Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/05BE02Tomic

CLOSURES IN JAVASCRIPT – ДЕТАЉНО ОБЈАШЊЕЊЕ И ПРИМЕНА

CLOSURES IN JAVASCRIPT – DETAILED EXPLANATION AND USAGE

Стефана Томић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *У овом раду је представљена анализа затварања у JavaScript језику. Кроз примере је представљен JavaScript језик и могућност затварања као и објектни модел ДОМ са својим карактеристикама.*

Кључне речи: *Closures, JavaScript, ДОМ*

Abstract – *In this paper the analysis of closing in JavaScript language is presented. JavaScript language and possiblity of closing is presented through examples as well as DOM object with its characteristics.*

Keywords: *Closures, JavaScript, DOM*

1. УВОД

Примена затварања у JavaScript језику је широка. Од најједноставнијих примера скрипти које захтевају комуникацију корисника са одређеном HTML страном, до сложених примера употребе објеката у различитим наредбама [1]. Скрипт језици нису тако комплексни као уобичајени програмски језици, па нису подесни за писање комплекних програма. Углавном се користе када је потребно направити једноставне рутине којим се проверавају подаци на клијентској страни или када је потребно манипулисати елементима web странице. Ако је реч о сложенијим захтевима, онда се користи техника угњеждених објеката. У случају JavaScript-а код скрипта се или умеће директно у HTML документ или се из тог документа референцира неки JavaScript. У првом делу рада представљен је опис решаваног проблема где је описана употреба JavaScript-а на HTML странама, манипулација подацима у документима, употреба функције као и гранања. Описане су методе, приступ одређеним пољима, објектима као и prompt наредба. Такође описан је процес покретања функција у JavaScript језику који се користи у затварању.

2. ОПИС РЕШАВАНОГ ПРОБЛЕМА

Closure представљају функције које имају приступ променљивим вредностима које спадају у другу функцију скрипте. Овакав приступ је могуће извршити тако што се функција уграђује у другу функцију и на тај начин се изврши дефинисање ланца опсега.

Након тога затварање се може имплементирати и запамтити референцу из домена родитеља (из функције у којој има приступ променљивој). Такође променљиве које се користе у спољној функцији морају бити чуване у меморији како би биле доступне функцији затварања до краја процеса извршавања. Овакав приступ често може додатно успорити извршавање скрипте сходно превеликој употреби локалне меморије. Константним напредком JavaScript језика омогућено је побољшање употребе затварања. Битни фактори су да затварање памти промењљиве из спољне функције иако је спољна функција затворена. Затварање памти референцу на коју је референцирана промењљива функције. Такође је могуће извршавање редекларисање промењљиве.
Истраживања из ове области показују да је сталним напредком script језика омогућена додатна употреба затварања и примена у решавању различитих комплексних задатака. Представљени су показни примери основних JavaScript функција на које је могуће применити затварање.

2.1. Употреба JavaScript основне функције у HTML-у

Функција се позива навођењем имена функције и прослеђивањем стварних параметара у заградама. Често је потребно да неку функционалност која се понавља у програму издвојимо као посебну целину која се може користити из остатка програма. Параметри функције су варијабле које се у функцију прослеђују приликом позивања функције и могу се користити у телу функције као обичне варијабле.

2.2. Уписивање текста у документ

InnerHtml property се користи за постављање садржаја HTML елемента. У следећем примеру Слика 1. JavaScript функција се ставља у делу HTML тага и извршава се када се кликне на дугме.

```
    <script type="text/javascript">
        function izmenaTeksta()
        {
            document.getElementById('boldovano').innerHTML = 'Primer za master
rad';
        }
    </script>
</head>
<body>
    <p>
        FTN:<b id='boldovano'>br.2</b>
        </p>
        <input type='button' onclick='izmenaTeksta()' value='Klikni i izmeni
tekst' />
</body>
```

Слика 1. *Уписивање текста у документ*

НАПОМЕНА:
Овај рад проистекао је из рада чији ментор је био др Александар Купусинац, ван.проф.

