Vezba 10 –Napredni jQuery i Ajax

Cilj vežbe:

- Upoznavanje sa naprednim mogućnostima jQuery biblioteke
- Manipulacija DOM strukturom pomoću jQuery metoda
- Korišćenje Javascript struktura (nizova i objekata) za definisanje podataka koji se koriste na web strani
- Razumevanje načina funkcionisanja ajax poziva
- Korišćenje data atributa za skladištenje podataka na klijentskoj strani

1. Obilazak DOM stable sa jQuery-jem

Ako koristite JavaScript za rad sa DOM elementima *native* funkcije kao "getElementById" i "childNodes" su jako korisne ali često ne dovoljno fleksibilne i ekspresivne.

Fundamentalna struktura DOM elemenata je stablo, gde svaki element ima svog roditelja, dete, ili decu i rođake. Element takođe može biti sledeći ili prethodni u odnosu na trenutni element koji obrađujemo. U ovom primeru ćemo prikazati neke od jQuery metoda koje nam omogućuju da se nesmetano krećemo po ovom DOM stablu.

Prve DOM metode koje ćemo prikazati se koriste za pristup roditeljima ili deci trenutno selektovanog čvora. Roditelj trenutnog čvora možemo selektovati pomoću **parent()** jQuery funkcije pri čemu možemo proslediti selektor kao parametar za filtriranje. Na primer $('p').parent('div') će selektovati direktnog roditelja paragraph elementa za koji se očekuje da je div.

Ukoliko želite da pristupite nekom čvoru-roditelju koji je se nalazi više u DOM hijerarhiji (npr. deda ili pradeda ovog čvora) možemo koristiti **parents()**. Primetite razliku između ova dva metoda, u prvom slučaju se radi o jednini parent(), a u drugom o množini parents().

Deci bilo kog elementa ili čvora možemo pristupiti koristeći metod **children()**, kome takođe možemo proslediti neki selektor kao parametar. Ukoliko želimo da pretražimo više nivoa u dubinu DOM stabla možemo koristiti metodu **find()**.

Primer 1: Korišćenje metode parents()

```
$(document).ready(function(){
   $('#btn1')
   .parent()
   .hide();
});
```

Primer 2: Korišćenje metode children()

```
$(document).ready(function(){
   $('.klasa')
   .children()
   .html("novi tekst za treci h3!");
});
```

2. Metode za manipulaciju DOM stablom

Primer 3: Dodavanje i uklanjanje klasa

```
<script>
   function toggle() {
     var els = $("#clickme");
     if (!els.hasClass('nice'))
      els.addClass('nice');
     else
      els.removeClass('nice');
   }

   $("#clickme").click(toggle);
</script>
```

3. Manipulacija atributa i svojstva

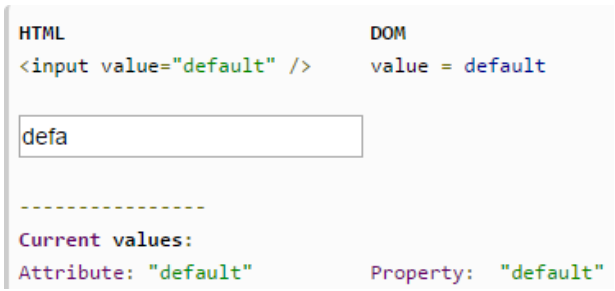
Razlika između HTML atributa i DOM svojstva

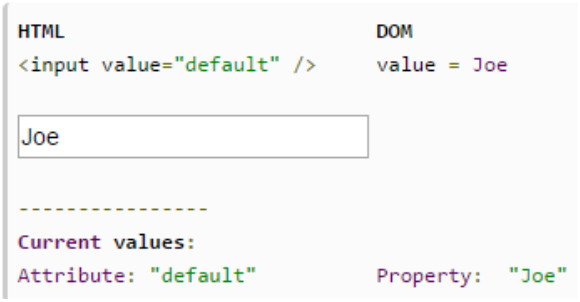
Atributi nose dodatne informacije o html elementima i zapisuju se u name=”value” parovima. Na primer . Ovde je dat element div koji ima atribut class čija je vrednost “klasa”.

