
JavaScript nasleđivanje

Autori:
Milan Segedinac
Goran Savić

1. Nasleđivanje

U klasičnim objektno-orijentisanim programskim jezicima (objektno-orijentisanim programskim jezicima koji se baziraju na klasama, kao što je Java) nasleđivanje je važna tema jer:

1. Specificira hijerarhiju tipova - smanjuje potrebu za eksplicitnim kastovanjem, obzirom da klasa naslednica može implicitno da bude kastovana u klasu predak
2. Omogućuje ponovno korišćenje koda - klasa naslednica treba samo da specificira po čemu se razlikuje od klase pretka

U programskom jeziku JavaScript podržan je *duck typing*¹ - stil dinamičkog tipiziranja u kome skup metoda i svojstava objekta određuje validnost operacija umesto da se za to koristi hijerarhija nasleđivanja. Stoga u programskom jeziku JavaScript ne postoji problem implicitnog kastovanja.

JavaScript je hibridni (multiparadigmatiski) programski jezik i omogućuje bogatije mehanizme ponovnog korišćenja koda od nasleđivanja. Međutim, kao što ćemo videti u ovoj lekciji, ti mehanizmi ponovnog korišćenja koda u potpunosti mogu da simuliraju nasleđivanje. U ovoj lekciji ćemo videti kako se nasleđivanje u JavaScript-u može realizovati na tri načina. U lekciji u kojoj će biti reči o ES6 specifikaciji videćemo još jedan način na koji se može ostvariti nasleđivanje.

Pseudoklasično nasleđivanje

U klasičnim objektnim jezicima objekti su instance klase, a klase mogu da nasleđuju jedne druge. U programskom jeziku JavaScript *objekti direktno nasleđuju jedni druge posredstvom prototipova*. Za kreiranje objekata mogu se koristiti konstruktorske funkcije. Setimo se da su to funkcije koje se pozivaju sa ključnom rečju `new` i da one vraćaju objekat za čiji prototip je postavljena vrednost koja je bila na svojstvu `prototype` te konstruktorske funkcije. Listingom ispod dat je primer konstruktorske funkcije.

```
var Osoba = function (spec) {  
    if(spec){  
        this.ime = spec.ime;  
        this.prezime = spec.prezime;  
    }  
};  
Osoba.prototype.predstaviSe = function () {  
    return "Ja se zovem " + this.ime + " " + this.prezime;  
};  
var markoMarkovic = new Osoba({ime:"Marko", prezime:"Markovic"});  
console.log(markoMarkovic.predstaviSe());
```

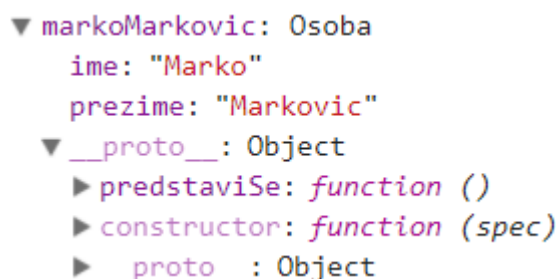
Listing - JavaScript konstruktorska funkcija

¹ Termin *duck typing* potiče od primera *Okamovog brijanja*, logičkog pravila po kome je najjednostavnije objašnjenje najčešće i tačno rešenje - „ako kvače i hoda kao patka, u pitanju je patka“.

JavaScript nasleđivanje

Varijabla `Osoba` je funkcija koja primi samo jedan parametar - `spec` - specifikaciju objekta koji se kreira. Podsetimo se da je, u konstrukorskim funkcijama, `this` objekat koji se kreira pri pozivu konstruktorske funkcije. U konstruktoru `Osoba`, ukoliko je prosleđen argument `spec`, u `this` objekat će na atribut `ime` biti postavljena vrednost `spec.ime`, a na `this.prezime` biće postavljen `spec.prezime`.

