

JavaScript Događaji Funkcije (callback funkcije) this

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

Sadržaj

◆ JS Događaji:

- ◆ Definicija

- ◆ Vrste

- ◆ Upravljanje događajima: oslušivači i kretanje (propagacija)

- ◆ **this:**

- ◆ Korišćenje u funkcijama

- ◆ **Funkcije kao objekti prvog reda:**

- ◆ Načini definisanja i poziva

- ◆ Callback funkcije

- ◆ Primeri: codepen.io

JavaScript događaji

- ◆ Većina koda klijentske Web aplikacije se pokreće tako što korisnik inicira akciju na korisničkom interfejsu
- ◆ Fizička akcija korisnika se transformiše u programski objekat **događaja**
- ◆ Aplikacija zna koji element je odgovoran za obradu događaja na osnovu funkcije **osluškivača**
- ◆ Primeri događaja:
 - ◆ Klik miša na elementu interfejsa
 - ◆ Klik na taster tastature
 - ◆ Pomeranje, promena veličine ili zatvaranje prozora
 - ◆ Završetak učitavanje Web stranice od strane pretraživača
 - ◆ Slanje podataka forme
 - ◆ Dodir i pokreti dodira

JavaScript događaji

- ◆ JavaScript događaji nisu deo jezgra jezika, njihova realizacije je prepuštena JS rutinama koje implementiraju Web pretraživači
- ◆ Tipovi događaja:
- ◆ Događaji miša (klik, pomeraj, ulazak/izlazak pokazivača iz/ u region stranice)
- ◆ Događaji tastature (keyup, keydown)
- ◆ Događaji fokusa (focus, blur : dobijanje/gubitak fokusa)
- ◆ Događaji forme (reset, submit)
- ◆ Događaji promene prikaza (resize, scroll)
- ◆ Događaji učitavanja resursa ili pristizanja zahteva (NodeJS)
- ◆ Nisu vezani samo za korisnički interfejs, već uopšte za celokupan stek aplikacije (frontend, backend)
- ◆ Kastomizovani događaji (oni koje sami definišemo)

Rukovanje, tj, obrada događaja

- ◆ Da bi se radilo sa događajima moraju se definisati sledeći koraci:
 - ◆ **Šta** se generiše: događaja od interesa
 - ◆ **Gde** se generiše: element od interesa (izvor događaja)
 - ◆ **Kako** reagovati: odgovor aplikacije na događaj
-
- ◆ U praksi se često dešava da korisnik ne pokrene akciju direktno nad elementom, ili da različiti elementi reaguju na identičan tip događaja
 - ◆ Potrebno je ustanoviti i mehanizam **prosleđivanja** i **dodeljivanja** događaja elementima

Šta se generiše: objekat tipa Event

- ◆ Logička reprezentacija događaja
- ◆ Korena klasa iz koje su izvedene klase za specifične tipove događaja (MouseEvent, KeyboardEvent)
- ◆ Karakteristični atributi:

Atribut	Značenje
type	Tip događaja kao string simbolička konstanta (click, mousedown, mouseup)
timeStamp	Trenutak generisanja događaja
currentTarget	Element kojem je pridružen osluškivač, tj. koji je registrovan za prijem i obradu događaja
target	Element nad kojim je generisan događaj

MouseEvent

◆ Različite radnje miša opisane skupom informacija

Atribut	Značenje
button	Dugme miša (0 – levo, 1 – srednje, 2 - desno)
click	Klik dugmeta
mouseDown	Pritisak dugmeta
mouseUp	Otpuštanje dugmeta
mouseMove	Pomeranje miša
mouseenter, mouseLeave	Pokazivač miša ulazi ili napušta površinu elementa kojem je pridružen osluškivač
pageX, pageY	Pozicija miša relativna u odnosu na gornji, levi ugao dokumenta (x-horizontalno, y-vertikalno)
screenX, screenY	Pozicija miša relativna u odnosu na gornji, levi ugao ekrana (x-horizontalno, y-vertikalno)

```
<div class="box">
  <p>
    Pozicija miša u okviru kontejnera.
  </p>
  <p>
    <code>pageX</code>: <span id="x">n/a</span>
  </p>
  <p>
    <code>pageY</code>: <span id="y">n/a</span>
  </p>
</div>
```

**Primer –
ispisivanje
pozicije
miša: HTML**

Pozicija miša u okviru kontejnera.

pageX: 419

pageY: 76

Primer – ispisivanje pozicije miša: JS

```
var box = document.querySelector(".box");
var pageX = document.getElementById("x");
var pageY = document.getElementById("y");

function updateDisplay(event) {
    pageX.innerText = event.pageX;
    pageY.innerText = event.pageY;
}

box.addEventListener("mousemove", updateDisplay);
box.addEventListener("mouseenter", updateDisplay);
box.addEventListener("mouseleave", updateDisplay);
```

