POGLAVLJE 1

Osnove biblioteke jQuery

Cody Lindley

1.0 Uvod

Pošto ste se latili knjige o biblioteci jQuery, autori ove knjige će najvećim delom pretpostavljati da imate barem neku predstavu o tome šta je jQuery i šta tačno radi. Iskreno govoreći, kuvari se, u načelu, obično pišu za čitaoce koji žele da nadograde osnovno poznavanje materije. Obrazac recepata: problem – rešenje – objašnjenje, koristi se da biste brzo došli do rešenja za uobičajene probleme. Međutim, ako je jQuery nepoznanica za vas, ne odbacujte i ne poklanjajte još uvek knjigu. Ovo poglavlje smo posvetili vama.

Ukoliko vam je potreban prikaz knjige ili ako ste tek malo (ili uopšte niste) radili sa bibliotekom jQuery, ovo, prvo poglavlje (u svim ostalim pretpostavljamo da ste savladali osnove) uputiće vas u glavne karakteristike jQueryja. Ukoliko ne znate ništa o JavaScriptu i DOM-u, možda ne bi bilo loše da zastanete i zapitate se ima li smisla da proučavate biblioteku jQuery *bez* osnovnog razumevanja jezgra jezika JavaScript i njegovog odnosa s modelom DOM. Preporučujem da proučite DOM i jezgro JavaScripta pre nego što počnete da učite jQuery. Toplo preporučujem priručnik *JavaScript: sveobuhvatni vodič* autora Davida Flanagana (izdavač Mikro knjiga) pre čitanja ove knjige. Ali ne dajte da vas moje skromno mišljenje spreči ako pokušavate da savladate jQuery bez poznavanja DOM-a i JavaScripta. Mnogi su stekli praktično znanje o ovom tehnologijama preko biblioteke jQuery. Znači, iako takav put nije idealan, i dalje se može preći.

Pošto smo to raščistili, pogledajmo formalnu definiciju biblioteke jQuery i kratak opis njene funkcionalnosti:

> jQuery je JavaScript biblioteka otvorenog koda koja pojednostavljuje interakcije između HTML dokumenta ili, preciznije, objektnog modela dokumenta (DOM) i JavaScripta.

Prosto rečeno (i za sve JavaScript hakere starog kova), jQuery opasno pojednostavljuje dinamički HTML (DHTML). Konkretno, jQuery pojednostavljuje manipulisanje HTML dokumentom i kretanje po njemu, obradu događaja čitača veba (engl. *web browser*), DOM animacije, Ajax interakcije i razvoj skriptova na JavaScriptu za različite čitače.

Nakon formalnog opisa biblioteke jQuery, razmotrimo razloge da odaberete jQuery.

Zašto jQuery?

Možda će vam se učiniti nepotrebnim da u ovom kuvaru govorimo o prednostima biblioteke jQuery, posebno pošto već čitate ovaj kuvar i verovatno ste već svesni tih prednosti.

Dakle, iako možda pričam ono što već znate, hajde da ukratko vidimo zašto bi programer odabrao jQuery. Svrha ovoga je da obogatim vaše osnovno znanje o biblioteci jQuery objašnjavajući „zašto“ pre nego što razmotrimo „kako“.

Hvaleći jQuery, neću joj dizati vrednost poredeći je s konkurencijom – zato što naprosto ne verujem da postoji direktna konkurencija. Takođe, verujem da je jQuery danas jedina dostupna biblioteka koja odgovara i dizajnerskim i programerskim zahtevima. U tom smislu, jQuery je klasa za sebe.

Iskreno verujem da svaka od čuvenih JavaScript biblioteka i okruženja na ovome svetu ima svoje mesto i vrednost. Opsežna poređenja su besmislena, ali često im se pribegava – priznajem, i ja sam to radio. Ipak, nakon dugog razmišljanja o tome, moram reći da duboko verujem da su sve JavaScript biblioteke dobre u nečemu. Sve imaju vrednost. To koliko je neka vrednija od druge zavisi više od korisnika i toga kako se upotrebljava, nego od toga šta ona tačno radi. Pored toga, zaključio sam da su male razlike između JavaScript biblioteka često trivijalne u kontekstu širih ciljeva razvoja JavaScripta. Zato, bez daljeg filozofiranja, prilažem listu atributa koji čvrsto govore u prilog korišćenju biblioteke jQuery:

- Otvorenog je koda, i reč je o projektu sa licencama MIT i GNU General Public License (GPL). Slobodna, i to na više načina!
- Mala je (18 KB u minimizovanom obliku) i gzip komprimovana (nekomprimovovana je 114 KB).
- Neverovatno je popularna, što znači da je korisnička zajednica ogromna i da ima mnogo onih koji pružaju pomoć kao programeri i/ili instruktori.
- Usklađuje razlike između veb čitača umesto vas.
- Namerno je napravljena sa svedenom osnovom, s jednostavnom, ali ipak pametno osmišljenom arhitekturom proširivanja dodatnim programskim modulima (engl. *plugins*).
- Ima veliki fond programskih modula *(http://plugins.jquery.com/)* koji se postojano širi otkad se biblioteka JQuery pojavila.
- Njen API je potpuno dokumentovan, uključujući primere ugrađenog koda, što je u svetu biblioteka JavaScripta luksuz. Ma, bilo kakva dokumentacija je već godinama luksuz.
- Namenski je napravljena tako da se izbegnu konflikti s drugim JavaScript bibliotekama.
- Podrška korisničke zajednice je prilično upotrebljiva, i obuhvata liste slanja, IRC kanale i ogromne količine uputstava, članaka i blogova.

- Razvija se otvoreno, što znači da svako može doprineti ispravljanju grešaka, unapređenju i razvoju.
- Razvija se postojano i konsistentno, što znači da razvojni tim ne beži od objavljivanja ažuriranih verzija.
- Činjenica da su je prihvatile velike organizacije (na primer, Microsoft, Dell, Bank of America, Digg, CBS, Netflix) doprineće, kao i do sada, njenoj dugotrajnosti i stabilnosti.
- Usvaja specifikacije organizacije W3C pre veb čitača. Evo primera: jQuery podržava veliki deo CSS3 selektora.
- Trenutno se testira i optimizuje za razvoj na modernim čitačima (Chrome 1, Chrome Nightly, IE 6, IE 7, IE 8, Opera 9.6, Safari 3.2, WebKit Nightly, Firefox 2, Firefox 3, Firefox Nightly).
- Moćna je alatka u rukama i dizajnera i programera – jQuery ne pravi razlike.
- Njena elegancija, metodologije i filozofije koje menjaju način na koji se piše JavaScript kôd, postaju standard sam za sebe. Setimo se samo koliko je mnogo drugih rešenja pozajmilo šeme selektora i ulančavanja od JQueryja.
- Neobjašnjiva ugodnost programiranja koju neplanirano izaziva, zarazna je i nezaobilazna; izgleda da se čak i kritičari zaljubljuju u karakteristike biblioteke jQuery.
- Njena dokumentacija sadrži mnoge elemente (na primer, API čitač, aplikacije kontrolne table, podsetnike) uključujući i API čitač van mreže (AIR aplikacija).
- Namerno je prilagođena kako bi mogla da služi za programiranje na nenametljivom JavaScriptu.
- Ostala je, u srži, JavaScript biblioteka (a ne skelet, engl. *framework*), ali uporedo s njom postoji i srodan projekat za elemente grafičkog korisničkog interfejsa i razvoj aplikacija (jQuery UI).
- Kriva učenja je povoljna jer se JQuery nadograđuje na koncepte koje većina projektanata i dizajnera već razume (na primer, CSS i HTML).

Po mom mišljenju, jQuery se izdvaja od drugih rešenja zbog kombinacije pomenutih svojstava, a ne zbog bilo kog od njih posebno. jQuery paket je, naprosto, neprikosnovena alatka za programiranje na JavaScriptu.

