Obrada izuzetaka

U prethodnim lekcijama ovog modula predstavljeni su različiti načini za kontrolu toka izvršavanja koda. U ovoj lekciji biće opisan još jedan pristup za postizanje uslovnog izvršavanja JavaScript koda. Reč je o pristupu koji se koristi za obradu izuzetaka koji nastaju usled pojave grešaka prilikom izvršavanja JavaScript koda. Stoga će u ovoj lekciji biti predstavljeno nekoliko novih pojmova, koji su veoma značajni za JavaScript programiranje. Na samom početku neophodno je upoznati se sa pojmom grešaka do kojih može doći prilikom izvršavanja programskog koda.

Softverske greške

Tokom izvršavanja programskog koda bilo kojeg jezika može se pojaviti greška. Greške su sastavni deo svakog programa. I pored velikih napora koji se ulažu u njihovo detektovanje i uklanjanje, veoma je teško napraviti ozbiljniju aplikaciju bez koda koji bi u nekom trenutku mogao da izazove grešku.

Greške prilikom izvršavanja koda mogu biti izazvane brojnim faktorima. Neke od njih se mogu pripisati manjkavo napisanom kodu, dok druge svoj izvor imaju u spoljnom okruženju. Spoljni faktori koji mogu da izazovu pojavu grešaka prilikom izvršavanja koda mogu da budu: nedostatak interne ili eksterne memorije, prekid mrežne konekcije, nekorektan rad nekog od uređaja i drugo. S obzirom na to da se JavaScript jezik izvršava unutar web pregledača ili nekog drugog softverskog proizvoda koji poseduje JavaScript izvršno okruženje, greškama koje su izazvane spoljnim faktorima uglavnom se rukuje automatski, bez potrebe za angažovanjem frontend programera.

Ipak, za frontend programere mnogo su bitniji interni faktori koji mogu da dovedu do pojave grešaka. Na primer, sledeći JavaScript kôd sadrži grešku:

```
console.log("Value of variable b is : " + b );
```

Prikazana naredba treba da unutar konzole ispiše vrednost promenljive `b`. Ipak, ukoliko promenljiva `b` nije prethodno deklarisana, navedena naredba proizvešće grešku:

```
ReferenceError: b is not defined
```

Ovo je samo jedan od primera koda koji u JavaScript jeziku može da proizvede grešku.

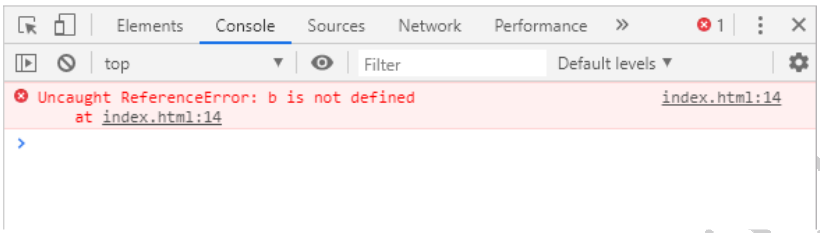
Kada u kodu dođe do ovakve situacije, izvršavanje JavaScripta se prekida, tako da će sav kôd koji se nalazi ispod onog koji je izazvao grešku ostati neizvršen. Da bismo to pokazali, prikazani primer ćemo proširiti na sledeći način:

```
console.log("Value of variable b is : " + b);
console.log("Hello World");
```

Sada je, nakon problematične, definisana još jedna, sintaksno i logički potpuno ispravna naredba. Ipak, ukoliko pokušate da izvršite ovakav kôd, moći ćete da vidite da se druga naredba uopšte ne izvršava, što potvrđuje nešto ranije iznetu tvrdnju da prilikom pojave greške dolazi do potpunog prekida izvršavanja skripte. Ipak, prekid izvršavanja koda nije jedino što se događa prilikom pojave greške, jer se u takvoj situaciji kreira i emituje specijalan tip objekta koji se naziva izuzetak.

Izuzetak (Exception)

Izuzetak predstavlja reprezentaciju greške koja se dogodila u kodu. To praktično znači da je greška uzrok, a izuzetak posledica. Tako je poruka sa opisom greške koja je dobijena u prethodnom primeru zapravo predstavljala izuzetak, emitovan od JavaScript izvršnog okruženja (slika 14.1).



Slika 14.1. Primer JavaScript izuzetka

U JavaScript jeziku izuzeci su objekti tipa `Error`, unutar kojih su objedinjene dve veoma značajne informacije o nastaloj grešci:

- `name` i
- `message`.