KeyboardEvent

- ◆ Informacije vezane za korišćenje tastera tastature

Atribut	Značenje
key	DOM string kao identifikator tastera tastature ('A', 'a', ...)
keyDown	Pritisak tastera
keyPress	Produžen pritisak tastera (duže od par sekundi)
keyUp	Otpuštanje tastera

Form event

- ◆ Slanje ili poništavanje podataka forme

Atribut	Značenje
submit	Poziva se na klik <i>submit</i> dugmeta
reset	Poziva se na klik <i>reset</i> dugmeta

```
<form id="test_form">  
  <input type="text" id="txt"  
  name="first_name" value="text"  
  placeholder="Text field..." />  
  <input type="submit"  
  name="form_submit" value="Submit" />  
</form>
```

**Primer –
submit
događaj:
HTML**

text

Submit

Primer – submit događaj: JS i alert prozor

```
var form = document.querySelector('#test_form');  
var inputTxt = document.getElementById("txt");  
  
form.addEventListener('submit', function(event) {  
    alert('text: ' + inputTxt.value);  
});
```

An embedded page at s.codepen.io says

text: text

OK

Koji osluškivač će biti izvršen za nastali događaj ?

- ◆ Elementi mogu sadržavati druge elemente i svi mogu imati registrovane osluškivače za isti tip događaja

- ◆ Na primer, korisnik klikće mišem na "xyz":

```
<body>
```

```
  <table>
```

```
    <tr>
```

```
      <td>xyz</td>
```

```
    </tr>
```

```
  </table>
```

```
</body>
```

- ◆ Ako svi tagovi imaju pridružene osluškivače, koji će biti pozvan ?
- ◆ Nekada unutrašnji elementi (čvorovi) obrađuju događaj
- ◆ Nekada je pogodnije da spoljni (roditeljski) čvorovi obrade događaj

Kretanje događaja kroz DOM stablo

◆ Kretanje od vrha ka dubini hijerarhije (*capture*)

Početak od spoljnog elementa (vrha hijerarhije) ka unutrašnjim elemenatima (deci)

Svaki element na putanji može eksplicitno zaustaviti put događaja

```
event.stopPropagation()
```

```
element.addEventListener(eventType, handler, true)
```

◆ Kretanje od unutrašnjih elemenata ka vrhu (*bubble*)

Početak od ugnježđenih elemenata ka roditeljima

Svaki element na putanji može eksplicitno zaustaviti put događaja

```
event.stopPropagation()
```

```
element.addEventListener(eventType, handler, false)
```

ukoliko se ne navede 3. parametar, false je podrazumevano

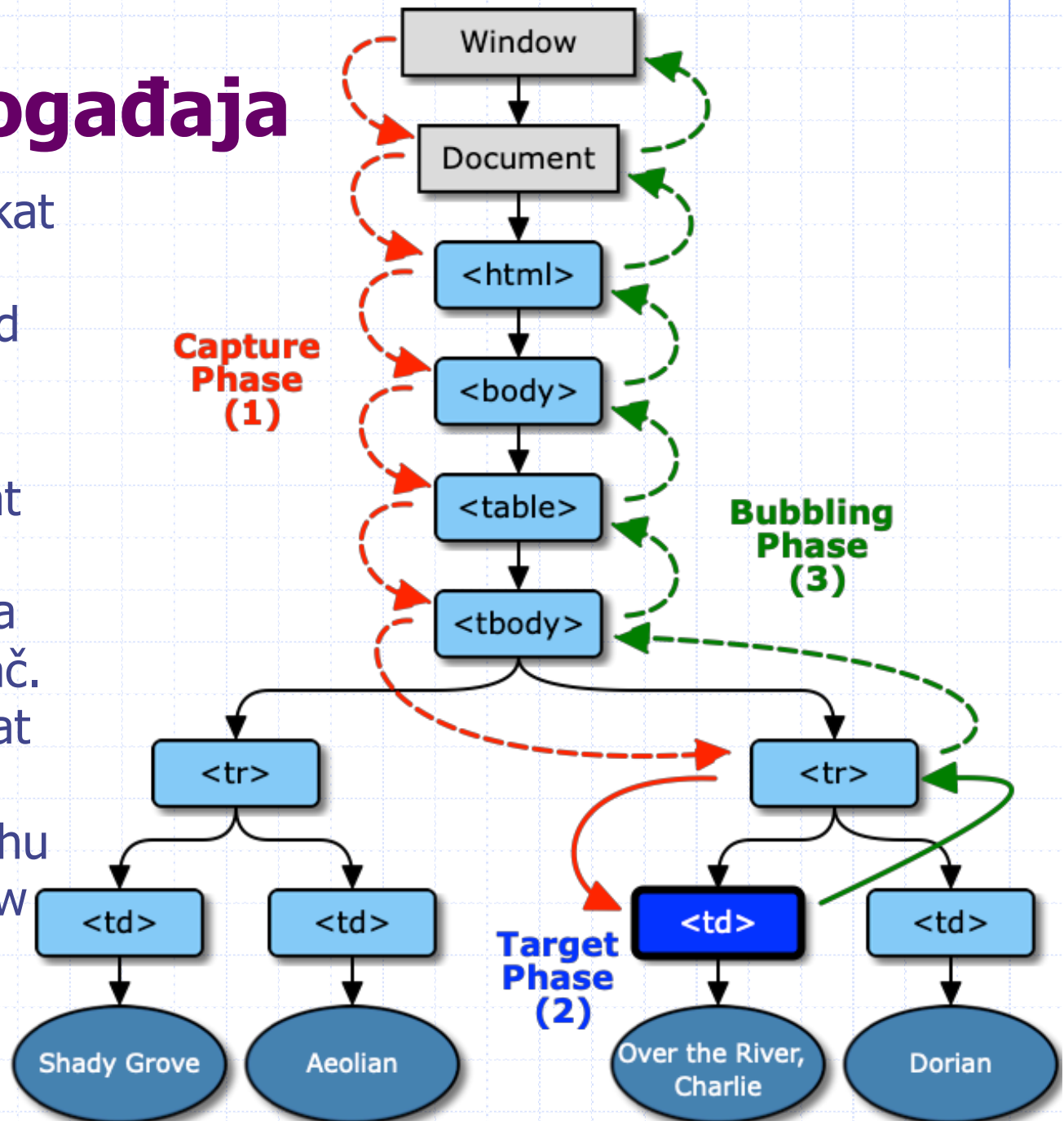
◆ U većini slučajeva se koristi *bubble* princip