Filozofija biblioteke jQuery

Filozofija biblioteke jQuery je „Piši manje, uradi više“ (engl. *write less, do more*). Ta filozofija se može dalje razložiti u tri koncepta:

- Nalaženje određenih elemenata (pomoću CSS selektora) i njihova upotreba na neki način (pomoću jQuery metoda)
- Ulančavanje više jQuery metoda radi primene nad skupom elemenata
- Primena jQuery omotača (engl. *wrapper*) i posredne (implicitne) iteracije

Neophodno je da dobro razumete ova tri koncepta kako biste mogli sami da pišete jQuery kôd ili da prilagođavate recepte iz ove knjige. Pregledajmo detaljno svaki od ta tri koncepta.

Naći određene elemente i uraditi nešto s njima

Ili, konkretnije, nađite niz elemenata u DOM-u, a potom uradite nešto s tim nizom elemenata. Na primer, razmotrimo situaciju u kojoj želite da sakrijete element `<div>` od korisnika, da učitate neki nov tekstualni sadržaj u taj skriveni element, izmenite atribut odabranog elementa, i da, na kraju, učinite taj element `<div>` ponovo vidljivim.

Poslednja rečenica bi, prevedena u jQuery kôd, izgledala otprilike ovako:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
</head>
<body>
<div>old content</div>
<script>

//sakrij sve elemente div na strani
jQuery('div').hide();

//ažuriraj tekst unutar svakog elementa div
jQuery('div').text('new content');

//dodaj atribut klase s vrednošću promenljive updatedContent svim div elementima
jQuery('div').addClass("updatedContent");

//prikaži sve elemente div na strani
jQuery('div').show();

</script>
</body>
</html>
```

Razmotrimo ova četiri jQuery iskaza:

- Svi elementi `<div>` na strani skrivaju se od korisnikovog pogleda.
- Tekst unutar skrivenih elemenata `<div>` zamenjuje se novim tekstom (`new content`).
- Elementi `<div>` se ažuriraju tako što im se dodeljuje nov atribut (`class`) i vrednost (`updatedContent`).
- Prikazuju se elementi `<div>` na strani tako da ih korisnici opet vide.

Ako vam sintaksa ovog segmenta jQuery koda izgleda mistično, ne brinite. Udubićemo se u osnove kad dođemo do prvog recepta u knjizi. Ponavljam – iz ovog primera

treba zapamtiti jQuery koncept „nalaženja nekih elemenata i obavljanja nečeg s njima“. U ovom segmentu koda samo nalazimo sve elemente `<div>` na HTML strani pomoću funkcije `jQuery()`), a potom nešto radimo s njma koristeći jQuery metode (na primer, `hide()`, `text()`, `addClass()`, `show()`).

Ulančavanje

jQuery omogućava ulančavanje metoda. Na primer, mogli bismo da nađemo neki element, a potom da ulančamo operacije nad njim. Prethodni segment koda, čija je svrha bila da se nađu neki elementi i da se uradi nešto s njima, mogla bi se pomoću ulančavanja (engl. *chaining*) svesti na samo jedan JavaScript iskaz.

Primer bi se pomoću ulančavanja mogao izmeniti od ovoga:

```
//sakrij sve elemente div na stranici
jQuery('div').hide();

//ažuriraj tekst unutar svakog elementa div
jQuery('div').text('new content');

//dodaj atribut klase s vrednošću promenljive updatedContent svim div elementima
jQuery('div').addClass("updatedContent");

//prikazuje sve elemente div na strani
jQuery('div').show();
```

u ovo:

```
jQuery('div').hide().text('new content').addClass("updatedContent").show();
```

ili, uz uvlačenje i prelome reda, u ovo:

```
jQuery('div')
    .hide()
    .text('new content')
    .addClass("updatedContent")
    .show();
```

Jednostavno rečeno, ulančavanje vam omogućava da primenite beskrajan lanac jQuery metoda na trenutno izabrane elemente (one koji su trenutno omotani funkcijom jQuery) pomoću funkcije `jQuery()` . Funkcija `jQuery()` primenjena na izabrane elemente uvek vraća te elemente kako bi lanac mogao da se nastavi. Kao što ćete videti u narednim receptima, i dodatni moduli se prave na ovaj način (tako da kao rezultat daju omotane elemente), pa primena dodatnog modula ne prekida lanac.

Ukoliko niste uočili, skrećemo vam pažnju na sledeće: kao što razmotreni segment koda pokazuje, ulančavanje takođe optimizuje obradu tako što se DOM elementi biraju samo jednom, a zatim se, pomoću ulančavanja, nad njima mnogo puta izvršavaju jQuery metode. Izbegavanje nepotrebnog kretanja po DOM strukturi, najvažniji je deo poboljšavanja performansi strane. Kad god je moguće, izabrane DOM elemente koristite iznova ili ih uskladištite.

jQuery omotački skup

Dobar deo vremena u radu s bibliotekom jQuery, dobijaćete ono što se zove *omotač* (engl. *wrapper*). Drugim rečima, biraćete DOM elemente s HTML strane omotane funkcijom `jQuery()`. Njih često nazivam „omotačkim skupom“ ili „omotanim skupom“ (engl. *wrapper set*) jer je reč o skupu elemenata omotanom funkcijom `jQuery()`. Ponekad će omotački skup da sadrži samo jedan DOM element, dok će u drugim situacijama biti više elemenata. Biće slučajeva kada će omotački skup biti bez elemenata. U takvim situacijama, metode/svojstva biblioteke jQuery neće se uspešno izvršiti ukoliko se pozovu za prazan omotački skup, što može biti korisno za izbegavanje nepotrebnih iskaza `if`.

Na osnovu segmenta koda kojim smo predstavili koncept nalaženja elemenata i obavljanja nečeg s njima, šta mislite da bi se desilo kad bismo dodali više elemenata `<div>` na veb stranu? U narednom ažuriranom primeru koda, dodao sam tri nova elementa `<div>` na HTML stranu, tako da imamo ukupno četiri elementa `<div>`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<script type="text/JavaScript" src="http://ajax.googleapis.com/ajax/libs/
jquery/1.3.0/jquery.min.js"></script> </head>
<body>
<div>old content</div>
<div>old content</div>
<div>old content</div>
<div>old content</div>
<script>
//sakrij sve elemente div na stranici
jQuery('div').hide().text('new content').addClass("updatedContent").show();

</script>
</body>
</html>
```

Možda niste direktno napisali nijednu programsku petlju, ali evo šta se dešava: jQuery pretražuje stranu i zamenjuje sve elemente `<div>` u omotačkom skupu tako da se jQuery metode koje koristim izvršavaju (implicitna iteracija) nad svakim DOM elementom u skupu. Na primer, metoda `.hide()` primenjuje se na svaki element skupa. Ukoliko ponovo pogledamo naš segment koda, videćete da će se svaka metoda koju primenimo izvršiti nad svakim elementom `<div>` na strani. To je kao kada biste na ovom mestu napisali petlju koja poziva svaku jQuery metodu za svaki DOM element. Ažurirani primer u kodu sakriće svaki element `<div>` na strani, upisati u njega novi tekst, dati novu mu vrednost klase i ponovo ga učiniti vidljivim.

Omotački skup i njegov podrazumevani sistem petlje (implicitna iteracija) od presudne su važnosti za uspostavljanje naprednih koncepata izvođenja petlje. Podsećamo da se jednostavna petlja na ovom mestu javlja pre nego što ste uopšte napravili konkretnu

petlju (na primer, `jQuery('div').each(function(){})`. Drugim rečima, svaka pozvana jQuery metoda obično će menjati svaki element omotačkog skupa.

Međutim, ima situacija u kojima pozvana jQuery metoda utiče samo na prvi element a ne na sve elemente u omotačkom skupu (na primer, `attr()`) – s tim ćemo vas upoznati u narednim poglavljima.

