jQuery

Osnove

Uvod

- JQuery je JS biblioteka. Mala, jednostavna za upotrebu, istovremeno brza pri radu. jQuery olakšava kreiranje JS koda:
  - rukovanje događajima,
  - kreiranje animacija,
  - **asinhroni rad**.
- jQuery biblioteka:
  - uživa podršku velikog broja programera okupljenih u jQuery zajednicu (blogovi, uputstva, knjige)
  - OpenSource – dostupan izvorni kod potpuno besplatno
  - Podržan od strane velikih kompanija: Google, Microsoft, IBM, Amazon…
  - Lako je proširiv

Uključivanje jQuery biblioteke

- Kako bi se koristila jQuery biblioteka mora se uključiti na mestu gde se referiše spoljašnji JavaScript kod u projektu.
- `<script src="putanja"> </script>`

- To se radi na dva načina:
- 1. Tako što se iskoristi biblioteka koja je već hostovana negde na Internetu – jQurey CDN (engl. Content Delivery Network).
  `putanja =`
- `https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js // Google`
- `https://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.3.1.min.js // Microsoft`

- 2. Preuzeti verziju biblioteke sa sajta www.jQuery.com i postaviti je na web serveru ili lokalnom fajl sistemu. A onda, slično kao u prethodnom slučaju, navesti putanju do fajla gde je skinuti fajl
  `putanja =`
- `./jQuery/jquery-3.3.1.min.js // Lokalno u folderu jQuery`

Minimalna verzija

- Umanjena jQuery biblioteka znatno otežava eventualno eventualnu izmenu ili ispravku koda. Zato je preporuka da se u toku razvoja koristi neumanjenu verzija svih skripti, a da se pri postavljanju na sajt tj. u produkciju koriste minimalne veličine fajlova.

Početak - jQuery()

- Oznaka **$()**, predstavlja alias za **jQuery()** funkciju. Ova funkcija vraća objekat kojim se mogu prihvatiti DOM elementi HTML dokumenta, po redosledu po kome su definisani u dokumentu a koji odgovaraju prosleđenom selektoru.
- Vraćeni objekat ove funkcije poseduje veliki broj korisnih unapred definisanih metoda. Na primer jQuery() funkcija se može koristiti:
  - Za selekciju i obavijanje DOM elemenata nad kojima treba izvršiti neku operaciju;
  - Za korišćenje globalnih funkcija;
  - Za kreiranje novih DOM elemenata u HTML jeziku;
  - Asinhroni rad sa udaljenim servisima.

Primeri selektovanja – adekvatni CSS selektorima

- `$(this).hide()` `// skriva tekući element.`
- `$("p").hide()` `// skriva svi <p> elemente - getElementByTagName.`
- `$(".test").hide()` `// elementi klase class="test" - getElementByClassName.`
- `$("#test").hide()` `// element čiji je id="test" - getElementById.`
- `$("[name='lozinka']").hide()` `// element naziva name="test" - getElementByName.`

Učitana stranica kao uslov

- Pri korišćenju JS funkcija u radu sa elementima dokumenta važno je da taj rad usledi pošto su učitani svi DOM elementi stranice u potpunosti, jer bi u suprotnom moglo da se primeni operacija nad elementom koji još nije učitan.
- Može se koristiti događaj **onLoad** kako bi omogućio izvršavanje tek nakon što se stranica u potpunosti učita. Sintaksa naredbe je obično sledeća:

```
$(fja)
$(document).ready(fja)
$("document").ready(fja)
$("img").ready(fja)
$().ready(fja)
```

```
$(document)
.ready(function () {
    // metode ...
});
```

```
// skraceno ...
$(function () {
// metode ...
});
```

Primer 1

```
// 1....
window.onload = promenaBoje;
function promenaBoje() {
    $("h2").css("color", "red");
}

// 2....
$(promenaBoje);
function promenaBoje() {
    $("h2").css("color", "red");
}

// 3....
$(function () {
    $("h2").css("color", "red");
});
```

Primeri selektora

- `$("*")` - svi elementi
- `$(this)` - trenutni HTML element
- `$("#divID")` - element za `id = "divID"`
- `$("p.naglasen")` - `<p>` elementi koji imaju klasu `"naglasen"`
- `$("p:first")` - prvi `<p>` element
- `$("ul li:first-child")` - prvi `<li>` element svakog `<ul>` elementa
- `$("[href]")` - svi elementi sa `href` atributom
- `$("a[href!='_xyz']")` - svi `<a>` elementi kod kojih je atribut href različit od `"_ xyz"`
- `$(":button")` - svi `<button>` elementi i `<input>` elemente koji imaju `type="button"`
- `$("tr:even")` - svi parni `<tr>` elementi