Kretanje događaja

Capture faza: Objekat događaj putuje od roditelja u dubinu, od window objekta do roditelja izvora

Target faza: Objekat događaj stiže na cilj, tj. u element koji ima registrovan osluškivač.

Bubble faza: Objekat događaj putuje od roditelja izvora ka vrhu hijerarhije, tj. window objektu



JS Event objekat

- ◆ **event.target** – element **izvor** događaja (na primer, slika na koju se klikne). U slučaju da elementu nije pridružen osluškivač, događaj se prenosi kroz DOM čvorove dok se ne nađe na osluškivač registrovan za taj tip događaja
- ◆ Konkretni element ne mora imati pridružen osluškivač
- ◆ Mehanizam se naziva delegiranje događaja (*event delegation*)
- ◆ **event.currentTarget** – element kojem je pridružen **osluškivač** za konkretni događaj
- ◆ U slučaju da je konkretnom HTML elementu pridružen osluškivač događaj, dva atributa se odnose na isti element
- ◆ U suprotnom mogu vratiti dva različita elementa

Višestruki obrađivači događaja

- ◆ Šta se dešava ako se osluškivač za isti tip događaja pridruži roditelju i detetu ?
- ◆ Da li se pokreću oba ?
- ◆ Koji se pokreće prvi ?

```
<div id="outer">  
  Click me!  
  <div id="inner">  
    No, click me!  
  </div>  
</div>
```

```
const outer = document.querySelector('#outer');  
const inner = document.querySelector('#inner');  
outer.addEventListener('click', onOuterClick);  
inner.addEventListener('click', onInnerClick);
```

Kretanje događaja od deteta ka roditelju (*bubble*)

- ◆ Oba oslušivača će se izvršiti klikom na element dete
- ◆ Podrazumevano, oslušivač deteta će se prvi izvršavati

div id="outer"

div id="inner"



```
<div id="outer">  
  Click me!  
  <div id="inner">  
    No, click me!  
  </div>  
</div>
```

Click me!

No, click me!



Kretanje događaja od deteta ka roditelju (*bubble*)

- ◆ Oba oslušivača će se izvršiti klikom na element dete
- ◆ Podrazumevano, oslušivač deteta će se prvi izvršavati

`div id="outer"`

`div id="inner"`



```
<div id="outer">  
  Click me!  
  <div id="inner">  
    No, click me!  
  </div>  
</div>
```

Click me!