Kako je organizovan jQuery interfejs za programiranje aplikacija (API)

Kad sam počinjao s blibliotekom jQuery, glavni razlog zašto sam je odabrao za JavaScript biblioteku bez sumnje je bio taj što je bila dokumentovana na odgovarajući način (i sa bezbroj dodatnih modula!) Kasnije sam shvatio da je drugi faktor koji me je zauvek vezao za jQuery bila činjenica da je API organizovan u logične kategorije. Bilo je dovoljno samo da vidim kako je API organizovan i da prepoznam funkcionalnost koja mi je bila potrebna.

Pre nego što se bacite na jQuery, predlažem da pregledate dokumentaciju dostupnu na internetu (*http://docs.jquery.com/Main_Page*) i da prvo samo usvojite organizaciju interfejsa za programiranje. Ako razumete kako je API organizovan, brže ćete u dokumentaciji nalaziti potrebne informacije, što je prilična prednost s obzirom na to da ima zaista mnogo načina da se rešenje izvede u jQuery kodu. Toliko je robustan da je lako uplesti se u implementaciju jer se jedan problem može rešiti na mnogo načina. Navešću sada kako je API organizovan. Predlažem da zapamtite strukturu interfejsa ili barem najopštije kategorije.

- jQuery jezgro
  - – funkcija `jQuery()`
  - – jQuery pristupne metode
  - – Podaci
  - – Dodatni moduli
  - – Interoperabilnost
- Selektori
  - – Osnove
  - – Hijerarhija
  - – Osnovni filtri
  - – Filtri sadržaja
  - – Filtri vidljivosti
  - – Filtri atributa
  - – Filtri direktnih potomaka (dece)
  - – Obrasci
  - – Filtri obrazaca

- Atributi
  - Attr
  - Klasa
  - HTML
  - Tekst
  - Vrednost
- Kretanje
  - Filtriranje
  - Nalaženje
  - Ulančavanje
- Manipulisanje
  - Izmena sadržaja
  - Unetanje unutra
  - Umetanje spolja
  - Umetanje okolo
  - Zamena
  - Uklanjanje
  - Kopiranje
- CSS
  - CSS
  - Pozicioniranje
  - Visina i širina
- Događaji
  - Učitavanje strane
  - Obrada događaja
  - Živi događaji
  - Pomoćnici za interakcije
  - Pomoćnici za događaje
- Efekti
  - Osnove
  - Klizanje
  - Postepeno nestajanje
  - Namenski
  - Podešavanja

- Ajax
  – AJAX zahtevi
  – AJAX događaji
  – Razno
- Uslužne operacije
  – Prepoznavanje čitača i odlika
  – Operacije nad nizovima i objektima
  – Operacije testiranja
  – Operacije nad znakovnim nizovima
  – URL adrese

Pre nego što pređemo na niz osnovnih jQuery recepata, hteo bih da pomenem da se recepti u ovom poglavlju nadovezuju jedan na drugi. Drugim rečima, znanje će vam se uvećavati na logičan način kako budete prelazili s jednog recepta na naredni. Predlažem da u prvom navratu ove recepte čitate redom – od 1.1 do 1.17.

1.1 Uključivanje koda biblioteke jQuery u HTML stranicu

Problem

Želite da na veb stranici koristite JavaScript biblioteku jQuery.

Rešenje

Trenutno postoje dva idealna rešenja za uključivanje biblioteke jQuery u veb stranicu:

- Upotrebite mrežu za isporučivanje sadržaja (engl. *content delivery network*, CDN) smeštenu na Googleu da biste uključili neku verziju jQueryja (rešenje u ovom poglavlju).
- Preuzmite sopstvenu verziju biblioteke jQuery s lokacije jQuery.com i smestite je na svoj server ili u lokalni sistem datoteka.

Objašnjenje

JavaScript biblioteka jQuery uključuje se u stranicu kao bilo koja spoljna JavaScript datoteka. Možete jednostavno upotrebiti HTML element `<script>` i dodeliti vrednost (URL adresu ili putanju direktorijuma) njegovom atributu `src=""`, i spoljna datoteka koju povezujete biće uključena (engl. *included*) u veb stranicu. Kao primer navodimo šablon (engl. *template*) koji uključuje biblioteku jQuery omogućavajući da započnete bilo koji jQuery projekat:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
</head>
<body>
<script type="text/JavaScript">
    alert('jQuery ' + jQuery.fn.jquery);
</script>
</body>
</html>
```

Primećujete da koristim minimizovanu verziju biblioteke jQuery sa Googlea (koju preporučujem za javne stranice). Međutim, otklanjanje JavaScript grešaka u minimizovanom kodu nije idealno. Tokom projektovanja ili pisanja koda, možda je bolje koristiti neminimizovanu verziju s Googlea zbog otkrivanja potencijalnih grešaka u JavaScriptu. Više informacija o verziji biblioteke jQuery s Googlea naći ćete na veb lokaciji za API-je Ajaxovih biblioteka: *http://code.google.com/apis/ajaxlibs/*.

Naravno, moguće je (a verovatno i zastarelo) da na svom računaru sačuvate kopiju biblioteke jQuery. Međutim, to bi u većini slučajeva bilo blesavo jer je Google bio dovoljno ljubazan da je čuva za vas. Ako koristite verziju biblioteke jQuery sa Googlea, radite sa stabilnom, pouzdanom, brzom i globalno dostupnom kopijom ove biblioteke. Dodatna korist je smanjeno kašnjenje, veći paralelizam i bolje keširanje. To se, naravno, može ostvariti i bez Googleovog rešenja, ali bi vas verovatno koštalu koju paru.

Moguće je i da iz nekog razloga ne želite da koristite Googleovu verziju biblioteke jQuery. Možda želite prilagođenu verziju ili nemate potrebe (ili mogućnosti) da koristite internet kada radite s bibliotekom. Možda naprosto smatrate da je Google „Veliki brat" koji prati sve što radite ili ne želite da koristite njegove resurse jer volite da sve sami držite pod kontrolom. Dakle, oni kojima Googleova verzija biblioteke jQuery nije potrebna ili naprosto ne žele da je koriste, mogu preuzeti jQuery s veb lokacije jQuery.com (*http://docs.jquery.com/Downloading_jQuery*) i smestiti je lokalno na sopstveni server ili u lokalni sistem datoteka. Prema šablonu koji smo naveli u ovom receptu, zamenili biste atribut `src` URL adresom ili putanjom direktorijuma jQuery JavaScript datoteke koju ste preuzeli.

1.2 Izvršavanje jQuery/JavaScript koda pošto je DOM učitan ali pre potpunog učitavanja strane

Problem

Moderne JavaScript aplikacije s nenametljivim JavaScript metodologijama obično izvršavaju JavaScript kôd tek kada je DOM potpuno učitan. Realno, DOM se mora učitati pre nego što se nad njim izvrši bilo koja operacija manipulisanja ili kretanja po njemu. Potreban je način da se odredi kada je klijent, najčešće veb čitač, potpuno učitao DOM

ali ne i sve elemente poput slika i SWF datoteka. Ako bismo u ovoj situaciji koristili događaj `window.onload`, čitav dokument – zajedno sa svim elementima – morao bi da se učita potpuno pre pokretanja događaja `onload`. Za mnoge korisnike veb čitača, to traje predugo. Treba nam događaj koji će nam reći kada je samo DOM spreman za operacije manipulisanja i kretanja po njemu.

Rešenje

jQuery ima namensku metodu `ready()` za obradu događaja koja je obično vezana za DOM objekat dokumenta. Metodi `ready()` prosleđuje se jedan parametar – funkcija koja sadrži JavaScript kôd koji treba da se izvrši pošto je DOM spreman za operacije manipulisanja i kretanja po njemu. Evo jednostavnog primera u kome ovaj događaj otvara `alert()` prozor pošto je DOM spreman, ali pre nego što se stranica potpuno učita:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    jQuery(document).ready(function(){//DOM nije učitan, mora da se koristi događaj
        ready alert(jQuery('p').text());
    });
</script>
</head>
<body>
<p>The DOM is ready!</p>
</body>
</html>
```

Objašnjenje

Metoda za obradu događaja `ready()` zamenjuje biblioteku jQuery za događaj `window.onload` jezgra JavaScripta. Može se primenjivati proizvoljno mnogo puta. Kada se koristi ovaj namenski događaj, preporučuje se da se uključi u veb strane nakon deklaracija opisa stilova i iskaza `include`. Na taj način, sva svojstva elementa biće ispravno definisana pre nego što događaj `ready()` izvrši bilo kakav jQuery kôd ili JavaScript kôd.

