**Događaji**

Jedinica: 28 od 30

**Rezime**

* Događaj je pojam koji opisuje neku pojavu.
* U jednoj HTML5 aplikaciji događaji mogu biti izazvani od strane browsera, ali i od strane korisnika takve aplikacije.
* JavaScript poseduje mogućnost reagovanja na događaje definisanjem programske logike, koja će se aktivirati kada do neke pojave dođe.
* Logika koja se aktivira kada dođe do nekog događaja (engl. event) naziva se event handler.
* Metoda addEventListener koristi se za registrovanje event handlera za određeni event.
* Objekat događaja je objekat koji sadrži informacije o samom događaju.
* Funkcija preventDefault otkazuje podrazumevano ponašanje događaja.

Sam pojam događaja pomenut je još na početku priče o JavaScript jeziku, kada su izneti različiti načini za integraciju JS koda u stranicu. Jedan od načina je bio integracija unutar [event handlera](https://www.link-elearning.com/linkdl/opisPojma.php?id=144354)nekog događaja. Ova lekcija biće u potpunosti posvećena pojmu događaja i rukovanju njima u jednoj HTML5 aplikaciji.

**Šta je događaj?**

Kao što i samo ime kaže, događaj je pojam koji opisuje neku pojavu. Kada u prirodi sevne munja, takva pojava se može nazvati događajem. U jednoj HTML5 aplikaciji, događaji mogu biti izazvani od strane browsera, ali i od strane korisnika takve aplikacije. Primeri događaja u jednoj HTML5 aplikaciji mogu biti:

* klik na dugme
* prelazak mišem preko kontrole
* završetak učitavanje stranice
* početak učitavanja stranice

Pojava koje mogu da izazovu događaj u HTML5 aplikaciji je zaista mnogo.

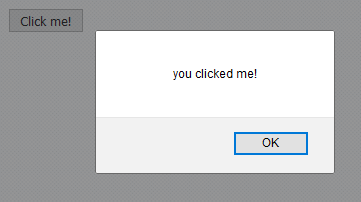
JavaScript poseduje mogućnost reagovanja na događaje do kojih dolazi u aplikaciji. Na taj način je moguće definisati programsku logiku, koja će se aktivirati kada do neke pojave dođe. Logika koja se aktivira kada dođe do nekog događaja (engl. event) naziva se *event handler*.

**Inline reagovanje na događaje**

Da bi se reagovalo na događaje u jednoj web aplikaciji, postoji nekoliko načina. Osnovni, a samim tim i vrlo ograničen način za rukovanje događajima jeste korišćenje HTML atributa na elementima:

|  |  |
| --- | --- |
| 1 | <button onclick='alert("you clicked me!")'>Click me!</button> |

Na button elementu definisan je atribut onclick, sa vrednošću koja predstavlja logiku koja će se aktivirati u trenutku kada korisnik klikne na ovaj button elemenat (slika 28.1).



*Slika 28.1 – Klik na button element*

Ipak, korišćenje ovakvog pristupa, kojim se piše takozvani inline JavaScript, nikako se ne preporučuje, zbog ograničenja u pogledu preglednosti, portabilnosti, otežanog održavanja i slično. Zato se preferiraju ostali načini koje HTML DOM API omogućava za obradu događaja.

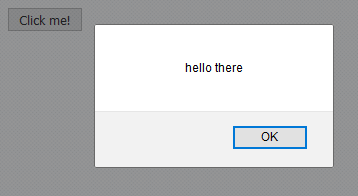
**Reagovanje na događaje korišćenjem događaja kao svojstava**

U prethodnom primeru, za obradu događaja oslanjali smo se na atribute HTML elemenata. Identičan efekat može se postići i na nešto drugačiji način.