No, click me!



stopPropagation()

- ◆ Prosledjivanje događaja kroz lanac roditelja se može eksplicitno zaustaviti sa `event.stopPropagation()`

```
function onInnerClick(event) {  
    inner.classList.add('selected');  
    console.log('Inner clicked!');  
    event.stopPropagation();  
}
```

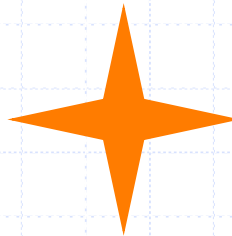
stopPropagation - primer

- ◆ Osluškivač će se izvršiti samo u detetu

```
function onInnerClick(event) {  
  inner.classList.add('selected');  
  console.log('Inner clicked!');  
  event.stopPropagation();  
}
```

div id="outer"

div id="inner"



```
<div id="outer">  
  Click me!  
  <div id="inner">  
    No, click me!  
  </div>  
</div>
```

Click me!

No, click me!



stopPropagation - primer

- ◆ Osluškivač će se izvršiti samo u detetu

```
function onInnerClick(event) {  
  inner.classList.add('selected');  
  console.log('Inner clicked!');  
  event.stopPropagation();  
}
```

div id="outer"



div id="inner"



```
<div id="outer">  
  Click me!  
  <div id="inner">  
    No, click me!  
  </div>  
</div>
```

Click me!

No, click me!



Kretanje događaja od roditelja ka detetu (*capture*)

- ◆ Da bi se događaj prosleđivao u suprotnom smeru, `addEventListener` metodi se prosleđuje 3. parametar

`event.addEventListener('click', onClick, true)`

```
const outer = document.querySelector('#outer');
const inner = document.querySelector('#inner');
outer.addEventListener('click', onOuterClick, true);
inner.addEventListener('click', onInnerClick);
```


Kretanje događaja od roditelja ka detetu (*capture*)

- ◆ Da bi se događaj prosleđivao u suprotnom smeru, `addEventListener` metodi se prosleđuje 3. parametar

`event.addEventListener('click', onClick, true)`

`div id="outer"`

`div id="inner"`

```
<div id="outer">  
  Click me!  
  <div id="inner">  
    No, click me!  
  </div>  
</div>
```

Click me!

No, click me!

Kretanje događaja od roditelja ka detetu (*capture*)

- ◆ Da bi se događaj prosleđivao u suprotnom smeru, `addEventListener` metodi se prosleđuje 3. parametar

`event.addEventListener('click', onClick, true)`

`div id="outer"`

`div id="inner"`

"Outer clicked!"

"Inner clicked!"

Click me!

No, click me!

stopPropagation() (capture)

- ◆ Prosledjivanje događaja kroz lanac roditelja se može eksplicitno zaustaviti sa `event.stopPropagation()`

```
function onOuterClick(event) {  
    outer.classList.add('selected');  
    console.log('Outer clicked!');  
    event.stopPropagation();  
}
```

stopPropagation() – capture primer

- ◆ Osluškivač će se izvršiti samo u roditelju

```
function onOuterClick(event) {  
  outer.classList.add('selected');  
  console.log('Outer clicked!');  
  event.stopPropagation();  
}
```

div id="outer"

div id="inner"

Click me!

No, click me!

"Outer clicked!"

stopPropagation() – capture primer

- ◆ Osluškivač će se izvršiti samo u roditelju

```
function onOuterClick(event) {  
  outer.classList.add('selected');  
  console.log('Outer clicked!');  
  event.stopPropagation();  
}
```

div id="outer"

div id="inner"



X

"Outer clicked!"

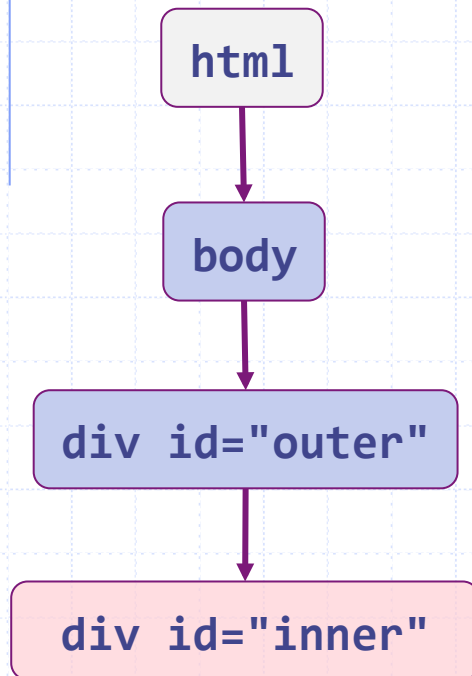
Click me!

No, click me!



