

KAPITOLA 6

Vylepšení formulářů

Za posledních deset let se HTML příliš nezměnilo. Od časů, kdy kolem roku 1997 prohlížeče implementovaly změny v HTML 4, zůstala kolekce značek a atributů, které dnes používáme pro tvorbu webových stránek, v podstatě stejná. Ovšem díky postupnému zdokonalování podpory CSS v prohlížečích jsme i s touto limitovanou výbavou značek schopni vytvářet stále bohatší a složitější designy. Podobně jako módní návrháři, kteří se každou sezónu vytasí s převratnými novými módními střihy, barvami a materiály, i styly jsou pořád čerstvé a nové, i když výchozí podkladové modely jsou si strašně podobné.

Je tu ovšem jeden aspekt webového designu, u kterého ani CSS nedokážou zakrýt stagnaci jazyka HTML – webové formuláře. Bez ohledu na to, jak je plánujete vylepšit, musíte se v posledních deseti letech u webových formulářů spokojit s docela omezenou sadou ovládacích prvků HTML:

- Textová vstupní pole.
- Zaškrťovací políčka.
- Přepínače.
- Rozevírací nabídky a seznamy (také se jim říká roletová menu).
- Víceřádkové textové oblasti.
- Tlačítka.

Protože je výběr takto omezen, formuláře byly jednou z prvních částí HTML, která přilákala pozornost vývojářů experimentujících s vylepšováním stránek pomocí JavaScriptu.

V této kapitole vybudujeme několik vlastních, opětovně využitelných a vylepšených formulářů. Dostanete tak příležitost aplikovat zde mnohé z dovedností, které jste si doposud v této knize osvojili a dokonce se naučíte i několik triků navíc.

Rozšíření DOM HTML

Ovládací prvky formuláře nejsou prvky HTML, s nimiž byste pracovali každý den, nehledě na skutečnost, že mají v sobě zabudováno bohatší chování a interaktivitu, než jaká se dá popsat s událostmi `click`, `mouseover`, `mouseout`, `focus` a `blur`, nebo s vlastnostmi jako jsou `id` a `classname`.

Důsledkem je, že kromě standardních vlastností, metod a událostí DOM, s nimiž jsme si už v této knize pohrávali, podporují HTML prvky vztahující se k formuláři ještě celé hejno dalších vlastností a metod, které jsou definovány v samostatné sekci¹ standardu DOM. Přehled nejužitečnějších vlastností a metod máte k dispozici v tabulkách 6.1 a 6.2. Ovládací prvky formuláře podporují i několik dalších událostí – jejich přehled je uveden v tabulce 6.3.

Tabulka 6.1. Dodatečné metody DOM pro formulářové ovládací prvky HTML.

Metoda	Prvek	Popis
<code>blur</code>	<code>input</code> <code>select</code> <code>textarea</code>	Odebere focus z klávesnice tomuto formulářovému ovládacímu prvku.
<code>click</code>	<code>input</code>	Simuluje kliknutí myši na tomto ovládacím prvku.
<code>focus</code>	<code>input</code> <code>select</code> <code>textarea</code>	Udělí focus z klávesnice tomuto formulářovému ovládacímu prvku.
<code>reset</code>	<code>form</code>	Obnoví všechny formulářové ovládací prvky na jejich výchozí hodnoty.
<code>select</code>	<code>input</code> <code>textarea</code>	Vybere textový obsah tohoto ovládacího prvku.
<code>submit</code>	<code>form</code>	Odešle formulář, aniž by spustila událost <code>submit</code> .

V následujících dvou praktických příkladech si pohrajeme s některými – ale nikoliv se všemi – funkcemi DOM, které jsou specifické pro formuláře. Jakmile si uděláte představu o tom, jakou funkcionalitu navíc dokáže JavaScript poskytnout, vraťte se k těmto tabulkám a popřemýšlejte o dalších způsobech, jimiž by mohly tyto nástroje vylepšit vaše webové formuláře.

Tabulka 6.2. Dodatečné vlastnosti DOM pro formulářové ovládací prvky HTML.

Vlastnost	Prvek	Popis
<code>elements</code>	<code>form</code>	Seznam uzlů, který obsahuje všechny formulářové ovládací prvky ve formuláři.