Nakon kreiranja konstruktorske funkcije u prototip dodajemo metode. Za primer konstruktora `Osoba` dodali smo samo jednu metodu: `predstaviSe` koja vrati string u kome je formatirano prikazano ime i prezime osobe. Nakon kreiranja objekta konstruktorskom funkcijom, objekat `markoMarkovic` imao bi attribute i metode prikazane slikom ispod.



```
▼ markoMarkovic: Osoba
  ime: "Marko"
  prezime: "Markovic"
  ▼ __proto__: Object
    ► predstaviSe: function ()
    ► constructor: function (spec)
    ► __proto__: Object
```

Slika - objekat kreiran pozivom konstruktorske funkcije

Ako bismo hteli da napravimo konstruktorsku funkciju koja kreira objekte koji predstavljaju studente, to znači da nam treba konstruktor koji kreira objekat koji ima sve što ima i `Osoba` (`ime` i `prezime`), a pored toga imaju još i `smer` i listu položenih ispita. Time bismo dobili da je `Student` naslednik `Osoba`.

Prvo moramo pozvati konstruktor pretka (`Osoba`) kojim će biti kreiran objekat koji ima `ime` i `prezime`. Svaki objekat ima referencu na konstruktor kojim je kreiran i to je atribut `constructor` prototipa. Ovu činjenicu možemo da iskoristimo za poziv konstruktora pretka, kao što je prikazano listingom ispod.

```
var Student = function(spec){
    this.constructor.apply(this, [spec]);
    this.smer = spec.smer;
    this.ocene = spec.ocene;
};
Student.prototype = new Osoba();
```

Listing - konstruktor `Student`

Prvo ćemo iskomentarisati poslednji red listinga: atribut `prototype` konstruktorske funkcije `Student` dobio je kao vrednost *objekat kreiran pozivom konstruktora* `Osoba` bez argumenata (setimo se da smo inače taj konstruktor pozivali šaljući kao argument specifikaciju sa imenom i prezimenom). Šta smo time dobili? Dobili smo da u prototipu `this` objekta u konstruktoru `Student` kao atribut `constructor` imamo konstruktorsku funkciju kojom je objekat kreiran, a to je upravo `Osoba`. Sada nam preostaje da u konstruktoru `Student` kairamo pretka (osobu) za zadatu specifikaciju (`ime` i `prezime`). To ne bismo mogli da izvršimo pozivom `this.constructor(spec)`, jer ovaj poziv ne bi imao

JavaScript nasleđivanje

pristup novokreiranom objektu. Konstruktor pretka moramo pozvati tako da `this` bude upravo objekat koji kreira konstruktorska funkcija `Student`. Zbog toga poziv izgleda kao u drugoj liniji koda listinga: `this.constructor.apply(this, [spec])`. Nakon toga, ostaje samo da se za `this` postave smer i ocene iz parametra `spec`.

Dodavanje nove metode prikazano je listingom ispod.

```
Student.prototype.getCV = function () {  
  
    return "studiram na smeru " +  
        this.smer +  
        " i imam sledece ocene" +  
        this.ocene;  
};
```

Listing - dodavanje metode potomka

Dodavanje nove metode u naslednicu svodi se na jednostavno dodavanje metode u prototip, kao što smo i do sada radili. Ali kako bi izgledalo redefinisanje metode? Primer redefinisanja metode `predstaviSe` dat je listingom ispod.

```
Student.prototype.predstaviSe = function () {  
  
    return "zovem se " + this.ime + " " +  
        this.prezime +  
        " i studiram na smeru "+this.smer;  
};
```

Listing - preklapanje metode

Obzirom da se metode i atributi uvek traže u bližim objektima u lancu prototipova, prosto dodavanje metode u prototip objekta preklapa metodu pretka sa istim nazivom.

Kreiranje naslednika dato je listingom ispod.

```
var peraPeric = new Student({ime:"Pera",  
  
    prezime:"Peric",  
    smer:"Racunarstvo i automatika",  
    ocene:[  
        {predmet:"Web programiranje",  
          ocena:9},  
        {predmet:"Operativni sistemi",  
          ocena:10}]]});
```

Listing - kreiranje naslednika

Prilikom kreiranja naslednika prosleđen je `spec` objekat koji sadrži sve attribute potrebne za konstrukciju pretka (`Osoba`) i naslednika (`Student`). Izgled kreiranog objekta prikazan je slikom ispod.

```
▼ peraPeric: Student
  ime: "Pera"
  ► ocene: Array(2)
  prezime: "Peric"
  smer: "Racunarstvo i automatika"
  ▼ __proto__: Osoba
    ► getCV: function ()
    ► predstaviSe: function ()
    ▼ __proto__: Object
      ► predstaviSe: function ()
      ► constructor: function (spec)
      ► __proto__: Object
```

Slika - objekat kreiran pozivom konstruktorske funkcije `Student`

Vidimo da kreirani objekat ima sve svoje atribute (`smer` i lista ocena) i sve atribute svog pretka (`ime` i `prezime`). Prototip kreiranog objekat ima metodu `getCV` i prektlopljenu metodu `predstaviSe`. Prototip prototipa ima metodu `predstaviSe` pretka.