Propagacija događaja

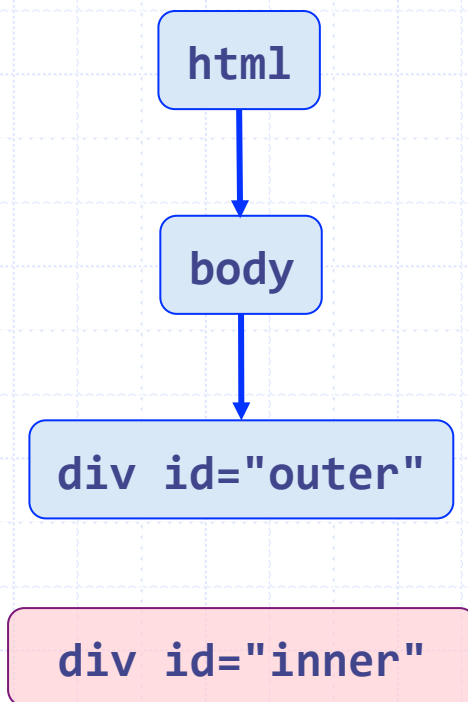
- ◆ U opštem slučaju, pretraživač **uvek** kreira putanju od izvora događaja do vrha hijerhije



```
<html>
  <head>
    <meta charset="utf-8">
    <title>JS događaji: Dva osluškivača</title>
  </head>
  <body>
    <div id="outer">
      Click me!
      <div id="inner">
        No, click me!
      </div>
    </div>
    <button>Reset</button>
  </body>
</html>
```

1 - Capture faza

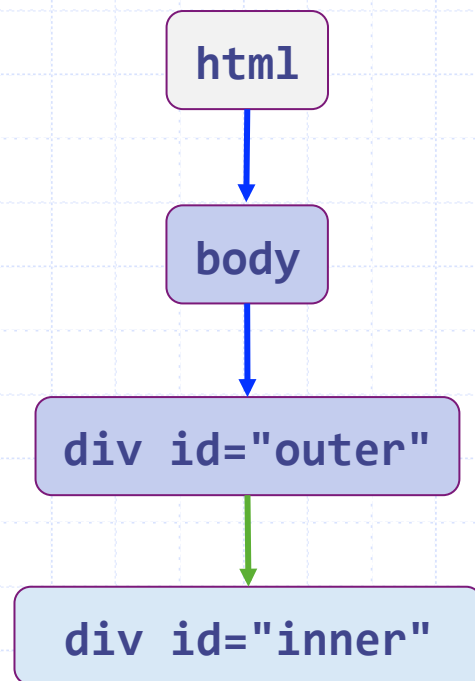
- ◆ Pretraživač kreće od vrha putanje i izvršava sve osluškivače koji imaju **true** vrednost metode do kraja putanje, tj. elementa izvora događaja



```
<html>
  <head>
    <meta charset="utf-8">
    <title>JS događaji: Dva osluškivača</title>
  </head>
  <body>
    <div id="outer">
      Click me!
      <div id="inner">
        No, click me!
      </div>
    </div>
    <button>Reset</button>
  </body>
</html>
```

2 - Target faza

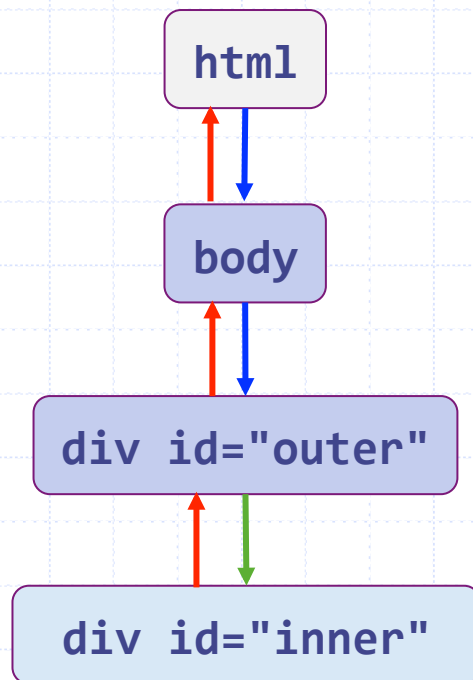
- ◆ Zatim pretraživač poziva osluškivač pridružen samom izvoru događaja (ukoliko postoji)



```
<html>
  <head>
    <meta charset="utf-8">
    <title>JS događaji: Dva osluškivača</title>
  </head>
  <body>
    <div id="outer">
      Click me!
      <div id="inner">
        No, click me!
      </div>
    </div>
    <button>Reset</button>
  </body>
</html>
```