Ulančavanje

- Karakteristika jQuery metoda jeste da povratna vrednost vraća isti vrstu objekata kao objekti nad kojim se funkcija izvršavala. Na taj način moguće nadovezati drugu funkciju koja treba da se izvrši nad istom grupom elemenata.
- Na primer, ukoliko želimo da dodamo novu klasu `removed` svakom od elemenata, pored toga što ćemo ih sakriti, napisali bismo sledeće:

Stilovi:

```
.stl1{
    color: red;
}
.stl2 {
    color: gray;
}
```

HTML: `<h3 class="stl1">Naslov</h3>`

JS:

```
$(function () {
    $("h3.stl1")
        .addClass("stl2")
        .removeClass("stl1");
});
```

Konteksni upiti / drugi argument

- Ukoliko se pri pisanju upita, navede drugi argument koji je jQuery objekat, pretraga se vrši nad sadržajem tog objekta, a ne nad celim dokumentom. Na primer:
- `$div1 = $("div.vazno"); // div elementi sa klasom .vazno`
- `$ul1 = $(".smerovi", $div1); // ul u nadjenom div1 objektu`
- `$li = $("li", $ul1); // li stavke u ul1 objektu`

Događaji

- jQuery poseduje predefinisane metode za određene događaje. Na primer:

Događaji miša	Događaji tastature	Događaji forme	Document/Window događaji
`click`	`keypress`	`submit`	`load`
`dblclick`	`keydown`	`change`	`resize`
`hover`	`keyup`	`focus`	`scroll`
`mouseenter`, `mouseleave`, `mouseout`		`blur`	`unload`

Primer 2:

```
// 1. Pomocu metode dogadjaja
$("li").dblclick(function () {
    $(this).hide();
});
// hover() metod ima 2 funkcije i kombinacija je mouseenter() i mouseleave() metoda.
// Prva funkcija se izvršava kada se miš naše iznad elementa a druga kada miš napušta element.
$("h3").hover(
     function () {
         alert("Nalazite se iznad h3 elementa!");
     },
     function () {
         alert("Napuštanje elementa");
     }
);
```

```
// 2. Pomocu metode on()
// Metoda on() dodaje jednu ili više funkcija događaja za selektovane elemente.
// Dodajemo clik događaja za svaki < li > element:
$("li").on({
    mouseenter: function () {
        $(this).css("background-color", "lightgreen");
    },
    mouseleave: function () {
        $(this).css("background-color", "lightyellow");
    },
    click: function () {
        $(this).css("background-color", "orange");
    }
});
```

Uključivanje događaja

- Ako se poziva ista funkcija za više događaja, kod se još pojednostavljuje:

```
var brojac = 0;
$("#div1").on("mouseenter mouseleave click", function () {
    $("#div2").text(brojac++);
});
```

Isključivanje događaja

- Kao što se događaji mogu dodati također je moguće i isključiti neke već dodate događaje:
  - Isključivanje metode poruka vezane za događaj mouseenter
  - `$("#div1").off({"mouseenter", poruka });`
  - Isključivanje svih metoda za događaj mouseenter
  - `$("#div1").off({"mouseenter"});`

Sadržaj elementa

- Tri metode za manipulaciju sadržajem DOM elemenata su:
  - `text()` – postavlja ili vraća tekstualni sadržaj
  - `html()` - postavlja ili vraća sadržaj uključujući HTML markup
  - `val()` - postavlja ili vraća vrednost form polja.

```
<p>Ime: <input type="text" id="ime" value="Petar P"></p>
<p id="testParagraf">ovo je <i>iskoseno</i> u ovom
paragrafu.</p>

<button id="btn0">cita val</button>
<button id="btn1">cita Text</button>
<button id="btn2">cita HTML</button> <br/>
<button id="btn3">postavlja val</button>
<button id="btn4">postavlja Text</button>
<button id="btn5">postavlja HTML</button>
```

```
$("#btn0").click(function () {
    alert("Value: " + $("#ime").val());
});
$("#btn1").click(function () {
    alert("Text: " + $("#testParagraf").text());
});
$("#btn2").click(function () {
    alert("HTML: " + $("#testParagraf").html());
});
```

```
$("#btn3").click(function () {
    $("#ime").val("Unesi ime");
});
$("#btn4").click(function () {
    $("#testParagraf").text("tekstualni sadrzaj");
});
$("#btn5").click(function () {
    $("#testParagraf").html("<b>html</b> sadrzaj");
});
```