Kada se dobije referenca na element u DOM strukturi, veoma je lako registrovati se na neki event korišćenjem svojstava koja predstavljaju događaje:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <html>  <head>   <title>DOM</title>  </head>  <body>     <button id="the-button">Click me!</button>     <script>      var button = document.getElementById("the-button");    button.onclick = displayMessage;      function displayMessage(){     alert('hello there');    }     </script>    </body>  </html> |

U primeru je iskorišćeno svojstvo onclick, za čiju vrednost je postavljena referenca na funkciju koja je potrebno da se izvrši kada do eventa dođe. Primer proizvodi rezultat kao na slici.

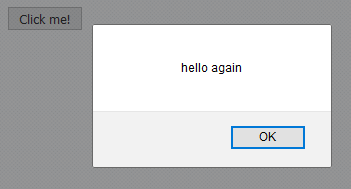


*Slika 28.2 – Klik na button element aktivira prikaz popupa*

Prikazana dva načina za registrovanje event handlera su potpuno ravnopravna, odnosno, njima se postiže identičan efekat. Njihov ograničavajući faktor jeste mogućnost registrovanja samo jednog eventa:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | <html>  <head>   <title>DOM</title>  </head>  <body>     <button id="the-button">Click me!</button>     <script>      var button = document.getElementById("the-button");    button.onclick = displayMessage;    button.onclick = displayAnotherMessage;      function displayMessage(){     alert('hello there');    }      function displayAnotherMessage(){     alert('hello again');    }     </script>    </body>  </html> |

U ovom primeru, dva puta je postavljena vrednost svojstva onclick. S obzirom da je reč o običnom svojstvu, ono može imati samo jednu vrednost. Tako će vrednost svojstva onclick biti ona koja je postavljena poslednja, a primer će proizvesti rezultat kao na slici 28.3.



*Slika 28.3 – Klik na button element aktivira prikaz popupa (2)*

Kao što se sa slike 28.3 može videti, prilikom klika na button element, izvršava se logika samo poslednje dodatog event handlera.

**addEventListener**

Još jedan način za registrovanje logike koja je potrebno da se izvrši kada dođe do nekog događaja jeste korišćenje metode addEventListener:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <html>  <head>   <title>DOM</title>  </head>  <body>     <button id="the-button">Click me!</button>     <script>      var button = document.getElementById("the-button");    button.addEventListener("click", displayMessage);      function displayMessage(){     alert('hello there');    }     </script>    </body>  </html> |

Metoda addEventListenerse koristi za registrovanje event handlera za određeni event. Osnovna osobina ovakvog načina za dodavanje event handlera jeste mogućnost višestruke pretplate na jedan isti događaj:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <html>  <head>   <title>DOM</title>  </head>  <body>     <button id="the-button">Click me!</button>     <script>      var button = document.getElementById("the-button");      button.addEventListener("click", displayMessage);    button.addEventListener("click", displayAnotherMessage);      function displayMessage(){     alert('hello there');    }      function displayAnotherMessage(){     alert('hello again');    }     </script>    </body>  </html> |

U primeru su izvršene dve registracije različitih event handlera za isti događaj – click. Klikom na button element izvršiće se prvo logika funkcije displayMessage, a zatim i funkcije displayAnotherMessage.

Funkcija addEventListener ima sledeću sintaksu:

|  |  |
| --- | --- |
| 1 | addEventListener(<event name>,<event function>,<optional cascade rule>) |

Kao što se može primetiti, funkcija addEventListener može da prihvati tri parametra od kojih su prva dva obavezna:

* event name – naziv događaja za koji se definiše event handler
* event function – logika koja se aktivira prilikom pojave događaja
* cascade rule – definiše kako se događaj propagira kroz lanac elemenata

**Korišćenjem funkcije addEventListener moguće je izvršiti dodavanje više event handler funkcija za isti event na istom elementu:**

 tačno

 netačno

**Različiti tipovi događaja**

Na samom početku obrade događaja, potrebno je znati kom eventu želimo da pridružimo neku event handler logiku. To je uostalom i prvi parametar koji metoda addEventListener prihvata. Događaja koji se mogu slušati je zaista mnogo i oni se mogu podeliti na nekoliko grupa. Neke od najznačajnijih grupa su:

* događaji miša
* događaji tastature
* događaji objekta
* događaji form elementa
* touch događaji

U nastavku lekcije će biti prikazani neki od najznačajnijih događaja svake navedene grupe.