3 - Bubble faza

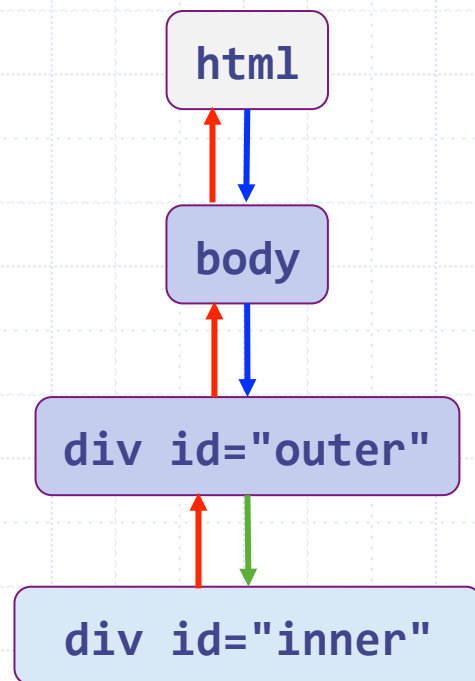
- ◆ Pretraživač se vraća povratnim putem i poziva sve osluškivače koji nisu kreirani kao *capture*, tj. podrazumevano su *bubble*



```
<html>
  <head>
    <meta charset="utf-8">
    <title>JS događaji: Dva osluškivača</title>
  </head>
  <body>
    <div id="outer">
      Click me!
      <div id="inner">
        No, click me!
      </div>
    </div>
    <button>Reset</button>
  </body>
</html>
```

stopPropagation()

- ◆ Zaustavlja putovanje događaja u lancu nezavisno od toga u kojoj se fazi nalazi !



```
<html>
  <head>
    <meta charset="utf-8">
    <title>JS događaji: Dva osluškivača</title>
  </head>
  <body>
    <div id="outer">
      Click me!
      <div id="inner">
        No, click me!
      </div>
    </div>
    <button>Reset</button>
  </body>
</html>
```

Konkurentnost događaja

- ◆ JS događaji se obrađuju sekvencijalno i procesiraju po jedan u datom trenutku
 - Ne postoji višenitna obrada događaja
- ◆ Obrada događaja se odvija kao nezavisan tok procesa
 - Ne meša se sa drugim izvršavanjima
- ◆ Funkcije obrađivača događaja zahtevaju pažljiv dizajn
- ◆ Obrada se mora završiti relativno brzo ($<10s$)
 - U suprotnom može doći do zastoja
- ◆ Suština kontrole je kroz broj događaja i pravilnu distribuciju obrade: obezbediti dovoljan broj obrađivača koji će za relativno kratko vreme izvršiti potrebne obrade

JavaScript klase

```
class ClassName {  
  constructor(params)  
  {  
    this.someField =  
      someParam;  
  }  
  
  methodOne() {  
    this.methodTwo();  
    const someValue =  
      this._someField;  
  }  
  
  methodTwo() {  
    ...  
  }  
}
```

Ne koristi se ključna reč
function

Sve metode su *public* (za sada)

Unutar klase, morate uvek
pozivati druge metode sa **this**

Unutar klase, poljima se mora
uvek pristupati sa **this**

this u konstruktoru

```
class Point {  
    constructor(x, y)  
{  
        this.x = x;  
        this.y = y;  
    }  
}
```

U konstruktoru klase,
this ukazuje na objekat
koji se kreira

this u funkciji

```
class Point {  
    ...  
  
    toString() {  
        return this.x + ", " + this.y;  
    }  
}
```

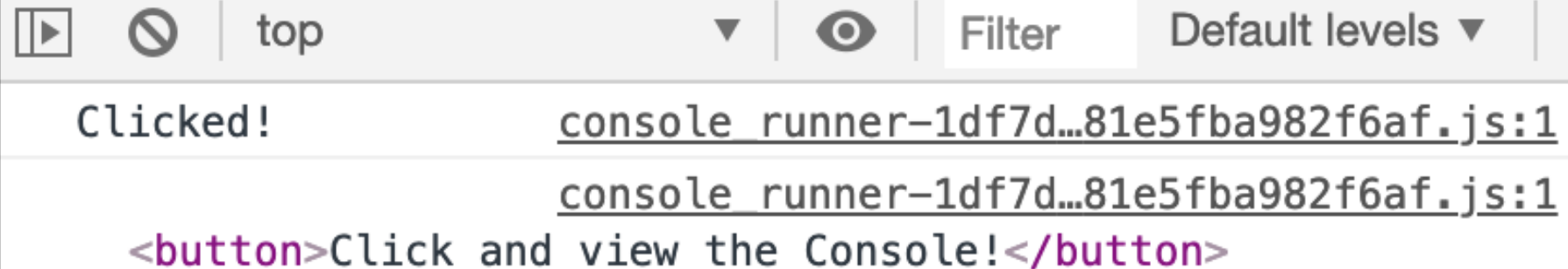
this može imati **različita značenja** u zavisnosti od toga kako se funkcija poziva, tj. u kom **kontekstu**
Svaka funkcija (uključujući i one koje se definišu van klase) sadrži this kao atribut
Njegova vrednost zavisi od toga kako se funkcija poziva !