Pored toga, funkcija `jQuery()` sama po sebi ima prečicu za primenu događaja `ready`. Koristeći tu prečicu, primer s metodom `alert()` može se napisati na sledeći način:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
```

```
<script type="text/JavaScript">
    jQuery(function(){ //DOM nije učitan, mora da se koristi događaj ready
        alert(jQuery('p').text());
    });
</script>
</head>
<body>
<p>The DOM is ready!</p>
</body>
</html>
```

Namenski jQuery događaj neophodno je koristiti samo ako treba ugraditi (engl. *embed*) JavaScript kôd u tok dokumenta na vrhu strane i kapsulirati ga u element `<head>`. Primenu događaja `ready()` izbegavam tako što sve JavaScript iskaze `include` i `inline` uvodim pre zatvaranja elementa `<body>`. To radim iz dva razloga.

Pre svega, po modernim tehnikama optimizacije, strane se učitavaju brže kada veb čitač učitava JavaScript na kraju raščlanjivanja (engl. *parsing*) strane. Drugim rečima, ako JavaScript kôd stavite na kraj veb strane, veb čitač će učitati sve ispred njega pre nego što učita taj kôd. To je dobro jer će većina čitača obično prestati da obrađuje druge inicijative za učitavanje dok JavaScript mehanizam ne prevede JavaScript kôd na veb stranici. To je neka vrsta uskog grla, pošto se JavaScript nalazi na vrhu dokumenta veb strane. Naravno, u nekim situacijama lakše je staviti JavaScript u element `<head>`. Ali, iskreno govoreći, nikada nisam video situaciju gde je to apsolutno neophodno. Svaka prepreka smeštanju JavaScript koda na kraj strane na koju sam naišao u mom projektantskom iskustvu, lako je uklonjena uz velike dobitke u optimizaciji.

Drugo, ako su nam cilj brze veb strane, zašto razvijati rešenje za situaciju koja se može izbeći jednostavno – premeštanjem koda na dno strane? Kad moram da biram između više i manje koda, biram ovo drugo. Ako ne koristimo događaj `ready()`, skraćujemo kôd; kraći kôd će se izvršavati brže od dužeg.

Evo primera koda s događajem `alert()` bez primene događaja `ready()` (izostavili smo neka objašnjenja):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p>The DOM is ready!</p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    alert(jQuery('p').text());//samo napred, DOM je učitan
</script>
</body>
</html>
```

Skrećem pažnju na to da sam sav JavaScript kôd smestio ispred završne oznake `</body>`. Sve dodatne oznake bi trebalo staviti iznad JavaScript koda u HTML dokumentu.

1.3 Biranje DOM elemenata pomoću selektora i funkcije jQuery()

Problem

Treba da izaberete jedan DOM element i/ili skup DOM elemenata kako biste na elemente primenjivali jQuery metode.

Rešenje

jQuery pruža dve mogućnosti za biranje DOM elemenata. U obe varijante koristi se funkcija `jQuery()` ili njen alijas `$()`. Prvo predstavljamo daleko ubedljivije rešenje koje se mnogo češće primenjuje: u njemu se koriste CSS selektori i namenski selektori (engl. *custom selectors*). Kada se funkciji `jQuery()` prosledi znakovni niz (engl. *string*) sa selektorskim izrazom, ona će da pročešljava DOM i da locira DOM čvorove. U narednom primeru biraju se svi elementi `<a>` u HTML dokumentu:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    //ispisuje da ima 6 elemenata
    alert('Page contains ' + jQuery('a').length + ' <a> elements!');
</script>
</body>
</html>
```

Ako biste ovu HTML stranu učitali u veb čitač, videli biste da ovaj segment koda izvršava funkciju `alert()` koja ispisuje poruku da stranica sadrži šest elemenata `<a>`. Tu vrednost sam prosledio metodi `alert()` tako što sam prvo izabrao sve elemente `<a>`, a potom pomoću svojstva `length` prosledio broj elemenata u jQuery omotačkom skupu.

Skrećem pažnju da će prvi parametar funkcije `jQuery()`, kako je korišćen u ovom primeru, prihvatiti i više izraza. Za to je potrebno da razdvojite selektore zarezima unutar znakovnog niza koji je prosleđen kao prvi parametar funkciji `jQuery()`. Evo primera kako bi to moglo da izgleda:

```
jQuery('selector1, selector2, selector3').length;
```

Druga, manje korišćena mogućnost za biranje DOM elemenata jeste da se funkciji `jQuery()` prosledi JavaScript referenca na DOM element(e). Kao primer, u narednom segmentu koda izabraćemo sve elemente `<a>` u HTML dokumentu. Primetićete da funkciji `jQuery()` prosleđujem niz elemenata `<a>` skupljenih pomoću DOM metode `getElementsByTagName`. Rezultat ovog primera je isti kao rezultat prethodnog segmenta koda:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body bgcolor="yellow"> <!-- taj atribut je zastareo, znam, ali idemo s njim -->
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
   //ispisuje poruku da ima 6 elemenata
    alert('Page contains ' + jQuery(document.getElementsByTagName('a')).length +
' <a> Elements, And has a '
     + jQuery(document.body).attr('bgcolor') + ' background');
</script>
</body>
</html>
```

Objašnjenje

jQuery svoju čuvenu snagu delom duguje svom selektorskom mehanizmu, Sizzlu (*http://sizzlejs.com/*), koji bira DOM elemente iz HTML dokumenta. Iako opcija prosleđivanja DOM referenci funkciji `jQuery()` postoji (a lepo ju je imati kad zatreba), nije to ono zbog čega je biblioteka jQuery privukla opštu pažnju. jQuery je jedinstven zbog svojih brojnih i moćnih opcija za rad sa selektorima.

Sve do kraja knjige učićete o moćnim i robusnim selektorima. Kada se susretnete s nekim selektorom, bitno je da razumete njegovu funkciju. To znanje će vam mnogo koristiti u budućim programerskim poduhvatima s bibliotekom jQuery.

1.4 Biranje DOM elemenata unutar zadatog konteksta

Problem

Treba da referencirate jedan DOM element ili skup DOM elemenata u kontekstu drugog DOM elementa ili dokumenta kako biste primenili jQuery metode na te elemente.