JavaScript poznaje razne specijalizovane vrste `Error` objekata koji obezbeđuju detaljnije informacije o različitim tipovima grešaka. Tako je u primeru sa slike 14.1. izuzetak reprezentovan objektom tipa `ReferenceError`.

Izuzeci su integralni sastojak sistema za rukovanje greškama, a njihova osnovna svrha je da pruže informaciju o grešci koja se dogodila u kodu. Ipak, pored informacije, izuzeci poseduju još jednu veoma značajnu osobinu. Njihova svrha je da budu obrađeni, a takav postupak se naziva obrada izuzetaka.

Obrada izuzetaka

Obrada izuzetaka podrazumeva adekvatno rukovanje `Error` (ili nekim srodnim) objektom, čime se sprečava prekid izvršavanja JavaScript koda. Tri pojedinačne naredbe koje se koriste u procesu obrade izuzetaka su:

- `try`,
- `catch`,
- `finally`.

Svaka od tri upravo prikazane naredbe poseduje svoje specifično zaduženje. Ipak, prikazane naredbe se nikada ne koriste samostalno, već se pribegava njihovom kombinovanju:

- `try...catch`,
- `try...catch...finally`.

U nastavku će biti obrađene upravo navedene naredbe za obradu izuzetaka.

try...catch

Naredba `try...catch` sačinjena je iz pojedinačnih naredbi `try` i `catch`. Svaka od ovih naredbi ujedno je i zasebni blok koda. Tako naredba `try...catch` ima sledeći oblik:

```
try {
	//block of code to try
}
catch (err) {
	//block of code to handle errors
}
```

Unutar bloka koji se definiše `try` naredbom navodi se kôd koji može da proizvede izuzetak. Ukoliko tokom izvršavanja koda unutar `try` bloka dođe do izuzetka, aktivira se blok koda definisan `catch` naredbom.

Uvidom u sintaksu `try...catch` naredbe može se videti da `catch` blok prihvata jedan parametar koji je u primeru imenovan kao `err`. Ovo je promenljiva koja se od izvršnog okruženja popunjava objektom tipa `Error` kada dođe do pojave izuzetka. Naziv `err` je proizvoljan i može se menjati.

`try...catch` naredba može se iskoristiti za obradu izuzetka iz primera sa početka ove lekcije:

```
try {
	console.log("Value of variable b is : " + b);
}
catch (err) {
	console.log('Error name: ' + err.name);
	console.log('Error message: ' + err.message);
}
console.log("Hello World");
```

Primer sa početka lekcije sada je obogaćen kodom za obradu izuzetka. Naredba koja proizvodi izuzetak smeštena je unutar `try` bloka. Svaki izuzetak do koga dođe unutar `try` bloka biva obrađen od bloka `catch`. Upravo zbog toga se u primeru, odmah nakon pojave izuzetka u `try` bloku, tok izvršavanja koda preusmerava na blok `catch`, a promenljiva `err` dobija vrednost objekta `Error`.

Unutar `catch` bloka obavlja se čitanje vrednosti svojstava `name` i `message` i ispis njihovih vrednosti u konzoli:

```
Error name: ReferenceError
Error message: b is not defined
Hello World
```

Bitno je primetiti da prikazani pristup za obradu izuzetka osigurava nastavak izvršenja skripte, pa se upravo zbog toga na kraju dobija poruka i iz poslednje naredbe.

try...catch...finally

Pored `try` i `catch` naredbi, nešto ranije je spomenuta i još jedna naredba koja se može koristiti u postupku obrade izuzetaka. Reč je o naredbi `finally`:

```
try {
      block of code to try
}

catch (err) {

      block of code to handle errors
}

finally {

      block of code to always execute
}
```

Kôd definisan `finally` blokom uvek se izvršava, bez obzira na to da li je do greške došlo ili greške nema:

```
try {

      console.log("Value of variable b is : " + b);
}

catch (err) {

      console.log('Error name: ' + err.name);

      console.log('Error message: ' + err.message);
}

finally {

console.log('Hello from finally block');
}

console.log("Hello World");
```

Prikazani kôd proizvodi sledeći ispis unutar konzole:

```
Error name: ReferenceError

Error message: b is not defined

Hello from finally block

Hello World
```

Analizom dobijenog ispisa može se zaključiti da se `finally` blok izvršava odmah nakon `try` i `catch` blokova. Ipak, bitno je znati da će se `finally` uvek izvršiti, pa čak i kada ne dođe do izuzetka.