**Događaji miša**

|  |  |
| --- | --- |
| **Događaj** | **Opis** |
| onclick | događaj do koga dolazi kada korisnik klikne na element |
| ondblclick | događaj do koga dolazi kada korisnik izvrši dupli klik na element |
| onmousedown | događaj do koga dolazi kada korisnik pritisne taster miša sa strelicom iznad elementa |
| onmouseenter | događaj do koga dolazi kada strelica miša uđe u oblast elementa |
| onmouseleave | događaj do koga dolazi kada strelica miša napusti oblast elementa |
| onmousemove | događaj do koga dolazi kada se strelica miša pomera, a nalazi se iznad elementa |
| onmouseover | događaj do koga dolazi kada strelica miša uđe u oblast elementa, ali i bilo kojeg njegovog potomka |
| onmouseout | događaj do koga dolazi kada strelica miša napusti oblast elementa, ali i bilo kojeg njegovog potomka |
| onmouseup | događaj do koga dolazi kada korisnik otpusti taster miša, a strelica se nalazi iznad elementa |

*Tabela 28.1 – Događaji miša*

**Događaji tastature**

|  |  |
| --- | --- |
| **Događaj** | **Opis** |
| onkeydown | događaj do koga dolazi kada korisnik pritisne neki taster na tastaturi |
| onkeypress | događaj do koga dolazi kada korisnik pritisne neki taster na tastaturi i tako nešto proizvede karakter |
| onkeyup | događaj do koga dolazi kada korisnik otpusti neki taster na tastaturi |

*Tabela 28.2 – Događaji tastature*

**Događaji objekta**

|  |  |
| --- | --- |
| **Događaj** | **Opis** |
| onload | događaj do koga dolazi kada se završi učitavanje nekog objekta |
| onresize | događaj do koga dolazi kada se veličina vidnog polja browsera promeni |
| onscroll | događaj do koga dolazi kada se upotrebljava scrollbar |

*Tabela 28.3 – Događaji objekta*

**Događaji form elementa**

|  |  |
| --- | --- |
| **Događaj** | **Opis** |
| onblur | događaj do koga dolazi kada element izgubi fokus |
| onfocus | događaj do koga dolazi kada element dobije fokus |
| oninput | događaj do koga dolazi kada input element dobije sadržaj |
| onsubmit | događaj do koga dolazi kada se forma prosledi |

*Tabela 28.4 – Događaji form elementa*

**Touch događaji**

|  |  |
| --- | --- |
| **Događaj** | **Opis** |
| ontouchstart | događaj do koga dolazi kada korisnik dodirne ekran uređaja osetljivog na dodir |
| ontouchmove | događaj do koga dolazi kada korisnik prevlači prst po ekranu osetljivom na dodir |
| ontouchend | događaj do koga dolazi kada korisnik skloni prst sa ekrana osetljivog na dodir |

*Tabela 28.5 – Touch događaji*

|  |
| --- |
| **Napomena**    *Kada se prikazani događaji koriste unutar funkcije addEventListener(), navode se bez prefiksa on.* |

**Objekat događaja**

Nakon pretplate na event odgovarajućeg naziva i namene, potrebno je definisati logiku koja će se aktivirati kada dođe do pojave događaja. Rešavanje opisanog problema jednostavno se realizuje kreiranjem metode, koja se drugačije naziva event handler funkcija. To je regularna funkcija, kao i svaka druga. Ipak, za uspešno rukovanje nekim događajima unutar *event handler funkcija*, potrebno je rukovati jednim specijalnim objektom – *objektom događaja*.