Rešenje

Funkcija `jQuery()` kojoj se prosledi CSS izraz prihvatiće i drugi parametar koji je upućuje na kontekst u kome treba da traži DOM elemente na osnovu izraza. U ovom slučaju, drugi parametar može biti DOM referenca, jQuery omotač ili dokument. U narednom segmentu koda ima dvanaest elemenata `<input>`. Obratite pažnju na to kako pomoću datog konteksta na osnovu elementa `<form>` biram samo određene elemente `<input>`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>

<form>
<input name="" type="checkbox" />
<input name="" type="radio" />
<input name="" type="text" />
<input name="" type="button" />
</form>

<form>
<input name="" type="checkbox" />
<input name="" type="radio" />
<input name="" type="text" />
<input name="" type="button" />

</form>
<input name="" type="checkbox" />
<input name="" type="radio" />
<input name="" type="text" />
<input name="" type="button" />

<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">

//pretražuje po svim elementima obrasca, koristeći omotač za kontekst,
//ispisuje "selected 8 inputs"
alert('selected ' + jQuery('input',$('form')).length + ' inputs');
```

```
//traži unutar prvog obrasca, koristeći DOM referencu kao kontekst,
//ispisuje "selected 4 inputs"
alert('selected ' + jQuery('input',document.forms[0]).length + ' inputs');

//traži po body elementu sve input elemente pomoću izraza,
//ispisuje "selected 12 inputs",
alert('selected ' + jQuery('input','body').length + ' inputs');

</script>
</body>
</html>
```

Objašnjenje

Kao što je pomenuto u rešenju iz ovog recepta, moguće je izabrati dokumente kao kontekst za pretraživanje. Na primer, moguće je pretraživati u okviru konteksta XML dokumenta koji je poslat usled XHR zahteva (Ajax). Više detalja o toj opciji naći ćete u poglavlju 16.

1.5 Filtriranje omotačkog skupa DOM elemenata

Problem

Potrebno je da iz jQuery omotačkog skupa izabranih DOM elemenata uklonite one koji ne odgovaraju novom navedenom izrazu (ili izrazima) kako biste dobili nov skup elemenata s kojima ćete raditi.

Rešenje

jQuery ima metodu za filtriranje koja se primenjuje na jQuery omotački skup DOM elemenata da bi se iz njega izostavili elementi koji *ne odgovaraju* zadatom izrazu (ili izrazima). Ukratko, metoda `filter()` omogućava da filtrirate tekući skup elemenata. Ova metoda se razlikuje od jQuery metode `find()` koja redukuje omotački skup DOM elemenata tako što nalazi nove elemente (preko novog selektorskog izraza), uključujući elemente decu tekućeg omotačkog skupa.

Naredni segment koda pomoći će vam da bolje razumete metodu `filter()`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<a href="#" class="external">link</a>
<a href="#" class="external">link</a>
<a href="#"></a>
<a href="#" class="external">link</a>
<a href="#" class="external">link</a>
```

```
<a href="#"></a>
<a href="#">link</a>
<a href="#">link</a>
<a href="#">link</a>
<a href="#">link</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">

    //ispisuje da su u skupu ostala četiri elementa
        alert(jQuery('a').filter('.external').length + ' external links');
</script>
</body>
</html>
```

HTML stranica iz navedenog primera sadrži veb stranu sa deset elemenata `<a>`. Za spoljne veze, klasa se zove `external`. Pomoću funkcije `jQuery()` biramo sve elemente `<a>` na strani. Potom, pomoću metode `filter()`, iz izvornog skupa uklanjamo sve elemente čiji atribut `class` nema vrednost `external`. Kada se prvobitni skup DOM elemenata izmeni pomoću metode `filter()`, pozivamo svojstvo `length` koje sadrži informaciju o broju preostalih elemenata u skupu pošto je filter primenjen.

Objašnjenje

Metodi `filter()` moguće je poslati i funkciju koja se može upotrebiti za filtriranje omotačkog skupa. Prethodni primer u kome se metodi `filter()` prosleđuje znakovni izraz, može se izmeniti tako da se umesto toga koristi funkcija:

```
alert(
    jQuery('a')
        .filter(function(index){ return $(this).hasClass('external');})
        .length + ' external links'
);
```

Metodi `filter()` se, u ovom slučaju, prosleđuje anonimna funkcija. Ona se poziva s kontekstom jednakim tekućem elementu. To znači da kada upotrebim referencu `$(this)` u funkciji, zapravo referenciram svaki DOM element u omotačkom skupu. Unutar funkcije, proveravam svaki element `<a>` u omotačkom skupu kako bih proverio da li je rezultat metode `hasClass()` jednak `external`. Ako je tako, logička promenljiva ima vrednost `true` i element će ostati u skupu, a ako nije (`false`), element se uklanja iz skupa. Drugi način je da se, ukoliko je rezultat funkcije `false`, element uklanja. Ako je rezultat funkcije bilo šta drugo, element ostaje u omotačkom skupu.

Možda ste primetili da sam funkciji prosledio parametar `index` koji ne koristim. Ovaj parametar se po potrebi može koristiti da se numerički referencira indeks elementa u jQuery omotačkom skupu.

1.6 Nalaženje elemenata potomaka u trenutno odabranom omotačkom skupu

Problem

Treba naći potomke (decu) DOM elemenata (ili jednog elementa) u kontekstu trenutno odabranih elemenata.

Rešenje

Pomoću metode `.find()` napravite omotački skup elemenata na osnovu konteksta tekućeg skupa i potomaka elemenata tog skupa. Na primer, radite s veb stranom koja sadrži nekoliko pasusa. U ovim pasusima reči su naglašene (kurzivne). Ako želite da izaberete samo elemente `<em>` unutar elemenata `<p>`, to biste mogli uraditi na sledeći način:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p>Ut ad videntur facilisis <em>elit</em> cum. Nibh insitam erat facit
<em>saepius</em> magna. Nam ex liber iriure et imperdiet. Et mirum eros
iis te habent. </p>
<p>Claram claritatem eu amet dignissim magna. Dignissim quam elit facer eros
illum. Et qui ex esse <em>tincidunt</em> anteposuerit. Nulla nam odio ii
vulputate feugait.</p>
<p>In quis <em>laoreet</em> te legunt euismod. Claritatem <em>consuetudium</em>
wisi sit velit facilisi.</p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
 //ispisuje ukupan broj kurzivnih reči unutar elemenata <p>
    alert('The three paragraphs in all contain ' +
    jQuery('p').find('em').length + '
italic words');
</script>
</body>
</html>
```

Prethodno rešenje smo mogli da izvedemo i tako što bismo funkciji `jQuery()` prosledili kontekstnu referencu kao drugi parametar:

```
alert('The three paragraphs in all contain ' + jQuery('em',$('p')).length +
' italic words');
```

Pored toga, valja pomenuti da su poslednja dva segmenta koda navedeni kao ilustracija. Logičnije, ako ne pragmatičnije, bilo bi da se pomoću CSS selektora odaberu svi kurzivni elementi potomci unutar elemenata `<p>` predaka.

```
alert('The three paragraphs in all contain ' + jQuery('p em').length +
' italic words');
```

Objašnjenje

Pomoću jQuery metode `.find()` može se napraviti nov skup elemenata na osnovu konteksta tekućeg skupa DOM elemenata i njihove dece. Metode `.filter()` i `.find()` često se mešaju. Najlakše ih je razlikovati po tome što metoda `.find()` radi s decom tekućeg skupa (bira ih), dok metoda `.filter()` radi samo s tekućim skupom elemenata. Drugim rečima, ako želite da izmenite tekući omotački skup koristeći ga kao kontekst za dalju selekciju dece izabranih elemenata, upotrebite metodu `.find()`. Ukoliko samo hoćete da filtrirate tekući omotački skup i da dobijete nov podskup isključivo od DOM elemenata u tom skupu, upotrebite `.filter()`. Možemo razliku da opišemo i ovako – metoda `find()` vraća decu, dok metoda `filter()` samo filtrira elemente u tekućem omotačkom skupu.

1.7 Vraćanje prethodnoj selekciji pre destruktivne promene

Problem

Potrebno je eliminisati destruktivnu jQuery metodu (na primer, `filter()` ili `find()`) primenjenu na skup elemenata da bi se skup vratio u stanje pre primene destruktivne metode i da bi se potom s njim moglo raditi kao da destruktivna metoda nije pozvana.

Rešenje

jQuery ima metodu `end()` pomoću koje se skup DOM elemenata vraća u stanje pre izvršenja destruktivne metode. Naredni segment HTML koda pomoći će vam da shvatite kako funkcioniše metoda `end()`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p>text</p>
<p class="middle">Middle <span>text</span></p>
<p>text</p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    alert(jQuery('p').filter('.middle').length); //alerts 1
    alert(jQuery('p').filter('.middle').end().length); //alerts 3
    alert(jQuery('p').filter('.middle').find('span')
.end().end().length); //alerts 3
</script>
</body>
</html>
```