Izmena sadržaja

- Metode za izmenu sadržaja elementa su:
- `append()` – dodaje sadržaj na kraj postojećeg
- `prepend()` – ubacuje sadržaj na početak postojećeg
- `after()` – ubacuje sadržaj nakon selektovanog elementa
- `before()` – ubacuje sadržaj ispred sel. elementa.

```
$("li").append("Dodato na kraj.");
$("li").prepend("Dodato ispred.");
```

```
function dinamicBtns() {
    var btn1 = "<button id='btn1'>Prvo</button>";
    var btn2 = "<button id='btn2'>Drugo</button>";
    var btn3 = document.createElement("button");
    btn3.innerHTML = "Treće";
    $("body").append(btn1, btn2, btn3);
}
```

Izbacivanje elementa

- Metod `remove()` izbacuje selektovane elemente i odgovarajuće deca elemente.
- `$("#div1").remove();`

Pražnjenje elementa

- Metod `empty()` vrši pražnjenje tj izbacivanje dece elemenata nad selektovanim elemementima.
- `$("#div1").empty();`

CSS klase

- Dodavanje i izbacivanje CSS klase:
- `$("button").click(function () {`
- `    $("li").addClass("zeleno");`
- `    $("li, p").removeClass("standardno");`
- `});`

Toglovanje CSS kalse

- Često je potrebno obezbediti naizmenično dodavanja odnosno izbacivanje iste CSS klase. Na primer:

```
$("button").click(function () {
    $("li, p").toggleClass("zeleno");
});
```

Čitanje vrednosti svojstva

- Čitanje određenog svojstva koje je važeće za element:

```
alert("Background color = " + $("body").css("background-color"));
```

Definisanje svojstava

- Neposredno dodavanje svojstava selektovanim elementima se takođe vrši primenom funkcije `css()`. Na primer:
- ```
  $("h3").css({
  "background-color": "lightgreen",
  "color":"darkgreen",
  "font-size": "200%"
  });
  ```

Funkcije koje koriste DOM stablo:

- `$("span").parent(); // roditeljski el.`
- `$("span").parents(); // svi rodit. el.`
- `$("span").parents("div"); // svi rod. el koji su div`
- `$("span").parentsUntil("div"); // svi rod. el do div`
  ```

Zadaci i obećanja

- U JavaScript-u funkcije se mogu tretirati kao objekti i kao takve prosleđivati drugim funkcijama ili objektima. Neke funkcije zahtevaju određeno vreme za završetak koje može biti predugo za korisnika i izazvati zastoj u radu i samim tim predstavlja loše korisničko iskustvo.
- Dakle, moguće je da jedna funkcija sačeka završetak druge funkcije i u zavisnosti od rezultata izvršavanja vrši opciono izvršavanje.
- Tipičan primer je tzv. **pretplatnički način rada** u kom se jedan objekat (ili funkcija) pretplaćuje na određene događaje i izvršavaju pridružene funkcije kada se taj događaj desi. Tako se može napraviti čitav lanac opcionih funkcija u zavisnosti od događaja.

- Odloženi zadaci se kreiraju na sledeći način:

```
var odlZadatak1 = $.Deferred();
var odlZadatak2 = $.Deferred(onAkcija);
```

- Ako se zadatku prosledi funkcija zadatak počinje da se izvršava (`odlZadatak2`). Odloženi zadatak se može naći u dva stanja, npr. u zavisnosti od rezultata izvršavanja, u stanju „`resolved`“ ili „`rejected`“. Izvršavanje se može pokrenuti i na drugi način:

```
var deferred = $.Deferred();

deferred.done(function () {
    alert("ura!");
});
console.log(deferred.state());  // "čekanje"
deferred.resolve(); // kreira promise i razrešava stanje
console.log(deferred.state());  // "resolved"
deferred.reject();  //  nema efekta jer je Promise već razrešen
```

- Pri izvršenju funkcija `resolve()` ili `reject()` vrši se razrešenje stanja i kreiranje `Promise` objekta za koji se vezuje dalje izvršavanje. Na primer, može se funkcijom `done()` pokrenuti izvršavanje druge funkcije.

- Primer:

```
<script>
    var deferred = $.Deferred();

