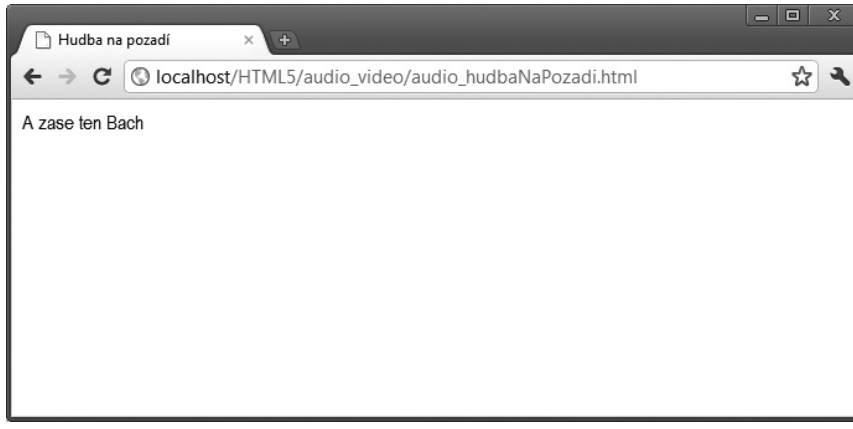
```
<!DOCTYPE html>
<html>
  <meta charset="utf-8" />
  <link rel="stylesheet" href="styly.css">
  <title>Hudba na pozadí</title>

  <audio autoplay loop>
      <source src="johann_sebastian_bach_air.ogg">
      <source src="johann_sebastian_bach_air.mp3">
  </audio

  <h1>A zase ten Bach</h1>

</html>
```

Jak sami vidíte, je přehrávání opakujícího se zvukového klipu na pozadí záležitostí pouhé deklarace jednoho elementu `audio` s atributy `autoplay` a `loop` (viz obrázek 3.4).



Obrázek 3.4: Automatické přehrání hudby s použitím atributu `autoplay`

Jak ztratit návštěvníky mrknutím oka

Brian říká: „S velkou mocí přichází i velká odpovědnost, a jen proto že něco *můžete,* ještě neznamená, že byste *měli*. Jen si vzpomeňte na element `blink`.

Nenechte se zlákat možnostmi přehrávání audia a videa a nepoužívejte je tam, kde to není vhodné. Pokud už máte dobrý důvod k použití atributu `autoplay` (například v prohlížeči multimedií, kde uživatel automatické zahájení přehrávání očekává), nezapomeňte dát uživateli možnost tuto funkci deaktivovat. Nic neodradí uživatele od webu rychleji než otravný obsah, který nejde nijak snadno vypnout."

PŘEHRÁVÁNÍ PŘI PŘEJETÍ MYŠÍ

Dalším jednoduchým příkladem práce s videoklipy je volání metod `play` a `pause` na základě pohybu myší nad videem. To se může hodit u webů, na kterých se zobrazuje mnoho videoklipů a uživatel si může vybrat, který se má přehrát. Videogalerie může přehrát krátkou ukázku videa, když nad ním uživatel přejede myší, a celé video, když na něj uživatel klepne. Tohoto efektu je možné snadno docílit pomocí podobného kódu, jako je ten následující (viz ukázkový soubor `video_mys.html`):

```
<!DOCTYPE html>
<html>
  <meta charset="utf-8" />
  <link rel="stylesheet" href="styly.css">
  <title>Přehrání videa po přejetí myší</title>

  <video id="movies" onmouseover="this.play()" onmouseout="this.pause()"
```

```
    autobuffer="true" width="400px" height="300px">
    <source src="klip.ogv"
      type='video/ogg; codecs="theora, vorbis"'>
    <source src="klip.mp4"
      type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  </video>

  <h1>Video přehrajete, pokud nad ním přejedete myší</h1>
</html>
```

Pouhým nastavením několika atributů je možné docílit toho, aby přehrávání videa vyvolalo přejetí kurzoru myši nad ním, jak ukazuje obrázek 3.5.

Obrázek 3.5: Video se přehraje po přejetí kurzoru myší nad ním

Shrnutí

V této kapitole jsme se seznámili se dvěma velmi důležitými elementy HTML5, `audio` a `video`, a s jejich možnostmi. Ukázali jsme si, jak mohou posloužit při tvorbě podmanivých webových aplikací. Elementy `audio` a `video` rozšiřují multimediální možnosti aplikací HTML5, které nyní mohou přehrávat audio- i videoobsah bez použití zásuvných modulů, a přitom nabídnou běžné a jednotné rozhraní.

Nejdříve jsme probrali audio- a videokontejnery a kodeky a seznámili se s aktuální situací v oblasti podpory jednotlivých kodeků. Seznámili jsme se s postupem umožňujícím prohlížeči přepnout se na typu obsahu, který je pro něj nejvhodnější.

Poté jsme si ukázali, jak s použitím aplikačního rozhraní ovládat přehrávání audia a videa, a na závěr jsme se podívali na možné využití multimedií HTML5 ve vašich aplikacích.

V následující kapitole si ukážeme, jak lze jen s minimem kódu zohledňovat ve vašich aplikacích geografickou lokalitu vašich návštěvníků.