Pseudoklasično nasleđivanje ima nekoliko ograničenja. Prvo, veoma ozbiljno, ograničenje je komplikovanost ovakvog pristupa i kontraintuitivnost - na primer, prvo je potrebno pozvati `constructor` metodu pretka pomoću `apply` metode, da bismo tek kasnije u kodu postavili pretka i to kao objekat kreiran pozivom konstruktora pretka bez parametara! Pored toga, ostalo je još dosta otvorenih pitanja. Neka od njih su:

- Kakva je razlika između konstruktora i obične funkcije?
- Šta će se desiti ako konstruktor pozovemo bez `new` ili ako običnu funkciju pozovemo sa `new`?
- Kako da definišemo privatna svojstva i metode?
- Kako da definišemo statička svojstva?
- Kako pozivamo super metode?

Zbog toga postoje i druge tehnike pomoću kojih se nasleđivanje može realizovati u programskom jeziku JavaScript.

Diferencijalno nasleđivanje

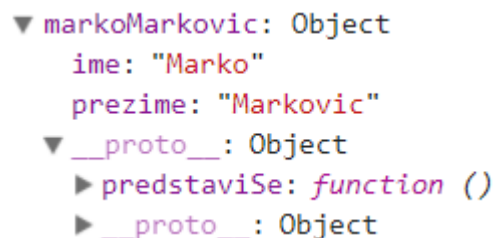
Komplikovanost pseudoklasičnog nasleđivanja prevashodno je bila posledica činjenice da je to programerski stil koji imitira korišćenje klasa u jeziku koji u kom ne postoje klase. Značajne konfuzije se mogu razrešiti naprosto tako što nećemo simulirati klase, nego ćemo kreirati osnovne objekte, a ostale objekte napraviti navodivši *po čemu se razlikuju* od tih, osnovnih, objekata. Primer takvog kreiranja jednog osnovnog objekata dat je listingom ispod.

JavaScript nasleđivanje

```
var osoba = {  
    predstaviSe: function() {  
        return "Ja se zovem " +  
            this.ime + " " + this.prezime;  
    }  
};  
var markoMarkovic = Object.create(osoba);  
markoMarkovic.ime = "Marko";  
markoMarkovic.prezime = "Markovic";
```

Listing - kreiranje osnovnog objekta

Kreirali smo objekat `osoba` koji ima metodu `predstaviSe`. Zatim smo kreirali objekat `markoMarkovic` koji kao prototip ima objekat `osoba`, što znači da će nad njim moći da se pozove metoda `predstaviSe`. Objektu `markoMarkovic` postavili smo atribute `ime` i `prezime`. Obratite pažnju da smo dobili istu funkcionalnost kao da smo kreirali konstruktorsku funkciju, samo je kod intuitivniji i čitljiviji. Kreirani objekat prikazan je listingom ispod.



```
▼ markoMarkovic: Object  
  ime: "Marko"  
  prezime: "Markovic"  
  ▼ __proto__: Object  
    ► predstaviSe: function ()  
    ► __proto__: Object
```

Slika - kreirani objekat

Kreirani objekat ima atribute `ime` i `prezime` i u prototipu ima metodu `predstaviSe`.