¹ <http://www.w3.org/TR/DOM-Level-2-HTML/>

Vlastnost	Prvek	Popis
checked	input	Když je daný ovládací prvek vybrán, má u ovládacích prvků <code>input</code> s <code>type</code> <code>checkbox</code> a <code>radio</code> tato vlastnost hodnotu <code>true</code> .
disabled	button input optgroup option select textarea	Když má vlastnost hodnotu <code>true</code> , je ovládací prvek pro uživatele nepřístupný.
form	button input option select textarea	Odkaz na prvek <code>form</code> , který obsahuje tento ovládací prvek.
index	option	Index tohoto prvku <code>option</code> uvnitř prvku <code>select</code> , který ho obsahuje (začíná se od 0).
options	select	Seznam uzlů obsahující všechny prvky <code>option</code> v této nabídce.
selected	option	<code>true</code> , je-li tato volba aktuálně vybraná, jinak <code>false</code> .
selectedIndex	select	Index aktuálně vybraného prvku <code>option</code> v seznamu (začíná se od 0).
value	button input option select textarea	Aktuální hodnota tohoto ovládacího prvku, jako kdyby se odeslala na server.

Tabulka 6.3. Dodatečné vlastnosti DOM pro formulářové ovládací prvky HTML.

Událost	Prvek	Spustí se, když...
change	input select textarea	... ovládací prvek ztratí focus poté, co se jeho hodnota změnila.
select	input textarea	... uživatel vybral v poli nějaký text.
submit	form	... uživatel požádal o odeslání formuláře.

Příklad – závislá pole

Nenuťte uživatele, aby musel vyplňovat všechna pole formuláře. Konkrétní hodnota, kterou uživatel zadá do jednoho pole formuláře, může způsobit, že jiné pole formuláře už bude irrelevantní. Proč se spoléhat na uživatele stránky, aby sám rozhodoval, která pole jsou pro něj relevantní a jaká ne, když je možné pomoci vlastnosti `disabled`, zmíněné v tabulce 6.2, znepřístupnit konkrétní pole, která tak může uživatel bezpečně ignorovat?

Zaškrťovací políčko na obrázku 6.1, které je umístěno pod druhým přepínačem, je uživateli přístupné (relevantní) pouze tehdy, když je vybrán tento druhý přepínač. Ve všech ostatních případech je toto zaškrťovací políčko pomocí JavaScriptu vypnuto (znepřístupněno).

Která Enterprise?

Která z lodí Enterprise je vaše nejoblíbenější?

- ☐ Enterprise NX-01
- ☒ USS Enterprise NCC-1701 (třída Constitution)
 - ☒ Po opravě
- ☐ USS Enterprise NCC-1701-A (třída Enterprise)
- ☐ USS Enterprise NCC-1701-B (třída Excelsior)
- ☐ USS Enterprise NCC-1701-C (třída Ambassador)
- ☐ USS Enterprise NCC-1701-D (třída Galaxy)
- ☐ USS Enterprise NCC-1701-E (třída Sovereign)
- ☐ USS Enterprise NCC-1701-J (neznámá třída)

Odeslat

Obrázek 6.1. Zaškrťovací políčko bude přístupné jen tehdy, když uživatel vybere druhý přepínač.

Nejllepší způsob, jakým se dají implementovat taková závislá pole, závisí (jak obratně řečeno) na konkrétní logice formuláře. Pro potřeby tohoto příkladu ovšem použijeme přístup natolik všeobecný, abyste ho mohli využít i v mnoha jiných situacích. Pro tento účel si nejprve stanovíme a ujasníme následující výchozí předpoklady:

- Každý formulář bude mít pouze jednu úroveň závislosti polí (závislé pole nemůže být závislé na nějakém jiném závislém poli).
- Závislými poli mohou být pouze prvky `input` s typem `text`, `password`, `checkbox` nebo `radio`.
- Závislá pole mohou záviset pouze na prvcích `input` těchto typů.
- Závislá pole následují v dokumentu okamžitě za poli, na nichž jsou závislá.
- Závislá pole budou obsažena v prvcích `label`.