Svojstva (*property)* predstavljaju reprezentaciju atributa u DOM strukturi. U ovoj strukturi div element iz prethodnog primera bi izgledao otprilike ovako:

- nodeName = “DIV”
- className = “klasa”
- style ….

Razlika je u tome što se atributi nalaze u HTML fajlu dok se svojstva nalaze u DOM stablu. Ovo znači da se atributi ne menjaju i uvek nose inicijalnu predefinisanu vrednost dok se svojstva mogu menjati. Na primer kada korisnik klikne na checkbox, unese neki tekst u edit box ili iskoristi JavaScript kod da promeni neku vrednost svojstva. Ilustrujmo to sledećim primerom:





Primetimo da na prvoj slici atribut i svojstvo imaju iste vrednosti (“default”) dok na drugoj slici, nakon ažuriranja edit polja, vrednost svojstva se promenila (“Joe”) dok je vrednost atributa ostala ista (“default”). U zavisnosti od toga da li želite da manipulišete atributima, svojstvima ili CSS svojstvima jQuery Vam pruža na korišćenje metode **attr(), prop() i css().** Ilustrujmo kako one funkcionišu na sledećem primeru:

Primer 4: Modifikacija običnih i CSS atributa

```
$(document).ready(function(){
   $( ".box" ).click(function() {
    var html = [ "The clicked div has the following styles:" ];
    var styleProps = $(this).css(["width", "height","color","background-color"]);
     $.each( styleProps, function( prop, value ) {
         html.push( prop + ": " + value );
     });
     $("#result").html( html.join( "<br>"));
   });

   $("#btn1").click(function() {  //change CSS properties at once
```

```
      $(".box").css({
            'background-color': 'lightblue',
            'color': 'pink',
            'height': '100px'
         });

         $( "#greatphoto" ).attr({
            alt: "Beijing Brush Seller",
            title: "photo by Kelly Clark"
         });
         $( "#input").prop( "value", "Milos");
         $( "#btn1" ).prop( "disabled", true );
      });
   });
```

Manipulacija HTML čvorovima

Postoji više načina da promenite postojeći HTML element. Najčešće se menja samo unutrašnji HTML ili atribut nekog elementa. Za tu namenu jQuery nudi više *cross-browser* metoda. Ukoliko se pozovu bez parametra one čitaju i vraćaju traženu vrednost. Ukoliko se pozovu sa parametrom one menjaju odnosno upisuju vrednost:

- .html() – čita ili upisuje HTML sadržaj
- .text() – čita ili upisuje tekst, html kod će biti uklonjen
- .width() – čita ili upisuje širinu u pikselima prvog elementa iz selekcije
- .height() – čita ili upisuje visinu prvog elementa iz selekcije
- .position() – vraća objekat koji sadrži položaj elementa u odnosu na njegovog roditelja (ne može da promeni poziciju elementa već samo da je pročita)
- .val() – šita ili upisuje vrednost elemenata neke forme (npr editbox itd.)

Ukoliko želite da kreirate nove ili uklanjate već postojeće čvorove možete koristiti sledeće funkcije:

- .insertAfter() i .after()
- .insertBefore() i .before()
- .appendTo() i .append()
- .prepandTo() i .prepend()
- .clone
- .remove() i detach()

Primer 5: Dodavanje i brisanje HTML čvorova

```
$(document).ready(function(){
   $("#btn1").click(function(){
      $("p").append(" <b>Appended text</b>.");
      });
   $("#btn2").click(function(){
      $("ol").append("<li>Appended item</li>");
      });
   });
```

4. Funkcije opšte namene i *Namespace*

Iako je obavijanje elemenata jedan od najčešće korišćenih upotreba $() funkcije, to nije jedina njena upotreba. Jedna od njenih dodatnih upotreba je da služi kao *namespace* prefiks za gomilu funkcija opšte namene. Ove funkcije nam pomažu u radu sa stringovima, nizovima (recimo funkcija `$.each` nam

omogućava obilazak niza elemenata ili svih atributa nekog objekta), ili u pakovanju i parsiranju XML ili JSON podataka. Navedimo par najbitnijih funkcija koje možemo koristili:

Metod $.parseJSON

Ovaj metod nam omogućava da parsiramo podatke koje dobijemo od servera koji su u string formatu, i ukoliko je taj string pravilno napisan, kao rezultat ćemo dobiti JavaScript objekat. Kako ovaj metod funkcioniše će biti ilustrovano primerom u odeljku koji se bavi ajax metodama i hvatanjem JSON podataka.