    $(function () {
        $("#btn0").click(function () {
            alert("deferred.state():" + deferred.state());
        });
        $("#btn1").click(function () {
            deferred.resolve();
        });
        $("#btn2").click(function () {
            deferred.reject();
        });
        $("#btn3").click(function () {
            deferred.done(prikaziStanje);
        });
    });

    function prikaziStanje() {
        alert("ura!");
    }
</script>
```

```
<body>
    <button id="btn0">stanje deferred objekta</button>
    <button id="btn1">resolve</button>
    <button id="btn2">reject</button> <br />
    <button id="btn3">uključi čekanje</button> <br />
</body>
```

- Svaki odloženi objekat obezbeđuje obećanje (engl. promise) koje obezbeđuje spoljnom kodu da se pretplati na uspešno/neuspešno stanje. Pretplatom se obezbeđuje da se spoljni kod izvrši ako se odloženi zadatak izvrši uspešno/neuspešno.
- `odlZadatak.promise();`
- `odlZadatak.promise(target);`
- Razlika između odloženih zadataka i obećanja je što se obećanja ne mogu razrešiti funkcijama `resolve()` i `reject()`.

```
var deferred = $.Deferred();
var promise = deferred.promise();
console.log(promise.state());  // "čekanje"
//promise.resolve(); // greska!!
deferred.resolve(); // ok!
console.log(promise.state());  // "resolved"
//promise.reject(); //  greska!!
```

- Dodela funkcije koja će se pozvati ako se zadatak uspešno završi je pomoću funkcije **`done`**`()`, a ako se neuspešno završi pomoću funkcije **`fail`**`()`.
- Metode `done()` i `fail()` se mogu nadovezivati ili više njih povezati, na primer: `odlZadatak.done(fja1).done(fja2).fail(fja3);`
- Postavljanje funkcija koje će se izvršiti kada se zadatak završi uspešno ili ne:

```
var odlZadatak = $.Deferred();
odlZadatak.promise()
    .done(function (val) {  alert("done: " + val);  })
    .fail(function (val) {  alert("fail: " + val);  });

$(function () {
    $("#btn0").click(function () {
        alert("odlZadatak.state():" + odlZadatak.state()); });
    $("#btn1").click(function () {
        odlZadatak.resolve("111"); });
    $("#btn2").click(function () {
        odlZadatak.reject("666"); });
});
```

```
<body>
    <button id="btn0">state</button>
    <button id="btn1">done</button>
    <button id="btn2">fail</button> <br />
</body>
```

Ulančavanje - then

- Funkcija **`then(.)`** vraća novi zadatak koji se na dalje primenjuje na isti način.
- Izvršavanje funkcije nakon uspešnog zadatka, osim primenom funkcije `done()`, može da se izvede i primenom funkcije `then(.)` sa jednim argumentom, na primer:
- `odlZadatak.then(fja1);`
- Ako funkcija ima dva argumenta, prvi je za uspešno završavanje zadatka, a drugi za neuspešno završavanje.
- `odlZadatak.then(fja1, fja2);`

```
var odlZadatak = $.Deferred();

odlZadatak
    .done(function (val) {
        val *= val;
        alert("done: " + val);
        return val;
    })
    // umesto then testirati done
    .then(function (val) {
        val += 5;
        alert("then: " + val);
        return val;
    })
    .done(function (val) {
        val -= 1;
        alert("done2: " + val);
        return val;
    })
odlZadatak.resolve(2);

Napomena: Nakon then je formirana nova
vrednost, nakon done ostaje ista.
```

Funkcija when

- Funkcija when konvertuje bilo koji objekat u odloženi zadatak. Na primer:

```
$.when(fjaFaktorijel(10000))
    .then(function (val) {
        alert("vrednost je: " + val);
    });
```

- Takođe, ova funkcija se koristi kada se više objekata testira na završetak pre dalje obrade, na primer:

```
$.when(obj1, obj2).then(fja1);
```

jQuery AJAX

- jQuery znatno olakšava rad sa AJAXom nudeći nekoliko osnovnih funkcija. Najviše se koriste: `ajax`, `load`, `post` i `get`.

- $.ajax(.) je najgeneralnija metoda. Postoje druge metode koje su specijalizovane za određeni tip zahteva.

$.ajax()

- Sintaksa je:
- `$.ajax(url);`
- `$.ajax(url, { name: value, name: value, ...});`
- `$.ajax({ url: value, name: value, name: value, ...});`
- `url` – obavezan. Predstavlja adresu funkcije tj. web strane koja će biti pozvana. (podr: tekuca strana)

- Neki mogući parametri su:
- `async` – `boolean`. Da li je zahtev sinhron ili ne. (podr: true)
- `cache` – `boolean`. Definiše da li se kešira zahtev ili ne. (podr: true. false za dataType 'script' and 'jsonp')
- `data` – podaci koji se šalju serveru. (Type: PlainObject ili String ili Array)
- `dataType` - `text, xml, html, json, jsonp, script` (podr: Intelligent Guess (xml, json, script, ili html)
- `contentType` – tip podatka koji se šalje preko hedera http zahteva. Podrazumevano je: application/x-www-form-urlencoded; charset=UTF-8, može biti: application/json; charset=UTF-8 ili application/xml; charset=UTF-8
- `type` – tipzahteva (GET ili POST) (podr: GET)
- `username` – kor. Ime
- `password` - lozinka

Napomena: Sve neobavezne vrednosti su opcione. Njihove podrazumevane vrednosti se postavljaju metodom ajaxSetup().

```
$.ajax({
    type: "POST",
    url: "http://www.viser.edu.rs/index.php",
    dataType: "html",
    success: function (val) {
        alert("ok " + val);
    },
    error: function () {
        alert("error");
    }
});
```

- Metoda `ajax()` vraća objekat za odlaganje izvršavanja. Tako se može isti efekat postići i primenom metoda koje znače da je odloženo izvršavaje završeno ispravno ili ne, na primer.

```
$.ajax("http://www.viser.edu.rs/pservis.php?username=ime&password=lozinka")
    .done(fja1)
    .fail(fja2)
    .fail(fja3);
```

- Ako zahtev uspe, fja1 prima prvi argument **sadržaj** koji je vratio server (string, xml, json). Drugi argument je **status**, a treći je **xhr** (XmlHttpRequest).

$( ).load

- Pomoću jQuery AJAX metoda mogu se dobaviti podaci u vidu: teksta, HTML, XML, ili JSON podaci od udaljenog servera koristeći HTTP Get odnosno HTTP Post.
- `$(selector).load(url, querystring, callback);`
- `url` - obavezan je i predstavja adresu koja se čita.
- `querystring` – opcioni parametar kojim se definiše upit tipa key/value.
- `callback` - opcioni parameter koji je funkcija koja se izvršava nakon što se load() metoda završi.

```
$("#div1").load("test.txt");

$("button").click(function () {
   $("#div1").load("test.txt", function (response, status, xhr) {
      if (status == "success")
          alert("Učitano uspešno!");
      if (status == "error")
          alert("Greška! xhr.status=" + xhr.status + " xhr.statusText=" + xhr.statusText);
    });
});
```

jQuery: get, post

- Postoje dve metode za request-response komunikaciju između klijenta i servera:
  - **GET** - slanje zahteva za određenim resursom
  - **POST** - slanje podataka koji se obrađuju od strane određenog resursa

- GET je osnovni način za dobijanje podataka od servera. **Napomena:** GET metod može vratiti i keširane podatke.
- POST može takođe biti korišćen za dobijanje takođe nekih podataka od servera. Metod POST ne kešira podatke, a takođe koristi se u slučaju kada se šalje više podataka i zaštićeno uz zahtev.

- `$.get(url, callback);`
- `url` - obavezan je i predstavlja adresu koja se čita.
- `callback` - opcioni parameter koji je funkcija koja se izvršava nakon što se get() metoda završi. Prvi parametar callback funkcije biće sadržaj tj. dobijeni podaci, a drugi parametar je status zahteva.

- `$.post(url, data, callback);`
- `url` - obavezan je i predstavlja adresu koja se čita.
- `data` – opcioni parametar. Predstavlja podatke koji se šalju zajedno sa zahtevom.
- `callback` - opcioni parameter koji je funkcija koja se izvršava.

```
$("button").click(function () {
    $.get("demo_test.asp", function (data, status) {
        alert("Podaci: " + data + " status: " + status);
    });
});
```

```
var sendData = {
    "ime": "Perica",
    "prezime": "P"
};

$(document).ready(function () {
    $("button").click(function () {
        $.post("login.php", sendData,
            function (data, status) {
                alert("Data: " + data + "\nStatus: " + status);
            });
    });
});
```

- Postoje još neke specijalizovane metode koje predstavljaju specifične pozive `ajax()` metode.
- Na primer, ako se postavi:
- `type: POST` – analogno sa - `$.post()`
- `type: GET` – analogno sa - `$.get()`
- `dataType: JSON` – analogno sa - `$.getJSON()`
- `dataType: SCRIPT` – analogno sa - `$.getScript()`