Prvi `alert()` iskaz je jQuery iskaz koji traži po dokumentu sve elemente `<p>`, a potom primenjuje metodu `filter()` na izabrane elemente `<p>` u skupu tako da se biraju samo elementi čiji atribut `class` ima vrednost `middle`. Posle toga, svojstvo `length` prijavljuje koliko je elemenata ostalo u skupu:

```
alert(jQuery('p').filter('.middle').length); //ispisuje broj elemenata u skupu
```

U narednom `alert()` iskazu koristi se metoda `end()`. Ovde se radi sve što je rađeno u prethodnom iskazu osim što se poništava dejstvo metode `filter()`, a rezultat je omotački skup elemenata kakav je bio pre primene metode `filter()`:

```
alert(jQuery('p').filter('.middle').end().length); //ispisuje broj elemenata u skupu
```

Poslednji `alert()` iskaz pokazuje kako se metoda `end()` primenjuje dvaput da bi se eliminisale destruktivne promene i `filter()` i `find()`, tako da se omotačkom skupu vraća početni sastav:

```
alert(jQuery('p').filter('.middle').find('span').end().end().length); //ispisuje broj
                                                                      //elemenata u skupu
```

Objašnjenje

Ako se metoda `end()` upotrebi u slučaju kada nije bilo prethodnih destruktivnih promena, vraća se prazan skup. Destruktivna operacija je svaka operacija koja menja skup izabranih jQuery elemenata, dakle svaka metoda za kretanje ili manipulisanje čiji je rezultat jQuery objekat, uključujući metode `add()`, `andSelf()`, `children()`, `closes()`, `filter()`, `find()`, `map()`, `next()`, `nextAll()`, `not()`, `parent()`, `parents()`, `prev()`, `prevAll()`, `siblings()`, `slice()`, `clone()`, `appendTo()`, `prependTo()`, `insertBefore()`, `insertAfter()` i `replaceAll()`.

1.8 Dodavanje prethodne selekcije tekućoj selekciji

Problem

Primetili ste metode na skup elemenata da biste dobili nov skup elemenata. Međutim, želite da radite i s prethodnim i s novim skupom.

Rešenje

Prethodnu selekciju DOM elemenata možete kombinovati s tekućom selekcijom pomoću metode `andSelf()`. Na primer, u narednom segmentu koda prvo biramo sve elemente `<div>` na strani. Posle toga, u tom skupu elemenata nalazimo sve elemente `<p>` unutar elemenata `<div>`. Da bismo radili i sa elementima `<div>` i sa elementima `<p>` nađenim u elementima `<div>`, mogli bismo uključiti elemente `<div>` u tekući skup pomoću metode `andSelf()`. Da sam izostavio metodu `andSelf()`, ivična boja bi se mogla primeniti samo na elemente `<p>`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<div>
<p>Paragraph</p>
<p>Paragraph</p>
</div>
<script type="text/JavaScript" src="http://ajax.googleapis.com/
ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    jQuery('div').find('p').andSelf().css('border','1px solid #993300');
</script>
</body>
</html>
```

Objašnjenje

Ne zaboravite sledeće: kada primenite metodu `andSelf()`, tekućem skupu se dodaje samo prethodni skup, ne i svi prethodni skupovi.

1.9 Kretanje po DOM strukturi na osnovu tekućeg konteksta da bi se dobio nov skup DOM elemenata

Problem

Izabrali ste skup DOM elemenata, i na osnovu pozicije selekcija unutar DOM strukture stabla, želite da pročešljate DOM kako biste dobili nov skup elemenata s kojim ćete raditi.

Rešenje

jQuery ima nekoliko metoda za kretanje po DOM-u na osnovu konteksta trenutno izabranih DOM elemenata.

Na primer, razmotrimo naredni segment HTML koda:

```
<div>
<ul>
<li><a href="#">link</a></li>
<li><a href="#">link</a></li>
<li><a href="#">link</a></li>
<li><a href="#">link</a></li>
</ul>
</div>
```

Hajde da odaberemo drugi element `<li>` pomoću namenskog selektora indeksa `:eq()`:

```
//bira drugi element iz skupa elemenata <li> po indeksu, početni indeks je 0
jQuery('li:eq(1)');
```

Sada imamo kontekst – početnu tačku unutar HTML strukture. Naša početna tačka je drugi element `<li>`. Odatle možemo da se krećemo bilo kuda – zapravo, gotovo bilo kuda. Pogledajmo gde možemo da idemo pomoću nekoliko jQuery metoda za kretanje po DOM-u. Komentari objašnjavaju postupke:

```
jQuery('li:eq(1)').next() // bira treći element <li>

jQuery('li:eq(1)').prev() // bira prvi element <li>

jQuery('li:eq(1)').parent() // bira sve elemente <ul>

jQuery('li:eq(1)').parent().children() // bira sve elemente <li>

jQuery('li:eq(1)').nextAll() // bira sve elemente <li> posle drugog elementa <li>

jQuery('li:eq(1)').prevAll() // bira sve elemente <li> pre drugog elementa <li>
```

Podsećamo da ove metode za kretanje stvaraju nov omotački skup, a prethodnom omotačkom skupu možete se vratiti pomoću metode `end()`.

Objašnjenje

Do sada smo metode za kretanje koristili za jednostavne pomake. Kada je reč o kretanju, postoje dva dodatna koncepta koja je važno savladati.

Prvi i verovatno najočigledniji koncept jeste da se metode za kretanje mogu ulančavati. Razmotrimo ponovo naredni jQuery iskaz:

```
jQuery('li:eq(1)').parent().children() //bira sve elemente <li>
```

Primetićete da sam od drugog elementa `<li>` prešao na roditelja, element `<ul>`, a potom od roditelja izabrao svu decu elementa `<ul>`. jQuery omotački skup sadržaće sve elemente `<li>` kojih ima u elementu `<ul>`. Naravno, ovo je namenski primer čija je svrha da se prikaže kako funkcionišu metode za kretanje. Da nam je zaista trebao omotački skup sa samo elementima `<li>`, mnogo jednostavnije bi bilo da odmah odaberemo sve elemente `<li>` (na primer, pomoću izraza `jQuery('li')`).

Drugi koncept koji bi trebalo da imate na umu kada radite s metodama za kretanje jeste da se mnogim metodama može proslediti opcioni parametar koji se može upotrebiti za filtriranje selekcija. Pogledajmo ponovo primer sa ulančavanjem i razmislimo kako se može izmeniti tako da se izabere samo poslednji element `<li>`. I ovo je ilustrativni primer čija je svrha da se pokaže kako se metodi za kretanje može proslediti izraz za biranje konkretnog elementa:

```
jQuery('li:eq(1)').parent().children(':last') //bira poslednji element <li>
```

jQuery ima i druge metode za kretanje koje nisu ovde prikazane. Kompletan spisak svih metoda za kretanje i prateću dokumentaciju naći ćete na adresi *http://docs.jquery.com/Traversing*. Te dodatne metode koristiće se do kraja knjige.

1.10 Pravljenje DOM elemenata, rad s njima i umetanje

Problem

Treba da napravite nove DOM elemente (ili samo jedan element) koji se odmah biraju, obrađuju i potom umeću u DOM.

Rešenje

Ako još uvek niste uvideli, funkcija `jQuery()` je višestrana jer se ponaša različito zavisno od prosleđenih parametara. Ako funkciji prosledite znakovni niz ili sirovi HTML, napraviće usput te elemente za vas. Na primer, naredni iskaz pravi element `<a>` omotan unutar elementa `<p>` s tekstualnim čvorom kapsuliranim u elementima `<p>` i `<a>`:

```
jQuery('<p><a>jQuery</a></p>');
```

Nakon što je element napravljen, možete dalje raditi s njim pomoću jQuery metoda. To je kao da ste u startu izabrali `<p>` iz postojećeg HTML dokumenta. Na primer, mogli bismo da pomoću metode `.find()` izaberemo element `<a>` i da mu onda izmenimo jedan od atributa. U slučaju narednog iskaza, atributu `href` dodeljujemo vrednost `http://www.jquery.com`:

```
jQuery('<p><a>jQuery</a></p>').find('a').attr('href','http://www.jquery.com');
```

Sjajno, zar ne? Biće još bolje, jer smo do sada samo pravili elemente usput i radili s njima u kodu. Tek treba da, takoreći, zaista izmenimo trenutno učitan DOM. Da bismo to uradili, moraćemo da primenimo jQuery metode za manipulisanje. Naredni primer je naš segment koda u kontekstu HTML dokumenta. U njemu pravimo elemente, radimo s njima, a onda ih umećemo u DOM pomoću metode `appendTo()`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
jQuery('<p><a>jQuery</a></p>').find('a').attr('href','http://www.jquery.com')
  .end().appendTo('body');
</script>
</body>
</html>
```

Obratite pažnju na to da pomoću metode `end()` poništavam dejstvo metode `find()` tako da kada pozovem metodu `appendTo()` ona dodaje sadržaj izvornog omotačkog skupa.