Metod $.data

jQuery data metod nam daje mogućnost da povežemo, odnosno pridružimo, proizvoljne podatke bilo kom DOM čvoru ili JavaScript objektu. To čini naš kod konciznijim i čistijim.

Ovaj metod možete pozvati kao jQuery objekat, ili možete koristiti `$.data()` funkciju umesto toga na sledeći način:

```
// Korišćenje data metoda:
$("#myDiv").data("key","neki_string");

// Korišćenje data funkcije:
$.data($("#myDiv").get(0),"key","neki_string");
```

Primetite da je potrebno da prosledimo DOM element kao prvi parametar funkciji `$.data()`, a ne jQuery objekat. To može biti nezgodno pa ćemo preferirati korišćenje `data` funkcije kao poziva metode.

Kada koristite `data` kao metod, potrebno je da prosledite dva parametra - **ključ** i **vrednost** koju treba zapamtiti. Ključ mora da bude string, a vrednost može biti bilo koja struktura podataka, uključujući funkcije, nizove i objekte. Postoji i alternativna sintaksa, u kojoj prosleđujete samo objekat kao parametar:

Primer 6: Korišćenje data atributa

```
$("#myDiv").data({"name":"Dejan","age":21});
```

Sada kada smo uneli i zapamtili podatke, možemo ih pročitati pozivom metode sa samo jednim parametrom – ključem.

```
var theValue = $("#myDiv").data("age"); // 21
alert(theValue);
```

Od verzije 1.4.3 *data* metod može pristupiti i *data atributima*. To znači da ako imamo dat ovakav element, možemo pristupiti `data` internal-id atributu tako što ćemo pozvati:

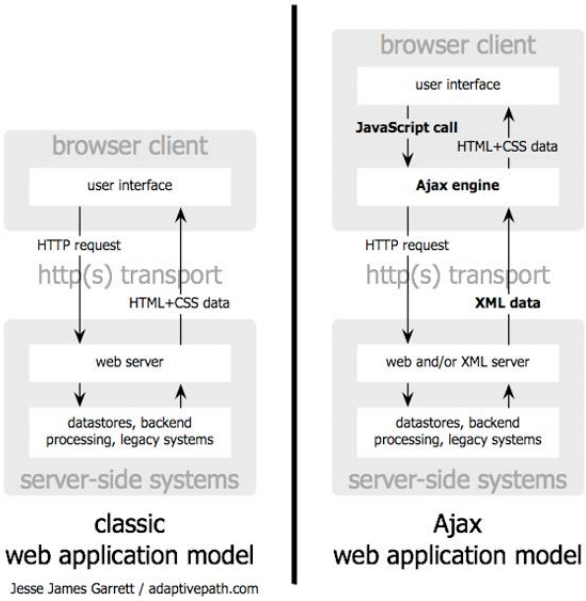
```
<img id="img1" data-internal-id="221" width="100" height="100" />
$("#img1").data('internal-id');
```

što naročito dobija na značaju u veb stranama koje koriste HTML5 i *data atribute*, ali dalja analiza HTML5 standarda prevazilazi opseg ovog rada.