2.2.1. Измена текста у документу

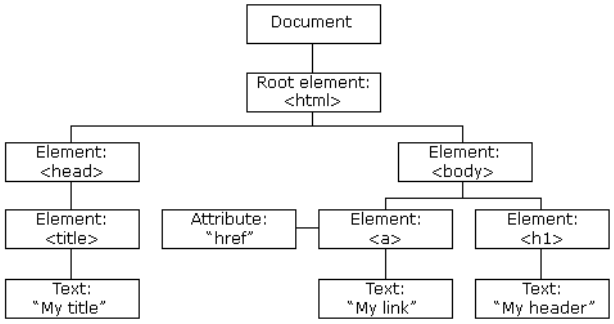
Предхони пример ће бити измењен тако што се мења текст који се исписује на почетку, текст дугмета као и текст након клика на дугме. Слика 2. представља другачији пример измене дугмета као и текст који мења почетни текст након клика на дугме.

```
    <script>
        function primerFunkcije() {
            document.getElementById("1111").innerHTML = "Tekst koji će se ispisati nakon klika
na dugme";
        }
    </script>
</head>
<body>
    <h1>Master rad - Primer</h1>
    <p id="1111">
        Tekst koji se ispisuje prilikom pokretanje stranice, a koji će biti zamenjen nakon
klika na dugme
    </p>
    <button type="button" onclick="primerFunkcije()">Probaj</button>
</body>
```

Слика 2. *Измена текста у документу*

2.3. Објектни модел HTML документа – DOM

Када се парсира HTML, он креира објектни модел HTML документа. Ово значи да он прави објекте који одговарају сваком HTML документу, атрибуту итд. Коришћењем ових објеката могуће је приступити вредностима HTML елемената. document.getElementById() је метода на објекту document која враћа референцу на објекат са датим идентификатором. Поља неких DOM објеката је могуће поставити на вредност неке функције што се генерално користи за реаговање на акцију корисника, тј. event handling. Начин на који се приступа и модификује садржај документа се назива Document Object Model или скраћено DOM. Објекти су организовани хијерархијски. Ова хијерархијска струкрура се односи на организацију објеката у web документу. Слика 3. представља једноставну хијерархију са DOM објектима.



Слика 3: *Једноставна хијерархија DOM модела*

• Window objekat− на врху хијерархије. То је спољни елемент објекта хијерархије.

• Document objekat− Сваки HTML документ који се прикаже у прозору читача постаје објекат Document. Document садржи садржај странице.

• Form objekat − Све што се налази између тагова <form>...</form> представља овај објекат.

• Контролни елементи форме (eng. Form control elements) − објекти форме садрже све елементе који су дефинисани у објекту као што су текст, поље за текст (text field), дугме (button), радио дугме (radio button) и бокс за штиклирање (checkbox).

• Постоји неколико модела ДОМ-а:

• Legacy DOM− Ово је модел који се појавио током раних верзија JavaScript језика, тако да је добро подржан од стране свих претраживача, али дозвољава приступ само одређеним кључним деловима докумената, као што су форме, елементи форме и слике.

• W3C DOM − Овај модел документ објекта омогућава приступ и модификацију свих садржаја документа и стандардизован је од стране World Wide Veb Consortium (W3C). Овај модел је подржан од стране скоро свих модерних веб читача.

• IE4 DOM - Уведен је у верзији 4 Microsoft-ovog Internet Explorer-a. IE 5 и новије верзије укључујући подршку за већину основних W3C DOM функција.

3. ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА И АЛАТА

JavaScript је платформски неутралан језик, као и HTML, што значи да би његов код (ако је писан по стандарду) требало да се извршава у оквиру читача клијента, без обзира која је хардверска машина или софтверско окружење у питању. Величина програма писаних у овом језику довољно је мала да може да се извршава и на машинама са лошијим перфомансама. Још једна од предности JavaScript језика је његова интегрисаност са HTML-ом. У оквиру једне странице могуће је на произвољан начин комбиновати JavaScript и HTML код. Такође из JavaScript-а могуће је генерисати сам HTML код, у зависности од одређене акције корисника. Програмски код овог језика се може укључити у оквиру HTML странице: директним писањем кода у оквиру странице. <SCRIPT LANGUAGE="JavaScript"> ...neki JavaScript kod… </SCRIPT> Није неопходно да се наводи атрибут language= “JavaScript”, јер он има подразумевану вредност JavaScript.

3.1 JavaScript објекти

Постоји велики број већ дефинисаних објеката (document, window итд.) које конструише сам интерпретер, а програмер скрипте може дефинисати додатне објекте.
Објекти у JavaScript-у се дефинишу на следећи начин:
{ime_polja1:vrednost_polja2,
ime_polja2:vrednost_polja2}
На пример, следећи објекат би могли користити да представимо корисника веб сајта:{email: “proba@gmail.com”, ime: “Milan”, prezime: “Markovic”, indeks: 1111}.

3.1.1 Методе и приступ пољима објекта

За приступање пољима објекта користи се оператор „.“ (тачка).Током извршавања програма могуће је додавати нова поља објектима, читати и мењати њихову вредност. Пољима је могуће приступити и користећи синтаксу индексирања у низ. Следећи пример демонстрира приступ пољима објекта:

var o = {} //празан објекат
JavaScript третира функције као грађане првог реда, што значи да је функције могуће третирати као било коју другу вредност: доделити их као вредност промењивој, прослеђивати у функције као параметар, враћати из функција као резултат итд.