this u obrađivaču događaja

```
function onClick() {  
  console.log('Clicked!');  
  console.log(this);  
}
```

```
const button = document.querySelector('button');  
button.addEventListener('click', onClick);
```

Kada se koristi u obrađivaču događaja, this pokazuje na **element kojem je osluškivač pridružen**



Clicked! console_runner-1df7d...81e5fba982f6af.js:1
console_runner-1df7d...81e5fba982f6af.js:1
<button>Click and view the Console!</button>

```
function onClick() {  
  console.log('Clicked!');  
  console.log(this);  
}  
const button =  
document.querySelector('button');  
button.addEventListener('click', onClick);
```

U onClick funkciji, this ukazuje na <button> zbog toga što se funkcija poziva u addEventListener.

this u funkciji - primer

```
function sayHello() {  
    console.log(this.name + ' says hello');  
}  
  
var person = {  
    name: 'Person',  
    hobbies: ['knitting', 'cooking', 'dancing'],  
    greeting: sayHello  
};  
person.greeting();
```

Kada koristimo this u funkciji koja se ne poziva iz obrađivača događaja, **this ukazuje na objekat iz kog se (ili nad kojim) se funkcija poziva**

this u funkciji

```
function sayHello() {  
    console.log(this.name + ' says hello');  
}  
  
var person = {  
    name: 'Person',  
    hobbies: ['knitting', 'cooking',  
    'dancing'],  
    greeting: sayHello  
};  
  
person.greeting();
```

"Person says hello"

Klasični programski jezici

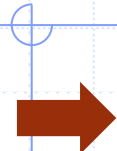
- ◆ Kod:
 - ◆ Lista instrukcija koje računar izvršava
 - ◆ Svaka linija koda je naredba
- ◆ Funkcija:
 - ◆ Grupa naredbi označena labelom
 - ◆ Sve naredbe se izvršavaju prilikom poziva funkcije
- ◆ Varijabla:
 - ◆ Podaci oznanačeni labelom

U JS funkcije predstavljaju i klasične funkcije i varijable !

```
const x = 15;  
let y = true;
```

```
const greeting = function() {  
    console.log('hello, world');  
}
```

Funkcija u JavaScript-u je objekat tipa Function



```
const x = 15;  
let y = true;
```

```
const greeting = function() {  
  console.log('hello, world');  
}
```

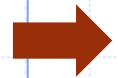
Funkcija u JavaScript-u je objekat tipa Function

memorija JS interpretera:

x

15

```
const x = 15;
```



```
let y = true;
```

```
const greeting = function() {  
  console.log('hello, world');  
}
```

Funkcija u JavaScript-u je objekat tipa Function

memorija JS interpretera:

x	15
y	true

```
const x = 15;  
let y = true;
```

```
➔ const greeting = function() {  
    console.log('hello, world');  
}
```

Funkcija u JavaScript-u je objekat tipa Function

memorija JS interpretera:

```
const x = 15;  
let y = true;
```

```
const greeting = function() {  
  console.log('hello, world');  
}
```

x

15

y

true

greeting

...



Funkcija u JavaScript-u je objekat tipa Function:

Kada se definiše funkcija, kreira se objekat tipa Function čiji je sadržaj kod funkcije

Atributi Function objekta

```
const greeting = function() {  
  console.log('hello, world');  
};  
  
console.log(greeting.name);  
console.log(greeting.toString());  
greeting.call();
```

Kada se definiše funkcija,
kreira se objekat tipa
Function koji sadrži attribute
kao što su :

name

toString()

Function objekat
takođe ima `call()`
metodu koja poziva
izvršni kod funkcije

Sintaksa `()` u pozadini
poziva ovu metodu !

"greeting"

"function () {
 console.log('hello, world');
}"

"hello, world"

Funkcije su specifičan tip objekta

- ◆ Jedini tip koji ima pridružen izvršni kod, odnosno objekat koji se može izvršavati
- ◆ Obični objekti se ne mogu izvršavati
- ◆ Ali mogu imati attribute čija je vrednost objekat tipa Function (tj. funkcije) koji se mogu izvršavati

```
function sayHello() {  
  console.log(this.name + ' says hello');  
}  
  
var person = {  
  name: 'Person',  
  hobbies: ['knitting', 'cooking', 'dancing'],  
  greeting: sayHello  
};  
person.greeting();
```

greeting
atribut je
objekat tipa
Function (tj.
sayHello
funkcije)

"Person says hello"

Razlog zbog kog su JS funkcije objekti

- ◆ Mogu se prosleđivati kao parametri drugih funkcija
- ◆ Poznate kao **callback funkcije**
- ◆ Zbog toga što kada definišemo funkciju u kodu u pozadini se kreira objekat tipa Function
- ◆ Programiranje zasnovano na obradi događaja
- ◆ Angular
- ◆ NodeJS

```
function onDragStart(event) {  
    ...  
}  
dragObject.addEventListener('pointerdown', onDragStart);
```