Objašnjenje

U ovom receptu, funkciji `jQuery()` prosledili smo znakovni niz sirovog HTML koda pomoću kojeg su direktno napravljeni DOM elementi. Funkciji `jQuery()` se može proslediti i DOM objekat koji je napravila DOM metoda `createElement()`:

```
jQuery(document.createElement('p')).appendTo('body'); //dodaje prazan p element strani
```

Naravno, osigurati da znakovni niz HTML koda s više elemenata pravilno funkcioniše, mogao bi biti prilično zahtevan zadatak.

Valjalo bi pomenuti da je metoda `appendTo()` samo vrh ledenog brega brojnih metoda za manipulisanje. Pored metode `appendTo()`, na raspolaganju su i naredne metode za manipulisanje:

- `append()`
- `prepend()`
- `prependTo()`
- `after()`
- `before()`
- `insertAfter()`
- `insertBefore()`
- `wrap()`
- `wrapAll()`
- `wrapInner()`

1.11 Uklanjanje DOM elemenata

Problem

Treba da uklonite elemente iz DOM-a.

Rešenje

Za uklanjanje izabranog skupa elemenata i njihove dece iz DOM-a, može se koristiti metoda `remove()`. Razmotrimo naredni segment koda:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
```

```
<h3>Anchors</h3>
<a href='#'>Anchor Element</a>
<a href='#'>Anchor Element</a>
<a href='#'>Anchor Element</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
  jQuery('a').remove();
</script>
</body>
</html>
```

Kada se prethodni segment koda učita u veb čitač, sidreni elementi (engl. *anchor elements*) ostaće na stranici dok se ne izvrši JavaScript kôd. Kada se metoda `remove()` primeni da bi se uklonila sva sidra iz DOM-a, na strani će se videti samo element `<h3>`.

Ovoj metodi se može proslediti i izraz da bi se pri uklanjanju filtrirali elementi. Na primer, navedeni segment koda mogao bi se izmeniti tako da uklanja samo sidra određene klase:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<h3>Anchors</h3>
<a href='#' class='remove'>Anchor Element</a>
<a href='#'>Anchor Element</a>
<a href='#' class="remove">Anchor Element</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
  jQuery('a').remove('.remove');
</script>
</body>
</html>
```

Objašnjenje

Kada koristite jQuery metodu `remove()`, morate imati na umu sledeće:

- Iako ova metoda uklanja izabrane elemente iz DOM-a, oni nisu uklonjeni iz jQuery omotačkog skupa. To znači da biste, teorijski posmatrano, mogli da nastavite rad s njima i čak da ih vratite u DOM ako želite.
- Ova metoda ne samo da uklanja elemente iz DOM-a, već uklanja i sve procedure za obradu događaja i interno keširane (uskladištene) podatke koje su uklonjeni elementi možda sadržavali.

1.12 Zamena DOM elemenata

Problem

Tekuće čvorove u DOM-u treba da zamenite novim DOM čvorovima.

Rešenje

Elemente kojima treba zameniti postojeće elemente možemo izabrati pomoću metode `replaceWith()`. U narednom segmentu koda, pomoću metode `replaceWith()` zamenjujemo sve elemente `<li>` čiji atribut `class` ima vrednost `remove` novom DOM strukturom:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<ul>
<li class='remove'>name</li>
<li>name</li>
<li class='remove'>name</li>
<li class='remove'>name</li>
<li>name</li>
<li class='remove'>name</li>
<li>name</li>
<li class='remove'>name</li>
</ul>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
  jQuery('li.remove').replaceWith('<li>removed</li>');
</script>
</body>
</html>
```

Nova DOM struktura dodata DOM-u jeste parametar tipa znakovnog niza prosleđen metodi `replaceWith()`. U ovom primeru, svi elementi `<li>`, uključujući decu, zamenjuju se novom strukturom, `<li>removed</li>`.

Objašnjenje

jQuery metoda koja se ponaša obrnuto ovoj metodi, tj. koja izvršava isti zadatak ali sa obrnutim parametrima, jeste metoda `replaceAll()`. Na primer, jQuery segment koda iz našeg recepta mogao bi se napisati na sledeći način:

```
jQuery('<li>removed</li>').replaceAll('li.remove');
```

U ovom iskazu, funkciji `jQuery()` prosleđujemo HTML znakovni niz, a potom pomoću metode `replaceAll()` biramo DOM čvor i njegovu decu koje hoćemo da uklonimo i zamenimo.

1.13 Kloniranje DOM elemenata

Problem

Treba da klonirate/kopirate deo DOM-a.

Rešenje

jQuery metoda `clone()` služi za kopiranje DOM elemenata. Koristi se vrlo jednostavno – odaberu se DOM elementi pomoću funkcije `jQuery()`, a onda se nad izabranim skupom elemenata pozove metoda `clone()`. Umesto izvorno izabranih DOM elemenata, rezultat je kopija DOM strukture koja se vraća radi ulančavanja. U narednom segmentu koda kloniramo `<ul>`, a potom dodajemo kopiju DOM-u pomoću metode za umetanje `appendTo()`. U suštini, strani dodajemo još jednu `<ul>` strukturu istovetnu onoj koja tu već postoji:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<ul>
<li>list</li>
<li>list</li>
<li>list</li>
<li>list</li>
</ul>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
  jQuery('ul').clone().appendTo('body');
</script>
</body>
</html>
```

Objašnjenje

Metoda za kloniranje je vrlo korisna za premeštanje DOM odlomaka koda unutar DOM-a. Od naročite je pomoći kada želite da kopirate i premestite ne samo DOM elemente, već i događaje vezane za klonirane DOM elemente. Pažljivo pregledajte HTML kôd i jQuery kôd u ovom primeru:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<ul id="a">
```