Pitanje

Unutar kog bloka se postavlja kôd koji može da proizvede izuzetak?

- **try**
- catch
- finally
- ništa od navedenog

Objašnjenje:

Blok try omogućava da se definiše blok koda koji će pokušati da se izvrši. Ukoliko tokom izvršavanja takvog koda dođe do greške, izvršavanje se prebacuje na catch blok.

Samostalno emitovanje izuzetaka

U do sada prikazanim primerima posao emitovanja izuzetaka obavljalo je samo okruženje. Ipak, moguće je i samostalno izvršiti njihovo emitovanje i to korišćenjem naredbe `throw`.

Sintaksa ove naredbe je:

```
throw expression;
```

Naredba `throw` formira se korišćenjem ključne reči `throw`, nakon koje se navodi izraz sa vrednošću koju je potrebno izbaciti. Naredbom `throw` može se izbaciti vrednost bilo kojeg tipa:

```
throw "This is Error 666";

throw 36;

throw false;
```

Ipak, najčešća je praksa izbacivanje izuzetaka objektnih tipova, koji su specijalno dizajnirani za tu namenu. Osnovni tip izuzetka u JavaScriptu je objektni tip `Error`. Pored ovog tipa izuzetka, JavaScript poznaje još nekoliko ugrađenih objekata za njihovo predstavljanje (tabela 14.1).

Tip greške	Opis
`Error`	osnovni objekat za predstavljanje grešaka
`RangeError`	objekat koji predstavlja grešku izazvanu prekoračenjem opsega; javlja se u situacijama kada se prekorači opseg niza, ili kada se metodama koje prihvataju numeričke vrednosti prosledi neodgovarajuća vrednost
`ReferenceError`	greška izazvana referenciranjem nepostojeće promenljive
`SyntaxError`	objekat koji predstavlja grešku izazvanu pokušajem interpretacije sintaksno pogrešno napisanog koda
`TypeError`	objekat koji predstavlja grešku do koje dolazi kada je vrednost koja učestvuje u nekom izrazu, neočekivanog tipa
`URIError`	objekat koji predstavlja grešku do koje dolazi kada se globalnim funkcijama koje rukuju URI-ima prosledi nekorektan URI

Tabela 14.1. Tipovi izuzetaka u JavaScriptu

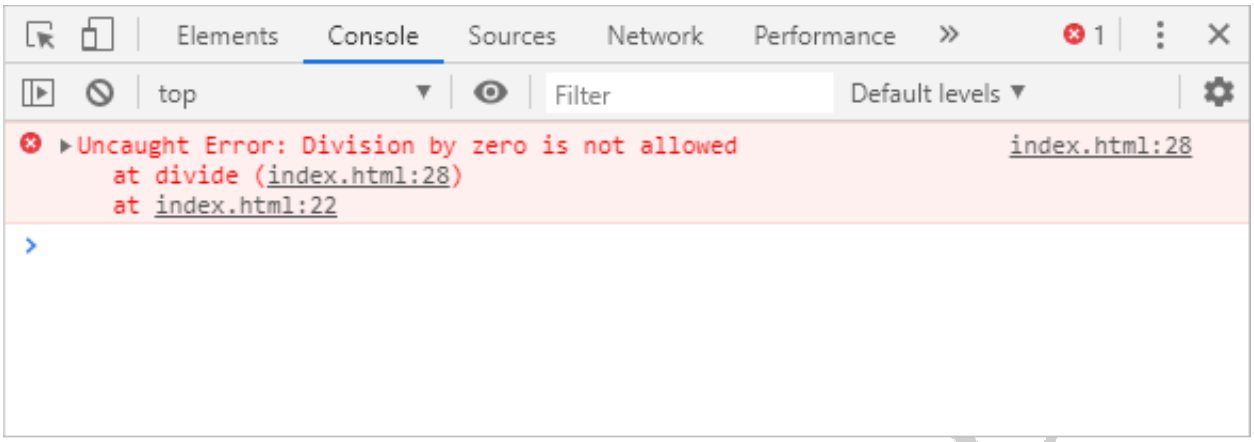
Samostalno emitovanje izuzetka korišćenjem naredbe `throw` biće ilustrovano na primeru deljenja broja nulom. Deljenje nekog broja nulom u matematici nije dozvoljeno, a neki programski jezici za takvo nešto imaju sankciju generisanja odgovarajućeg izuzetka. JavaScript nije jedan od takvih jezika, zbog postojanja specijalne numeričke vrednosti `Infinity`, pa je reč o idealnom primeru na kome se može ilustrovati samostalno emitovanje izuzetaka:

```
function divide(a, b) {
	if (b == 0) {
		throw Error('Division by zero is not allowed');
	}
	return a / b;
}
```