5. jQuery i Ajax

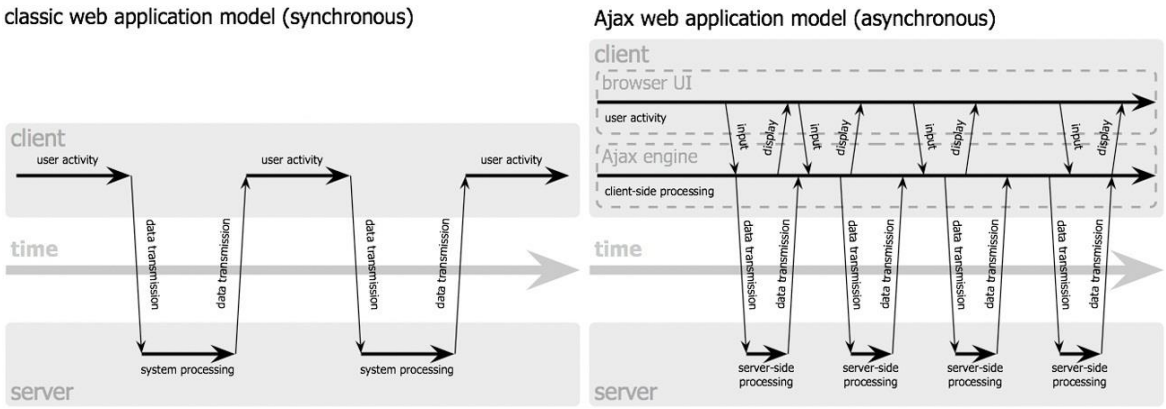
Neki tvrde da pod Ajax tehnologijom podrazumevamo bilo koji metod koji koji omogućava da uputimo zahtev serveru bez ponovnog učitavanja veb strane (tako što ćemo, na primer, *submit*-ovati zahtev skrivenom iframe elementu). Ipak većina ljudi smatra da se termin odnosi na korišćenje *XMLHttpRequest* (*XHR*) objekta ili Microsoft *XMLHTTP ActiveX* kontrole. Hajde da vidimo kako izgleda model Ajax aplikacije u odnosu na normalnu veb aplikaciju.

Klasični model rada veb aplikacije i AJAX model razlikuju se u nekoliko segmenata. Primarno se to odnosi na način preuzimanja i mesto obrade podataka, kao što se i vidi na slici 2.



Slika 1 - Model Ajax i Web aplikacije

Vrlo je značajno postići osetljivost aplikacije na korisničku interakciju koja je po svojoj prirodi asinhrona (korisnik ne voli da čeka da kompjuter završi sve što ima, već samo ono što mora). Kod tradicionalne veb aplikacije čitač preuzima veb stranicu tako što pošalje zahtev serveru i zatim, nakon što primi odgovor, stranicu prikaže u svom prozoru. Ovo uzrokuje kreni-stani način rada, tačnije uzrokuje pauze pri preuzimanju podataka u kojima korisnik pred sobom ima prazan prozor čitača. Ajax uvodi asinhronost u preuzimanju podataka, dodavanjem novog sloja u aplikaciju, kao što je prikazano na slici 1. Klijent šalje poziv Ajax sloju i nastavlja normalno sa radom. Ajax komunicira sa serverom i nakon dobijenog odgovora prosleđuje odgovor klijentu koji sve vreme nesmetano radi. Ovaj mehanizam je odgovoran kako za iscrtavanje sadržaja korisniku, tako i za komunikaciju sa serverom pri zahtevanju izmena ili preuzimanju novih podataka.



Slika 2- Model komunikacije Ajax i Web aplikacije

Kao što smo napomenuli na početku, implementacija Ajax mehanizma se razlikuje od čitača do čitača (često i u zavisnosti od verzije čitača). Još jednom jQuery spašava stvar i nudi nam interfejs za korišćenje Ajax mehanizma bez potrebe da brinemo o vrsti ili verziji čitača, a ujedno nam skraćuje kod i vreme koje je potrebno za kreiranje XMLHttpRequest objekta i njegovo prosleđivanje serveru (za šta bi nam u čistom JavaScript kodu trebalo 20-tak linija, a u jQuery-ju nam je dovoljna samo jedna).