Nyní se podívejte, jak bude vypadat kód nejenom pro formulářové pole typu přepínač, ale také pro formulářové pole typu zaškrťovací políčko, které je závislé na tomto přepínači:

dependentfields.html (výňatek).

```
<div>
```

```

<label for="ncc1701">
  <input type="radio" name="whichenterprise"
        value="ncc1701" id="ncc1701" />
  USS Enterprise NCC-1701 (třída Constitution)
</label>
<label for="ncc1701refit" class="secondary">
  <input type="checkbox" class="dependent"
        name="ncc1701refit" id="ncc1701refit"
        value="ncc1701refit" disabled="true" />
  Po opravě
</label>
</div>

```

Připomínám, že zaškrťovací políčko (checkbox) má třídu (class) `dependent` – takto specifikujeme, že zaškrťovací políčko je závislé na poli, které je v kódu před ním. Kdyby měl náš přepínač více závislých polí, každé z nich by mělo třídu `dependent`. Když se toto značkování předá prohlížeči, v němž je JavaScript vypnutý, budou závislá pole formuláře pořád přístupná pro uživatele.

Náš skript začleníme do objektu, který nazveme `DependentFields`:

dependentfields.js (výňatek)

```

var DependentFields =
{
  init: function()
  {
    ...
  },
  ...
};
Core.start(DependentFields);

```

Práci na našem skriptu zahájíme tím, že napíšeme hlavní funkce, které budou podle potřeby zpřístupňovat a znepřístupňovat příslušné pole formuláře:

dependentfields.js (výňatek)

```

disable: function(field)
{
  field.disabled = true;
  Core.addClass(field, "disabled");
  Core.addClass(field.parentNode, "disabled");
},
enable: function(field)
{
  field.disabled = false;

```

```
Core.removeClass(field, "disabled");
Core.removeClass(field.parentNode, "disabled");
},
```

Dost jednoduché, že? Obě funkce začínají tím, že nastaví vlastnost `disabled` pole. První funkce se aplikuje na formulářové pole s třídou `disabled` a na prvek, který jej obsahuje. Druhá funkce pak odebírá tuto třídu z tohoto pole a obsahujícího prvku. Třídu `disabled` dále použijeme pro ostylování tohoto nepřístupného pole formuláře:

dependentfields.css (výňatek)

```
label.disabled {
color: #A0A0A0;
}
```

Pokud dáváte přednost technice pozicování za levou hranu obrazovky, kterou jsem detailně popsal ve čtvrté kapitole, klidně ji použijte – pak nepřístupná pole formuláře schováte úplně.

Pro vytvoření kódu, který bude volat metody `disable` a `enable`, které jsme vám právě ukázali, je možné použít mnoho přístupů. Jako zřejmý se nabízí přístup, ve kterém bychom přidali posluchače události ke každému prvku `input`, jenž má jedno nebo více závislých polí, a nechali tyto posluchače, aby sami zpřístupňovali a znepřístupňovali závislá pole.

I když tento přístup vypadá jako docela evidentní a přirozený, je problematický. Například přepínač nikdy negeneruje užitečnou událost, když přestane být vybrán. Někdy je jedinou cestou, jak dát JavaScriptu najevo, že přepínač už není vybrán, událost `click` vedlejšího přepínače.

Další přístup je založen na tom, že ke každému formuláři na stránce se přidají dva posluchači události, kteří budou dávat pozor na události `click` a `change` probublávající z kteréhokoliv pole ve formuláři, přičemž po každé takové události budou aktualizovat stav všech závislých polí ve formuláři:

dependentfields.js (výňatek)

```
init: function()
{
    var forms = document.getElementsByTagName("form");
    for (var i = 0; i < forms.length; i++)
    {
        Core.addEventListener(
            forms[i], "change", DependentFields.changeListener);
        Core.addEventListener(
            forms[i], "click", DependentFields.clickListener);
    }
}
```

Abychom těmto posluchačům události trochu usnadnili práci (protože poběží docela často), bude metoda `init` procházet každý webový formulář na stránce, aby mohla vybudovat seznam závislých polí ve formulářích. Pro každé takové pole si uloží pak jednu referenci na pole, na němž je dané pole závislé:

dependentfields.js (výňatek)

```
var fields = forms[i].getElementsByTagName("input"); ❶
var lastIndependentField = null;
forms[i]._dependents = []; ❷
for (var j = 0; j < fields.length; j++)
{
    if (!Core.hasClass(fields[j], "dependent")) ❸
    {
        lastIndependentField = fields[j]; ❹
    }
    else
    {
        if (lastIndependentField) ❺
        {
            forms[i]._dependents[forms[i]._dependents.length] =
                fields[j]; ❻
            fields[j]._master = lastIndependentField; ❼
        }
    }
}
```

Tenhle kód vypadá na první pohled jako dost spleťitý, takže mi dovoľte, abych ho rozebral:

1. Získáme seznam všech prvků `input` ve formuláři.
2. Pro formulář vytvoříme vlastní vlastnost s názvem `_dependents`, do které uložíme seznam všech závislých polí ve formuláři. Na začátku ji pochopitelně inicializujeme na prázdné pole.
3. Pro každé pole ve formuláři zkontrolujeme, zdali je závislé či nikoliv.
4. Pokud se jedná o nezávislé pole, uložíme na něj referenci do proměnné nazvané jako `lastIndependentField`.
5. Pokud se jedná o závislé pole, ještě jednou zkontrolujeme, zdali jsme opravdu získali referenci na pole, na kterém je toto pole závislé.
6. Do vlastnosti `_dependents` uložíme referenci na toto závislé pole.
7. A nakonec do vlastní vlastnosti s názvem `_master` uložíme referenci na pole, na kterém závisí příslušné závislé pole.

Když tohle všechno shrneme, kód dodá nejenom seznam všech závislých polí v každém formuláři (`form._dependents`), ale také referenci z každého závislého pole na to konkrétní pole, na kterém je dané pole závislé (`dependents._master`).

Metoda `init` pak pro každý formulář nastaví počáteční stavy všech závislých polí. Protože se jedná o docela složitý proces, napíšeme pro tuto činnost samostatnou metodu `updateDependents`:

dependentfields.js (výňatek)

```
    DependentFields.updateDependents(forms[i]);  
  }  
},
```

Protože oba naši posluchači události zahajují stejný proces aktualizace stavů závislého pole, budou oba volat stejnou metodu:

dependentfields.js (výňatek)

```
changeListener: function(event)  
{  
    DependentFields.updateDependents(this);  
},  
clickListener: function(event)  
{  
    DependentFields.updateDependents(this);  
}
```

Nyní už nezbývá nic jiného, než konečně napsat tu veledůležitou metodu `updateDependents`, která buď zpřístupní, nebo znepřístupní každé závislé pole ve formuláři na základě toho, jaký stav bude mít pole, na němž je dané pole závislé.

dependentfields.js (výňatek)

```
updateDependents: function(form)  
{  
    var dependents = form._dependents; ❶  
    if (!dependents)  
    {  
        return;  
    }  
    for (var i = 0; i < dependents.length; i++) ❷  
    {  
        var disabled = true; ❸  
        var master = dependents[i]._master; ❹  
        if (master.type == "text" || master.type == "password") ❺  
        {  
            if (master.value.length > 0)  
            {  
                disabled = false;  
            }  
        }  
        else if (master.type == "checkbox" ||  
            master.type == "radio") ❻
```



```
{
  if (master.checked)
  {
    disabled = false;
  }
}
if (disabled) 7
{
  DependentFields.disable(dependents[i]);
}
else
{
  DependentFields.enable(dependents[i]);
}
},
```

Tuto metodu si opět rozebereme krok za krokem:

1. Začneme tím, že si vytáhneme seznam závislých polí formuláře, který nám připravila metoda `init`.
2. Tento seznam v cyklu projedeme, pěkně jedno závislé pole za druhým.
3. U každého pole na začátku předpokládáme, že bude nepřístupné. Následně si pak vyjasníme, zdali náš předpoklad byl chybný, nebo ne.
4. Abychom to zjistili, podíváme se na pole, na němž je dané pole závislé – potřebnou informaci obdržíme z vlastnosti `_master`, kterou jsme vytvořili v metodě `init`.
5. Je-li hlavním polem `text` nebo `password`, zkontrolujeme délku (`length`) hodnoty vlastnosti `value`. Pokud je větší než nula, závislé pole by nemělo být vypnuté (nepřístupné).
6. Pokud má hlavní pole typ `checkbox` nebo `radio`, zkontrolujeme, zdali je políčko zaškrtnuté, resp. zdali je přepínač vybraný (vlastnost `checked`). Pokud tomu tak bude, závislé pole by nemělo být vypnuté (nepřístupné).
7. Nyní už bezpečně víme, zdali má být závislé pole vypnuté (nepřístupné) či nikoliv, takže zavoláme buď metodu `disable`, nebo metodu `enable`, abychom nastavili patřičný stav.

A máme to z krku! Náš skript využil celou řadu rozšíření HTML z DOM:

- Připravili jsme posluchače události `change`, kterou generují některé ovládací prvky.
- Využili jsme vlastnost `disabled`, kterou podporují všechny formulářové ovládací prvky, abychom v případě potřeby znepřístupnili závislá pole.