Objekat događaja je objekat koji sadrži informacije o samom događaju. Na primer, kada korisnik pritisne neki taster na tastaturi, dolazi do pojave onkeydown događaja, a unutar objekta događaja nalaze se informacije o tome koji taster je pritisnut:

|  |  |
| --- | --- |
| 1  2  3  4  5 | document.addEventListener("keydown", logKey);    function logKey(e){  console.log(e.key);  } |

U primeru je registrovana event handler funkcija za događaj keydown. Ovoga puta event handler funkcija prihvata i jedan parametar sa nazivom e. Reč je o objektu događaja. Naziv promenljive koja predstavlja parametar funkcije je proizvoljan.

Unutar event handler funkcije čita se svojstvo key, objekta događaja, i njegova vrednost se ispisuje unutar konzole. Naravno, svojstvo key ukazuje na taster koji je pritisnut na tastaturi.

|  |
| --- |
| **Napomena**    *Različiti događaji imaju objekte događaja sa različitim svojstvima.* |

**Otkazivanje događaja**

Određene radnje na HTML stranici proizvode događaje koji se automatski obrađuju od strane browsera. Jedan od primera takvih događaja jeste klik na a element. Klikom na a element, browser automatski vrši redirekciju korisnika na URL koji je definisan unutar href atributa. U ovakvim i sličnim situacijama, moguće je izvršiti otkazivanje logike koja bi se uobičajeno dogodila:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <html>  <head>   <title>DOM</title>  </head>  <body>     <a href="http://www.link-group.eu/" id="the-link">Link Group</a>     <script>      var link = document.getElementById("the-link");    link.addEventListener("click", preventDefaultEvent);      function preventDefaultEvent(e){       e.preventDefault();     console.log("Default event is prevented!");      }       </script>    </body>  </html> |

U primeru je kreiran jedan a element, koji klikom vodi na eksternu stranicu. Na a element je u JavaScript kodu dodat event handler za klik događaj, koji se ogleda u funkciji preventDefaultEvent. Unutar funkcije preventDefaultEvent, nad objektom događaja pozvana je metoda preventDefault. Ovo je metoda koja otkazuje podrazumevano ponašanje događaja. U prikazanom primeru to će biti redirekcija na stranicu definisanu href atributom. Rezultat prikazanog primera će biti ispisivanje poruke: Default event is prevented!, bez redirekcije na ULR definisan href atributom.

Još jedan od načina da se otkaže podrazumevano ponašanje događaja jeste da se iz event hadler funkcije vrati vrednost false. Vraćanjem false vrednosti iz event handler funkcije se, pored otkazivanja podrazumevane logike, otkazuje i propagiranje događaja. Pojam propagiranja događaja je nešto što do sada nije pomenuto, tako da će sledeći redovi biti posvećeni upravo ovom pojmu.

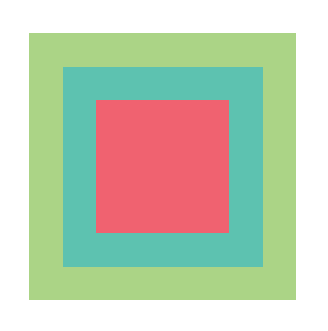
**Propagiranje događaja**

JavaScript Event model zasniva se na pojmu koji se naziva propagiranje događaja. Propagiranje događaja je pojam koji je usko povezan sa strukturom roditelj-dete, koju elementi međusobno grade na jednoj HTML stranici.