Kako bismo kreirali objekat „klase“ `student`? Bilo bi potrebno da se u prototipu nalazi objekat `osoba`, da bismo mogli da pozovemo metodu `predstaviSe`. Takođe, dodali bismo novu metodu `getCV`. Programski kod je dat listingom ispod.

```
var student = Object.create(osoba);  
  
student.getCV = function () {  
    return "studiram na smeru "+  
        this.smer+  
        " i imam sledece ocene "+  
        this.getOcene();  
}  
  
var peraPeric = Object.create(student);  
peraPeric.ime = "Pera";  
peraPeric.prezime = "Peric";  
peraPeric.smer = "Racunarstvo i automatika";  
peraPeric.ocene = [{predmet:"Web programiranje",  
    ocena:9},  
    {predmet:"Operativni sistemi",  
    ocena:10}];
```

Listing - nasleđivanje

JavaScript nasleđivanje

Objekat `student` kreiran je pozivom `Object.create(osoba)`, čime je kao prototip tog objekta postavljen objekat `osoba`. Obratite pažnju da se radi o običnom instanciranju objekta. Zatim je u objekat `student` dodata nova metoda - `getCV`. Instanca „klase“ `student` - `peraPeric` - kreirana je tako što je objekat `student` postavljen prototip tog objekta, ponovo pozivom `Object.create`. Zatim su postavljeni atributi `ime`, `prezime`, `smer` i `ocene`. Kreirani objekat prikazan je listingom slikom ispod.

```
▼ peraPeric: Object
  ime: "Pera"
  ► ocene: Array(2)
  prezime: "Peric"
  smer: "Racunarstvo i automatika"
  ▼ __proto__: Object
    ► getCV: function ()
    ▼ __proto__: Object
      ► predstaviSe: function ()
      ► __proto__: Object
```

Slika - diferencijalno nasleđivanje

Diferencijalnim nasleđivanjem rešili smo neke od problema pseudoklasičnog nasleđivanja, pre svega komplikovanost ovog pristupa. Međutim još nismo dali odgovor na pitanje kako bismo mogli da realizujemo privatna svojstva.

Funkcionalno nasleđivanje

Kreiranje objekata u funkcionalnom nasleđivanju može se u grubo opisati sledećim šablonom:

1. Funkcija kreira objekat - bilo kao literal, pozivom `Object.create` ili pozivom konstruktorske funkcije.
2. Javni atributi i metode su atributi i metode kreiranog objekta
3. Privatni atributi i privatne metode su *varijable funkcije* - obzirom da JavaScript podržava *closure* (funkcija ima pristup svom spoljašnjem kontekstu i nakon završetka izvršavanja spoljašnje funkcije), kreirani objekat će imati pristup atributima i metodama funkcije koja ga je kreirala. Činjenica da se funkcija koja je kreirala objekat završila garantuje nam da *niko drugi* neće imati pristup tim atributima i metodama.
4. Funkcija vrati kreirani objekat

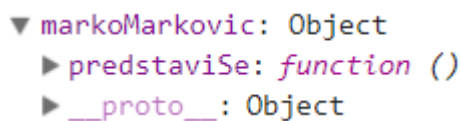
Ovaj šablon se u programskom jeziku JavaScript naziva još i *modul* - objekat koji ima svoj javni i privatni deo.

Primer ovakvog kreiranja objekata dat je listingom ispod.

```
var osoba = function(spec) {  
    var that = {};  
    that.predstaviSe = function(){  
        return "Ja se zovem " +  
            spec.ime + " " +  
            spec.prezime;  
    };  
    return that;  
}  
var markoMarkovic = osoba({  
    ime: "Marko",  
    prezime: "Markovic"  
});
```

Listing - modul

Navedeni listing zahteva objašnjenje. Kreirali smo funkciju `osoba` koja kao parametar prima `spec`, specifikaciju sa imenom i prezimenom. U ovoj funkciji kreirali smo objekat `that` koji će biti povratna vrednost funkcije. Objekat `that` ima jednu metodu - `predstaviSe`. U ovoj metodi vraćamo formatiran ispis imena i prezimena. Obratite pažnju da metoda pristupa atributima parametra `spec` (`spec.ime` i `spec.prezime`). Posetimo se da se u programskom jeziku JavaScript parametri tretiraju identično kao lokalne varijable funkcije. To nam garantuje da će `spec.ime` i `spec.prezime` biti privatni atributi objekta koji vrati funkcija `osoba`. Zatim je kreiran objekat pozivom funkcije `osoba`. Ovaj objekat prikazan je na slici ispod.



```
▼ markoMarkovic: Object  
  ► predstaviSe: function ()  
  ► __proto__: Object
```

Slika - objekat kreiran funkcijom `osoba`

Kao što vidimo na slici iznad, u objektu kreiranom pozivom funkcije `osoba` dostupna je jedino metoda `predstaviSe`. Ime i prezime su privatni i nisu vidljivi spolja.