Kreirana je funkcija sa nazivom `divide`, koja prihvata dva parametra. Unutar tela funkcije se prvo proverava da li je drugi parametar (`b`) jednak nuli. Ukoliko jeste, obavlja se emitovanje izuzetka korišćenjem konstruktorske funkcije `Error`. Njoj se prosleđuje tekst koji predstavlja opis izuzetka.
Pozivanje ovakve funkcije može da izgleda ovako:

```
divide(5, 0);
```

Kada se ovakav kôd izvrši, unutar konzole web pregledača dobija se korisnički emitovan izuzetak (slika 14.2).



Slika 14.2. Primer korisnički definisanog JavaScript izuzetka

Slika 14.2. predstavlja neobrađeni izuzetak koji smo samostalno emitovali iz korisničke funkcije, što se može zaključiti po poruci koju smo nešto ranije samostalno definisali. Korisnički emitovani izuzeci mogu se obraditi kao i oni koje emituje JavaScript:

```
try {
      console.log(divide(5, 0));
}
catch (err) {
      console.log('Error name: ' + err.name);
      console.log('Error message: ' + err.message);
}
function divide(a, b) {
      if (b == 0) {
            throw Error('Division by zero is not allowed');
      }
      return a / b;
}
```

Sada je naredba u kojoj se poziva funkcija `divide()` smeštena unutar `try` bloka.

U slučaju pojave izuzetka aktivira se `catch` blok i obavlja logovanje informacija o izuzetku:

```
Error name: Error

Error message: Division by zero is not allowed
```

U prethodnom primeru se može videti da se samostalno emitovanje izuzetka obavlja pozivanjem konstruktorske funkcije `Error`. Parametar koji se njoj prosleđuje jeste tekst poruke koji opisuje izuzetak (`message`). Ipak, prilikom samostalnog kreiranja izuzetka moguće je definisati i njegov naziv:

```
var e = new Error('Division by zero is not allowed');
e.name = 'Invalid division';
throw e;
```

Ovoga puta `Error` objekat je kreiran korišćenjem ključne reči `new`, a zatim je postavljena vrednost njegovog svojstva `name`. `Error` objekat upakovan je u promenljivu `e`, i emitovan korišćenjem naredbe `throw`.

Prilikom izbacivanja izuzetka korišćenjem naredbe `throw` moguće je koristiti i bilo koji objekat prikazan u tabeli 14.1.

To ilustruje sledeći primer:

```
function divide(a, b) {

      if (b == 0) {

            throw Error('Division by zero is not allowed');

      } else if (b > 1000) {

            throw RangeError("Divider can't be greater than 1000");

      }

      return a / b;

}
```

Unutar funkcije `divide()` sada je dodat još jedan uslovni blok u kome se proverava da li je delilac veći od 1000. Ukoliko jeste, emituje se izuzetak u obliku objekta tipa `RangeError`. Stoga, kada se funkciji `divide()` kao drugi parametar prosledi vrednost veća od 1000, unutar konzole se dobija:

```
Error name: RangeError

Error message: Divider can't be greater than 1000
```

Samostalno emitovanje izuzetka se može obaviti i korišćenjem bilo kojeg korisnički kreiranog objekta:

```
function divide(a, b) {
	if (b == 0) {
		throw Error('Division by zero is not allowed');
	} else if (b > 1000) {
		throw {
			name: "Out of range",
			message: "Divider can't be greater than 1000"
		}
}
	return a / b;
}
```

Kôd proizvodi sledeći rezultat:

```
Error name: Out of range
Error message: Divider can't be greater than 1000
```

Rezime

- Tokom izvršavanja programskog koda bilo kojeg jezika može se pojaviti greška.
- Izuzetak predstavlja reprezentaciju greške koja se dogodila u kodu.
- Izuzetak se u JavaScriptu predstavlja `Error` objektom.
- Obrada izuzetaka podrazumeva adekvatno rukovanje `Error` objektom i sprečavanje prekida izvršavanja koda korišćenjem specijalne `try...catch` naredbe.
- Pored `try` i `catch` blokova, naredba `try…catch` može da sadrži još jedan dodatni blok, `finally`, koji se koristi kako bi se definisao kôd, koji će se obavezno izvršiti.
- Samostalno izbacivanje izuzetka se može postići korišćenjem naredbe `throw`.