Za demonstraciju propagiranja događaja, biće korišćena sledeća HTML struktura:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | <html>  <head>   <title>DOM</title>   <style type="text/css">    #div1{     width: 400px;     height: 400px;     background-color: #ABD486;     display: flex;     justify-content: center;     align-items: center;    }      #div2{     width: 300px;     height: 300px;     background-color: #5DC2B0;     display: flex;     justify-content: center;     align-items: center;      }      #div3{     width: 200px;     height: 200px;     background-color: #F06270      }   </style>  </head>  <body>     <div id="div1">    <div id="div2">     <div id="div3">     </div>    </div>   </div>    </body>  </html> |

HTML struktura se sastoji iz tri div elementa, koji su postavljeni jedan unutar drugog. Roditeljski element je div1, njegov direktan potomak div2, dok je potomak elementa div2 element div3. Prikazani kod na stranici proizvodi efekat kao na slici.



*Slika 28.4 – Tri div elementa jedan unutar drugog*

Pitanje je sledeće:

*Kada se izvrši klik na element koji je u strukturi elementa najdublje (div3), da li će se takav klik tretirati kao klik samo na div3 element ili i na elemente div2 i div1?*

Elementi div2 i div3, svakako su članovi elementa div1, pa je klik na div3 element, zapravo klik i na div2 i na div1. Da li je to tako?

Da bi se odgovorilo na postavljena pitanja, za sva tri elementa biće kreirani event handleri za click događaj:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | var div1 = document.getElementById("div1");  var div2 = document.getElementById("div2");  var div3 = document.getElementById("div3");    div1.addEventListener("click", eventHandler);  div2.addEventListener("click", eventHandler);  div3.addEventListener("click", eventHandler);    function eventHandler(e){     console.log("Hello from " + this.id);    } |

Ukoliko se izvrši klik na div3 element, u konzoli se dobija sledeći ispis:

|  |  |
| --- | --- |
| 1  2  3 | Hello from div3  Hello from div2  Hello from div1 |

Analizom izlaza se može zaključiti da se klik na div3 element tretira kao klik i na div3 element, ali i na div2 i na div1 elemente.

JavaScript događaji prolaze dve etape nakon aktiviranja na nekom elementu. To su faze:

* capturing
* bubbling

Događaj se prvo propagira od najvišeg elementa u hijerarhiji do najnižeg elementa, koji je u suštini i izazvao događaj. Ta faza se naziva *capturing faza*.

Nakon što je događaj stigao na najniži izvorišni element, on započinje fazu koja se naziva bubbling, odnosno propagiranje događaja nazad na vrh. Svi događaji se u JavaScript jeziku podrazumevano hvataju u bubbling fazi, i baš zbog toga je u prethodnom primeru prvo aktiviran event handelr div3elementa. Ali, kako uhvatiti događaj dok je još u capturing fazi?

Odgovor na ovo pitanje leži u sintaksi nešto ranije prikazane metode za registrovanje event handler metoda:

|  |  |
| --- | --- |
| 1 | addEventListener(<event name>,<event function>,<optional cascade rule>) |

Reč je, naravno, o metodi addEventListener, koja kao treći parametar prihvata jednu booleanvrednost. Ovaj parametar podrazumevano ima vrednost false, a definiše da li će događaji biti obrađeni tokom capturing faze.

|  |  |
| --- | --- |
| 1  2  3 | div1.addEventListener("click", eventHandler, true);  div2.addEventListener("click", eventHandler, true);  div3.addEventListener("click", eventHandler, true); |

Ovoga puta je prilikom registrovanja event handlera korišćenjem metode addEventListener, naveden i treći parametar sa vrednošću true. To znači da će događaji biti obrađeni tokom capturing faze. Ukoliko se nakon ove male izmene ponovo pokrene prethodni primer i izvrši klik na div3element, rezultat izvršavanja će biti sledeći:

|  |  |
| --- | --- |
| 1  2  3 | Hello from div1  Hello from div2  Hello from div3 |

S obzirom da se događaji sada obrađuju u capturing fazi, prvo se aktivira event handler logika koja je povezana sa div1 elementom.

|  |
| --- |
| **Napomena**    *Otkazivanje propagiranja događaja može se postići korišćenjem funkcije stopPropagation.* |