```
<li>list</li>
<li>list</li>
<li>list</li>
<li>list</li>
</ul>
<ul id="b"></ul>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
 jQuery('ul#a li')
    .click(function(){alert('List Item Clicked')})
    .parent()
        .clone(true)
            .find('li')
            .appendTo('#b')
        .end()
    .end()
    .remove();
</script>
</body>
</html>
```

Ako biste izvršili ovaj segment koda u veb čitaču, klonirali biste na strani elemente `<li>` kojima je pridružen događaj click, umetnuli biste te nove klonirane elemente (uključujući događaje) u prazan element `<ul>` i potom uklonili element `<ul>` koji ste klonirali.

Možda je ovo previše za projektanta koji tek počinje da koristi jQuery, pa predlažem da pregledamo naredni jQuery iskaz redom da bismo razjasnili kako funkcionišu ulančane metode:

1. `jQuery('ul#a li')` = Bira element `<ul>` sa `id` atributom vrednosti `a`, potom bira sve elemente `<li>` unutar elementa `<ul>`.
2. `.click(function(){alert('List Item Clicked')})` = Pridružuje događaj click svakom elementu `<li>`.
3. `.parent()` = Kreće se po DOM-u menjajući moj izabrani skup u element `<ul>`.
4. `.clone(true)` = Klonira element `<ul>` i svu njegovu decu, uključujući svaki događaj pridružen elementima koji se kloniraju. To se radi tako što se metodi `clone()` prosleđuje logička vrednost `true`.
5. `.find('li')` = Sada se u okviru kloniranih elemenata menja skup elemenata tako da sadrži samo elemente `<li>` koji se nalaze unutar kloniranog elementa `<ul>`.
6. `.appendTo('#b')` = Izabrani klonirani elementi `<li>` smeštaju se unutar elementa `<ul>` čiji atribut `id` ima vrednost `b`.
7. `.end()` = Vraća prethodni izabran skup elemenata koji je bio klonirani element `<ul>`.
8. `.end()` = Vraća prethodno izabran skup elemenata – izvorni element `<ul>` koji je kloniran.
9. `.remove()` = Uklanja izvorni element `<ul>`.

Poznavanje načina manipulisanja izabranim skupom elemenata ili vraćanja na prethodno izabran skup, od presudnog je značaja za razumevanje složenih jQuery iskaza.

1.14 Čitanje vrednosti, zadavanje vrednosti i uklanjanje atributa DOM elementa

Problem

Treba da saznate ili da zadate vrednost atributa DOM elementa izabranog pomoću funkcije `jQuery()`.

Rešenje

jQuery metoda `attr()` služi za čitanje i zadavanje vrednosti atributa. U narednom segmentu koda zadavaćemo, a potom i čitati vrednost atributa `href` elementa `<a>`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<a>jquery.com</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js">
</script>
<script type="text/JavaScript">
// obaveštava o URL adresi jQuery matične strane
alert(
    jQuery('a').attr('href','http://www.jquery.com').attr('href')
);
</script>
</body>
</html>
```

U prethodnom primeru, biramo element `<a>` u HTML dokumentu, zadajemo vrednost njegovog atributa `href`, a potom čitamo vrednost tog atributa tako što istoj toj metodi prosleđujemo njegovo ime. Da je u dokumentu bilo više elemenata `<a>`, metoda `attr()` bi pristupila prvom odgovarajućem elementu. Kada se učita u veb čitač, ovaj segment koda će pomoću metode `alert()` ispisati zadatu vrednost atributa `href`.

Pošto većina elemenata ima više atributa, moguće je zadati vrednost većem broju atributa jednim pozivom metode `attr()`. Recimo, u prethodnom primeru mogli bismo da zadamo vrednosti i atributu `title` tako što bismo metodi `attr()` prosledili objekat umesto dva parametra tipa znakovnog niza:

```
jQuery('a').attr({'href':'http://www.jquery.com','title':'jquery.com'}).attr('href')
```

Mogućnost dodavanja atributa elementima praćena je mogućnošću uklanjanja atributa i njihovih vrednosti. Za uklanjanje atributa HTML elemenata koristi se metoda `removeAttr()`. Ovoj metodi se prosleđuje znakovni niz atributa koji želite da uklonite (npr. `jQuery('a').removeAttr('title')`).

Objašnjenje

Pored metode `attr()`, jQuery ima i poseban skup metoda za rad sa atributom `class` HTML elementa. Pošto atribut `class` može da ima više vrednosti (na primer, `class="class1 class2 class3"`), ove jedinstvene metode za atribute koriste se za rad s tim vrednostima.

Reč je o narednim jQuery metodama:

`addClass()`
: Ažurira vrednost atributa `class` pomoću nove klase/vrednosti, uključujući svaku klasu s već zadatom vrednošću

`hasClass()`
: Čita vrednost atributa `class` određene klase

`removeClass()`
: Uklanja jedinstvenu klasu iz atributa `class` zadržavajući svaku prethodno zadatu vrednost

`toggleClass()`
: Dodaje određenu klasu ako već ne postoji; uklanja određenu klasu ukoliko postoji

1.15 Čitanje i zadavanje HTML sadržaja

Problem

Potrebno je pročitati fragment HTML sadržaja s tekuće veb strane ili ga zadati.

Rešenje

Za čitanje i zadavanje fragmenata (ili DOM struktura) HTML elemenata može se koristiti jQuery metoda `html()`. U narednom segmentu koda, pomoću ove metode zadajemo, a potom čitamo HTML vrednost elementa `<p>` iz HTML dokumenta:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p></p>
<script type="text/JavaScript"
```

```
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js">
</script>
<script type="text/JavaScript">
jQuery('p').html('<strong>Hello World</strong>, I am a <em>&lt;p&gt;</em> element.');
alert(jQuery('p').html());
</script>
</body>
</html>
```

Kada se ovaj segment koda izvrši u veb čitaču, čitač će ispisati šta je HTML sadržaj elementa `<p>` koji smo zadali, a potom i učitali pomoću metode `html()`.

Objašnjenje

Ova metoda koristi DOM svojstvo `innerHTML` da bi čitala i zadavala fragmente HTML sadržaja. Imajte na umu da metoda `html()` nije dostupna za XML dokumente (premda će funkcionisati za XHTML dokumente).

1.16 Čitanje i zadavanje tekstualnog sadržaja

Problem

Potrebno je pročitati ili zadati tekst unutar HTML elementa.

Rešenje

Za čitanje i zadavanje tekstualnog sadržaja elemenata služi jQuery metoda `text()`. U narednom segmentu koda, koristimo ovu metodu da zadamo, a potom i pročitamo tekstualnu vrednost elementa `<p>` u HTML dokumentu:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p></p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js">
</script>
<script type="text/JavaScript">
    jQuery('p').text('Hello World, I am a <p> element.');
    alert(jQuery('p').text());
</script>
</body>
</html>
```

Kada se ovaj segment koda izvrši u veb čitaču, čitač će ispisati šta je sadržaj elementa `<p>` koji smo zadali, a potom i učitali pomoću metode `html()`.

Objašnjenje

Metoda `text()` slična je metodi `html()` s tim što metoda `text()` ne uzima u obzir HTML označavanje (menja simbole `<` i `>` njihovim HTML verzijama). To znači sledeće: ako stavite oznake u znakovni niz prosleđen metodi `text()`, ona će ih konvertovati u njihove HTML verzije (`<` i `>`).

1.17 Korišćenje alijasa $ bez uzrokovanja globalnih konflikata

Problem

Želite da umesto navođenja imena globalnog imenskog prostora (jQuery) koristite skraćenu verziju, alijas `$`, ne rizikujući da nastane globalni konflikt imena.

Rešenje

Napravite anonimnu samopozivajuću funkciju kojoj se prosleđuje jQuery objekat, a potom upotrebite znak `$` kao parametar pokazivač na jQuery objekat.

Na primer, sav jQuery kôd mogao bi se kapsulirati unutar naredne samopozivajuće funkcije:

```
(function($){ //funkcija koja pravi privatnu oblast važenja, s parametrom $
    //privatna oblast važenja i korišćenje znaka $ bez rizika od konflikta
})(jQuery); //poziva bezimenu funkciju i prosleđuje joj jQuery objekat
```

Objašnjenje

Prosledili smo globalnu referencu na jQuery kôd funkciji koja pravi privatnu oblast važenja. Da nismo tako uradili i da smo alijas `$` primenili u globalnoj oblasti važenja, prenebregli bismo rizik od konflikta imena pretpostavljajući da nijedan drugi postojeći (ili budući) skript u HTML dokumentu ne koristi znak `$`. Zašto rizikovati kada možete napraviti privatnu oblast važenja?

Druga prednost ovakvog postupka jeste to što će se kôd unutar anonimne samopozivajuće funkcije izvršavati u sopstvenoj privatnoj oblasti važenja. Budite sigurni da je malo verovatno kako će ono što se nalazi unutar funkcije ikada izazvati konflikt s drugim JavaScript kodom napisanim u globalnoj oblasti važenja. Dakle, zašto rizikovati programske sukobe? Naprosto, napravite sopstvenu privatnu oblast važenja.