3.1.2 Приступ пољима објекта из метода

Методе имају својство да припадају неком објекту. Као такве, могу приступити пољима тог објекта. За приступ пољима објекта којем метода припада, користи се специјална референца објекта this. У следећем примеру, користи се референца this за приступ пољима name и lastName.

```
var student = {name: "Dejan",
lastName: "Maksimovic",
sayHello: function() {
console.log(this.name);
console.log(this.lastName); }
};
student.sayHello();
```

Други начин укључивања JavaScript-и је позив js документа. У оквиру тага се дефинише спољашњи документ у оквиру атрибута src. Спољашњи JavaScript код – омогућује коришћење/позивање исте скрипте у више HTML докумената <script type="text/javascript" src="xxx.js"></script>.
Није неопходно декларисати промењљиву пре прве доделе вредности (аутоматски ће се извршити декларисање), али је и предекларисање дозвољено. Називи промењљивих - имена промењљивих могу да садрже бројеве и слова енглеске абецеде, али први знак мора да буде слово енглеске абецеде или симбол "_" , поред тога:

- Не могу се користити празна места у оквиру имена
- Не могу се користити резервисане речи као имена промењљивих

Промењљива или варијабла - описно речено је меморијска локација, "спремник" за одређену информацију/податак или вредност. Користи се за декларисање промењљиве (декларација је креирање промењљиве, а дефиниција значи и иницијализацију - постављање почетне вредности), чија синтакса је: var imePromenljive; Опционо могуће је извршити и њену иницијализацију.

3.1.3 Prompt наредба

Будући да JavaScript не обезбеђује једноставан метод за прихватање корисничких података, за ове потребе, користе се HTML форме и prompt диалог прозори. Prompt диалог прозор се појављује са једноставним пољем за текст (textfield). Након што корисник у prompt диалог унесе текст, вредност се враћа. Prompt dialog box користи два аргумента: string, тј. текст који обично представља питање постављено кориснику опомињући корисника да одговори на питање и још један string који обично представља default одговор који је написан у пољу за унос. Ако корисник притисне OK дугме, цео текст из дијалог прозора се враћа. Ако притисне cancel прозор биће му враћено null.
Синтакса је: prompt("sometext","defaultvalue")

4. ОПИС РЕШЕЊА ПРОБЛЕМА - ПРИМЕНА ЗАТВАРАЊА (CLOSURES)

Већина савремених програмских језика користи lexical scoping што подразумева извршавање функција уз помоћ варијабилног опсега који је дефинисан на почетку извршења а не у току позивања. Имплементација лексичког опсега захтева да интерно стање објеката JavaScript функције укључујује код функције и референцу на тренутни ланац опсега. Комбиновање објеката и опсега (опсеге представља скуп променљивих) где су варијабле решене функције назива се затварање (closure) у компјутерској науци. Све JavaScript функције су затварања у случају да су објекти и да имају ланац опсега који је са њима повезан. Већина описаних функција се позива у истом ланцу обима који је коришћен када је функција дефинисана и у том случају није потребно затварање [4]. Најчешће се дешава да се угњеждени објекат одређене функције врати из функције у којој је дефинисан. Кроз низ програмерских техника које укључују овај тип угњеждених функција затварања, примена угњеждених техника постала је честа у програмирању у JavaScript језику [6].

4.1 Имплементација затварања

Процес затварања (closures) једноставно је разумети и применити ако се користи правило лексичког опсега: функције се извршавају коришћењем ланца опсега који је примењен на почетку процеса дефинисања. Програмери у току имплементације сматрају да су затварања збуњујућа али се кроз детаљан опис имплементационог тока и ти проблеми решавају. Локална варијабла дефинисана у спољној функцији престаје да постоји када се спољна функција врати док се угњеждена функција може извршити користећи тренутни ланац опсега. Ако су локалне промењљиве функције дефинисане у стеку процесора потребно је да престану да постоје у случају враћања функције [2].
Сваки пут када се јавља JavaScript функција врши се креирање новог објекта који обухвата локалне варијабле за тај позив а након тога се тај објекат додаје у ланац обима. Када се функција враћа, објекат који је везан за варијаблу се уклања из ланца оквира. У случају да не постоје угнеждене функције, референце на објекат везивања не постоје и таложи се део програмског кода који се извршава [3].Такође, уколико су угњеждене функције дефинисане, тада свака од тих функција има референцу на ланац обима а тај ланац обима се односи на објекат везивања варијабле. Ако су угњеждени функционални објекти унутар спољашње функције у том случају се враћа или чува у неком својству а за то својство везана је и спољна референца.
Closure је способност функције да запамти и приступи свом лексичком scope-у када се функција извршава изван свог лексичког scope-а.