Ovakav pristup kreiranju objekata dovoljno je fleksibilan da podrži brojne mogućnosti klasičnog objektno-orientisanog programiranja. Primeri su dati listingom ispod.

JavaScript nasleđivanje

```
var student = function(spec) {

    var that = osoba(spec); //konstruktor pretka
    var studira = true; // privatni atribut,
                                // zbog closure je vidljiv samo u
funkciji
    that.smer = spec.smer; // javni atribut that objekta koji
                                //ce biti vraćen iz funkcije
    that.super_predstaviSe = that.predstaviSe; // super
                                //metoda koje ćemo koristiti
    student.brojac = student.hasOwnProperty('brojac')?(student.brojac
+ 1):1; //statičko polje
    var brojIndeksa = student.brojac; // broj indeksa je
                                //autoinkrement
    that.getIndeks = function () { //javna metoda
        return brojIndeksa;
    }
    that.getCV = function() { //javna metoda
        return "Moj broj indeksa je " + that.getIndeks() + "
studiram na smeru " + spec.smer +
        " i imam sledece ocene " + getOcene();
    };

    that.predstaviSe = function () {
        return that.super_predstaviSe() + " i " + (studira?"jos
uvek studiram":"vise ne studiram");
    };
    var getOcene = function() { //privatne metode
        var i;
        var retVal = "";
        for (i in spec.ocene) {
            retVal += spec.ocene[i].predmet + ":" +
spec.ocene[i].ocena + " ";
        }
        return retVal;
    };
    that.upisiOcenu = function(ocena) { //javna metoda koja
menja // privatni atribut
        spec.ocene.push(ocena);
    };

    return that;
};
```

Listing - neke od mogućnosti funkcionalnog nasleđivanja

Kreirali smo funkciju `student` koja kao parametar prima specifikaciju objekta `spec`. Ova funkcija kreira objekat `that` koji će biti vraćen iz funkcije pozivom funkcije `osoba` za argument `spec`. Time smo postigli da `student` nasleđuje `osoba`.

Atribut `studira` je lokalna varijabla funkcije `student`, što znači da će biti privatni atribut vraćenog objekta. Objekat sam će moći da mu pristupi, ali ovom atributu neće moći da se pristupi od spolja.

Atribut `smer` je javni, pa je postavljen kao atribut objekta `that` koji će biti vraćen iz funkcije.

JavaScript nasleđivanje

Kako bismo od spolja mogli da pristupimo metodi `predstaviSe` pretka? To je ilustrovano postavljanjem atributa `super_predstaviSe` pre nego što je u `that` postavljena preklopljena metoda.

Statičko polje ilustrovano je brojačem pomoću kog se postavlja broj indeksa. Naime, `brojac` je atribut *funkcije* `student`, što znači da će biti zajednički za sve objekte koje kreira ova funkcija.

Broj indeksa je privatno polje koje dobije vrednost koju trenutku kreiranja ima `brojac`.

Javne metode ilustrovane su sa `getIndeks` i `getCV`. Za ove metode kažemo da su privilegovane jer imaju pristup privatnim atributima.

Metodu `predstaviSe` smo preklopili u funkciji `student` prosto tako što smo u objekat `that` postavili novu metodu `predstaviSe`.

Formatiran prikaz ocena realizovan je privatnom metodom `getOcene`. Ovoj metodi nije moguće pristupiti od spolja.

Metoda `upisiOcenu` je javna privilegovana metoda koja menja privatno listu `ocene`.

Obratite pažnju da smo sve navedene funkcionalnosti realizovali *bez korišćenja prototipa*! Ovaj primer nam je ilustrovao da programski jezik JavaScript nikako nije oskudniji i siromašniji od programskih jezika poput Jave. Naravno, ovo je samo mali podskup funkcionalnosti i šablona koji se koriste u programskom jeziku JavaScript. U narednim lekcijama susrešćemo se sa još takvih primera.