Glavna metoda jQuery-ja za sve Ajax zahteve je **$.ajax()**. Ona je osnova za sve zahteve koje čitač upućuje serveru, kao i odgovore koje dobija od servera. Da biste inicirali zahtev serveru morate proslediti ovoj metodi

određene parametre. Postoji ogroman broj parametara koji se može proslediti i on se prosleđuje kao jedan options objekat. Navešćemo one koji se najčešće koriste: type, url, dataType, error, success. [2]

- **type** - Prvi parametar koji treba proslediti kada šaljete Ajax poziv je tip http zahteva, koji će u velikoj većini slučajeva biti GET ili POST,
- **url** - Parametar je naravno adresa servera od kog zahtevamo podatke i bez koga naš zahtev ne bi imao nikakvog smisla,
- **dataType** – Tip podatka koje će server vratiti. Najčešće text, XML, JSON. JSONP, script ili HTML,
- **error** - Ovo je callback metoda ili funkcija, o kojima je bilo reči u poglavlju 3.1.4. Ova metoda će se pozvati i izvršiti ukoliko dođe do greške u toku obrade našeg Ajax zahteva,
- **success** – slično error metodi, ova metoda će se izvršiti ukoliko je zahtev uspešan i server kao odgovor vrati kod 200.

Pre nego što prikažemo par primera napomenimo da na vašem računaru morate imati instaliran server i neki serverski jezik (u našem slučaju PHP) kako biste mogli lokalno da pokrenete primere. Primere morate pokrenuti iz čitača preko url adrese kako bi se PHP kod izvršio. Pokažimo sada kako bi ajax poziv izgledao sledećim primerom:

Primer 7: Primer ajax poziva

```
$.ajax({
            url: "http://localhost/test/test1.php",
            dataType: 'html',
            type: 'GET',
            error: function(xhr, textStatus){
               alert("error");
            },
            success: function(data) {
               alert(data);
               $( "#myDiv" ).append( data );
               }
            });
```

Neki od dodatnih parametara koji se mogu postaviti su:

- **Async** – ako je postavljen na `false` serveru će se poslati sinhroni zahtev, odnosno aplikacija će čekati na odgovor servera pre nego što nastavi sa izvršavanjem,
- **Cashe** – Parametar koji omogućava keširanje na stani čitača. Ukoliko je setovan na `false` jQuery će nadovezati GET promenljivu na GET string koja će biti neki random broj (npr. `/server-ajax-gateway?_=6273551235126`), koja će sprečiti čitač, proxy i server da vrati odgovor zapamćen u kešu,
- **Complete** – callback metoda koja će se izvršiti i u slučaju da dođe do greške, i u slučaju da je zahtev uspešan,
- **Data** – podaci koje treba poslati serveru.

Primer: Pozivanje sinhronog Ajax poziva sa dodatnim parametrima i smeštanje odgovora u promenljivu `json`:

```
json = $.ajax({
       url: proxy.url + 'getEvents&' +     'ids='       +  ids    +
           '&start='    +  start        +  '&end='        +  end,
       async: false,
       dataType: "json"
     }).responseText;
```

Ukoliko nemate potrebu da koristite sve napredne parametre Ajax metoda, na raspolaganju su vam i brojne skraćene metode Ajax poziva koje ćemo ilustrovati kroz primere:

Pozivanje GET zahteva sa prosleđivanjem dodatnih parametara:

```
$.get(
         "http://localhost/test/test2.php?q=0",
         function(data) {
            alert(data);
            $( "#tekst" ).append( data );
         });
      $("#btn4").data("x", 1);
   });
```

Pozivanje POST zahteva sa prosleđivanjem tekstualnih parametra calendar i c:

```
$.post(' .php?c=updateCalendar', {"calendar": "Calendar");
```

6. jQuery i JSON

Primer 8: Pozivanje **`getJSON`** metoda koji će vraćene JSON podatke parsirati direktno u JavaScript objekat. Poziv u primeru šalje podatke o nekom *event-u* pomoću promenljive `send_data`, server obrađuje poslate podatke, kreira novi *event* u bazi i vraća podatke o *event*-u u JSON formatu koji se parsiraju u JavaScript objekat `received_data`. Na kraju koristimo metodu `fullCalendar` da dalje obrađujemo i prikažemo *event* na Web stranici:

```
$.getJSON("http://localhost/test/test2.php?q=-1",
         function(o){
            var pod = "<tr>" +
               "<td>" + o.ime + "</td>" +
               "<td>" + o.prezime + "</td>" +
               "<td>" + o.broj_godina + "</td>" +
               "<td></td>" +
               "<td></td>" +
            "</tr>";
            $( "#tekst" ).children().append( pod );
      });
```

Metod load se poziva na grupi selektovanih jQuery objekata. On preuzima HTML sa navedenog URLa i pokušava da ga doda u selektovane elemente direktno što predstavlja kratak i efikasan način za direktno ažuriranje Web sadržaja:

```
$('#newContent').load('/foo.html');
```

Ajax pozivi se mogu koristiti za različite stvari. Najčešća upotreba je pribavljanje HTML koda sa servera i umetanje u DOM strukturu. Mogu se koristiti i za razmenu podataka između servera i klijenta.