JS funkcije kao objekti prvog reda

- ◆ Mogu se prosleđivati kao parametri
- ◆ Mogu biti vrednosti varijabli
- ◆ Mogu biti definisane bez imena/identifikatora
 - Anonimne funkcije
 - Arrow funkcije

Načini definisanja funkcije

```
function myFunction(params) {  
}
```

```
const myFunction = function(params) {  
};
```

```
const myFunction = (params) => {  
};
```

Pozivaju se na identičan način, bez obzira kako se definišu:

```
myFunction();
```

Načini definisanja funkcije - primer

```
function imeFunkcije(pozvanaFunkcija) {  
    pozvanaFunkcija();  
}
```

```
var varijablaFunkcije = function() {  
    console.log('funkcija kao varijabla');  
};
```

```
var arrowFunkcija = () => {  
    console.log('arrow funkcija');  
};
```

```
imeFunkcije(varijablaFunkcije); "funkcija kao varijabla"  
imeFunkcije(arrowFunkcija);
```

```
"arrow funkcija"
```

Callback funkcija

- ◆ Funkcija koja se prosleđuje kao parametar drugoj funkciji
- ◆ Logično sa obzirom da je funkcija objekat prvog reda u JS (instanca tipa Object)
- ◆ Mehanizam koji često koriste JS okviri za razvoj klijentskih Web aplikacija
- ◆ Angular
- ◆ NodeJS
- ◆ Express
- ◆ React
- ◆ ...

Anonimna funkcija kao callback

```
var fruits = ["apple", "bannana", "orange"];  
fruits.forEach(function(fruit, index){  
    console.log(index + "." + fruit);  
});
```

"0.apple"

"1.bannana"

"2.orange"

Imenovana funkcija kao callback

```
function namedCallback(){  
    alert("namedCallback()");  
}  
function testFunction(callback){  
    callback();  
}  
testFunction(namedCallback);
```

An embedded page at s.codepen.io says

namedCallback()

OK

Parametri callback funkcije

```
var author = "Caller";  
function namedCallback(param){  
    alert("namedCallback() called by " + param);  
}  
function testFunction(callback){  
    callback();  
}  
testFunction(namedCallback(author));
```

An embedded page at s.codepen.io says
namedCallback() called by Caller

OK

Callback + parametar kao promenjiva

U konkretnom primeru, callback funkcija je opcionalna kao parametar (može se i ne mora prosleđivati)

```
var message = "hello there!";  
function myCallbackFunction(){  
    alert("myCallbackFunction is called()");  
}  
function testCallback(param, callback){  
    alert(param);  
    if(callback && typeof(callback) === "function"){  
        callback();  
    }  
}  
testCallback(message, myCallbackFunction);
```

Resultat?

An embedded page at s.codepen.io says
hello there!

1.

OK

An embedded page at s.codepen.io says
myCallbackFunction is called()

2.

OK

Linkovi ka primerima

Tipovi događaja:

<https://developer.mozilla.org/en-US/docs/Web/Events>

Ispisivanje pozicije miša:

<https://codepen.io/mljovanovic/pen/ZPmzNd>

Submit događaj: <https://codepen.io/mljovanovic/pen/aMQWdX>

JS događaji bubble: <https://codepen.io/mljovanovic/pen/NJeKya>

JS događaji bubble stopPropagation:

<https://codepen.io/mljovanovic/pen/JzwjWK>

JS događaji capture: <https://codepen.io/mljovanovic/pen/MxZYEJ>

JS događaji capture stopPropagation:

<https://codepen.io/mljovanovic/pen/OqrNOW>

Načini definisanja funkcije:

<https://codepen.io/mljovanovic/pen/WmLJXb>

Callback funkcije: <https://codepen.io/mljovanovic/pen/pYqZVB>

Function objekat: <https://codepen.io/mljovanovic/pen/RdEqdQ>

this u obrađivaču događaja:

<https://codepen.io/mljovanovic/pen/PLXrye>

this u funkciji: <https://codepen.io/mljovanovic/pen/KEENem>

Sadržaj

◆ JS Događaji:

- ◆ Definicija

- ◆ Vrste

- ◆ Upravljanje događajima: oslušivači i kretanje (propagacija)

- ◆ **this:**

- ◆ Korišćenje u funkcijama

- ◆ **Funkcije kao objekti prvog reda:**

- ◆ Načini definisanja i poziva

- ◆ Callback funkcije

- ◆ Primeri: codepen.io

JavaScript Događaji Funkcije (callback funkcije) this

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

U pripremi prezentacije korišćene su ilustracije i primeri sa
<https://developer.mozilla.org>