4.1.1 Примена closure функције

Closures функције су нам најпотребније када користимо callback механизам и креирамо функције које обрађују догађаје. Слика 4 представља пример који се често користи у пракси:

```
<script type="text/javascript">
      function dodaj() {
           var blok = document.createElement("div");
           blok.innerHTML = "Kliknite na dugme.";
           document.body.appendChild(blok);

           var dugme = document.createElement("input");
           dugme.type = "button";
           dugme.value = "Dugme";
           dugme.onclick = klik;    // definišemo event handler
           document.body.appendChild(dugme);

           function klik() {
               blok.innerHTML = "Kliknuto!";
           }
       }
       //...
       dodaj();
   </script>
```

Слика 4: *Пример примене closure функције*

Функција dodaj() креира један DIV блок и једно INPUT дугме. Да би се променила порука у блоку, потребно је кликнути на дугме и зато му задајемо onClick својство као референцу на функцију klik().

Док корисник кликне на дугме, функција dodaj() је одавно завршена. Међутим, функција klik() која се тада позива, „одржава у животу“ све локалне променљиве њене надфункције и блок и дугме (иако јој дугме није потребно). Захваљујући томе, промениће текст у DIV блоку.

4.1.2. Closure памти променљиве из спољне функције чак иако је спољна функција извршена

Након коришћења - наредба return прекида извршење функције и постојање њених промењивих у привременој меморији.
Међутим „Closure“ памти промењиве из спољне функције чак иако је вратила неку вредност са return.

4.1.3. Closure памти референцу на коју указује променљива

Када се каже да „closure функција” има приступ променљивој под тим се мисли да има приступ одређеном месту у меморији на коју указује та промењива тј. њеној референци у тренутку позивања “closure функције“. Уколико се на том месту у меморији тј. референци мења вредност, closure ће увек узети најновију тренутну вредност. У наредном примеру се види како унутрашња функција “памти” референцу и користи тренутну вредност у меморији.

4.1.4. Редекларисање променљиве

Ако након дефинисања closure, промењивој која је потребна closure доделимо неку другу референцу (место у меморији које чува неку вредност), closure ће да користи референцу која је била у оптицају у тренутку дефинисања closure.

5. ЗАКЉУЧАК

Употребом затварања омогућено је:

- Додавање вредности променљивој коју враћа функција која се позива у скрипти
- Функција која се позива у скрипти извршава се само једном, постављањем бројача и враћањем функције у форми израза
- Променљива у том случају може приступити бројачу (counter) или надређеној функцији.

Функција у том случају има приватне промењиве и може приступити надређеном ланцу опсега и у случају да је надређена функција затворена.

Добра страна примене затварања је што се све може извршити на клијентској страни и знатно смањити коришћење меморије серверске стране.

Sever – side scripting је Веб сервер технологија која омогућава да се кориснички захтев испуњава помоћу скрипта који се извршава на серверској страни како би се генерисале динамичке HTML странице. Применом затварања могуће је унапредити скрипту и знатно убрзати време извршавања и декларисања променљивих. Обично се користи да би се интерактивне Web странице повезале са базама података, ради идентификације корисника, ажурирања садржаја, стварања разних дискусионих група и још због много тога. Разлика између скриптовања на серверској и корисничкој страни је у томе што се код корисничке стране скриптови извршавају у web читачу корисника, најчешће помоћу JavaScript - a.

Client – side scripting технологије се обично односе на групу рачунарских програма на вебу који се извршавају на страни клијента, тј. од стране web читача на клијентским рачунарима, уместо на страни сервера, од стране апликација web сервера.

6. ЛИТЕРАТУРА

[1] Computer Programming Skills & Maintenance, CHEN Yuanyi,ZHOU, JavaScript Closures Research & Typical Applications, 2011
[2] David Flanagan, JavaScript - The Definitive Guide, 6th Edition, 2011
[3] Daniel Jackson, functions, scope & closures
[4] Advanced JavaScript Essentials, O'Reilly School of Technology's (OST)[6] Douglas Crockford, The Good Parts, 2012
[5] Boris Fritscher, Zero to Hero JavaScript, 2012
[6] Douglas Crockford, The Good Parts, 2012

Кратка биографија:

Стефана Томић рођена је у Сомбору 1992. године. Школске 2011/2012. године уписала се на Факултет Техничких Наука у Новом Саду. Дипломирала је на основним студијама 2016. године. Исте године уписала је мастер Примењено софтверско инжењерство.