7. Cross-Origin XMLHttpRequest

8. jQuery i XML

9. Zadaci za samostalan rad studenta

Zadatak 1: Izmeniti primer 1 tako da samo h3 elementi promene tekst (ne svi elementi);

Zadatak 2: U zadatku 6 dodati još jedno dugme koje će obrisati prvu stavku u listi. Zatim dodati i četvrto dugme koje će obrisati tekst dodat pritiskom na prvo dugme (Append text). Napisati kod koji to treba da uradi:

Zadatak 3: Šta je JSONLint?

Zadatak 4: Kreirati 3 dugmeta, a zatim kreirati četvrto dugme. Klik na četvrto dugme treba da na ekranu ispiše koje je zadnje dugme kliknuto (dugme 1, 2, 3 ili 4). Zadatak rešiti korišćenjem data atributa.

Zadatak 5: Izmeniti kod u primeru 8 tako da se u html kod doda select dugme kojim ce korisnik moći da izabere broj studenta (1,2,3,4 ili 5). Klikom na treće dugme (Dodaj red sa servera“) proslediće se serveru trenutni izbor i server ce vratiti podatke o studentu sa tim rednim brojem.
Trenutna implementacija dodaje podatke o studentu u tabelu kao i u div iznad tabele. Izmeniti kod tako da se podaci o studentima dodaju samo u tabeli.

Zadatak 6: Izmeniti kod u primeru 8 tako da klik na dugme „Dodaj podatke u data“ otvori dijalog (ili samo dodati formu ispod svih html elemenata) u kome je potrebno uneti podatke o studentu (Ime, prezime, broj godina, grad i zanimanje). Unete podatke treba zapamtiti u data atributu nekog html elementa i zatim klikom na dugme „Dodati red iz data“ dodati ga u listu studenata.

Firstname	Lastname	Age	Hometown	Job
Tanja	Tanjic	23	Svrljig	Web Dizajner
Seka	Aleksic	34	Pukovac	Starleta
Mitar	Mitric	45		
Mitar	Mitric	45		

Ime: iiiiiiiiiiiii
Prezime: fsdf
Broj Godina: sfd
Grad: fsd
Zaimanje: fsd

Zadatak 7: Izmeniti kod u primeru 8 koji će sa url adrese http://localhost/test/test3.php učitati podatke o instituciji u kojoj radite. Prepoznajte koje vam podatke server šalje, u kom formatu, i prikažite ih u headeru web strane.

VisokaTehnickaSkolaNis

00381641409020

Aleksandra Medvedeva bb

12-12-2014

Ucitaj Header | Dodaj tabelu | Dodaj podatke u data | Dodaj red iz data | Dodaj red sa servera | Dodaj JSON podatke sa Servea

Zadatak 8:

Zadatak 9: Napisati ajax poziv korišćenjem jQuery metode .ajax koja će biti ekvivalentna sledećim primerima:

a. Primer getJSON zahteva

```
$.getJSON("http://localhost/test/test2.php?q=-1",
        function(o){ console.log ("JSON data returned", o); });
```

b. Primer get zahteva

```
$.get(   "http://localhost/test/test2.php?q=" + tmp,
        function(data) { $( "#tekst" ).children().append( data ); }, "script");
```

Zadatak 10: Pokrenite primer 8 tako što ćete ga naći na lokalnom hard disku, kliknuti desnim dugmetom miša i selektovati opciju open with, a zatim izaberite chrome ili neki drugi veb čitač. Zatim otvorite isti fajl tako što ćete u čitaču otići na localhost adresu i tako otvoriti fajl. Šta se desilo? U čemu je razlika između ova dva načina otvaranja istog fajla?

Student Overava

_____ _____