Serijalizacija JSON-a

U prethodnoj lekciji bilo je reči o parsiranju JSON teksta korišćenjem metode `JSON.parse()`. Podjednako važan segment rada sa JSON podacima jeste i proces konvertovanja JavaScript objekata u JSON tekst. Realizacija takvog postupka drugačije se naziva serijalizacija. JSON serijalizaciji biće posvećena lekcija koja je pred vama.

Serijalizacija JavaScript objekata u JSON

Ugrađeni `JSON` objekat JavaScript jezika poseduje jednu posebnu metodu koja se koristi za obavljanje serijalizacije. Reč je o metodi `stringify()`.

JSON.stringify()

Metoda `JSON.stringify()` koristi se za serijalizaciju JavaScript objekata u JSON tekst. Njena sintaksa izgleda ovako:

```
JSON.stringify(value[, replacer[, space]])
```

Metoda `stringify()` može da prihvati tri parametra, od kojih je samo prvi parametar obavezan:

- `value` – vrednost koja će biti konvertovana u JSON tekst; ovo je obavezni parametar
- `replacer` – funkcija kojom se može uticati na osobine serijalizacije; ovo je opcioni parametar
- `space` – parametar kojim se može uticati na količinu praznih mesta koja će biti kreirana prilikom serijalizacije, kako bi se dobio čitljiviji kod

Povratna vrednost metode `JSON.stringify()` jeste JSON u tekstualnom obliku.

Nakon načelnog upoznavanja osobina metode `JSON.stringify()`, odmah će biti prikazan jedan primer serijalizacije relativno jednostavnog JavaScript objekta. Nakon prvog primera, upustićemo se u detaljniju analizu načina na koji se obavlja serijalizacija objekata u JSON, kao i u upoznavanje različitih pristupa za prilagođavanje takvog procesa.

Prvi primer serijalizacije korišćenjem JSON.stringify() metode

Prvi primer serijalizacije, koji podrazumeva prevođenje JavaScript podataka u JSON tekstualni format, biće obavljen nad JavaScript objektom koji će predstavljati jednu osobu. Objekat će biti kreiran na sledeći način:

```
var person = {};
person.id = 34;
person.firstName = "Ben";
person.lastName = "Lord";
person.address = {};
person.address.street = "Bos En Lommerweg 65";
person.address.zipCode = "1008 AG";
person.address.city = "Amsterdam";
person.adult = true;
```

Iako je objekat mogao biti kreiran i korišćenjem objektnog literala, odlučili smo se za ovakav pristup, koji podrazumeva deklarisanje i inicijalizovanje svojstava korišćenjem dot notacije, kako bi se kod za kreiranje JavaScript objekta razlikovao od JSON teksta. Kreirani objekat poseduje 5 svojstava:

- `id` – identifikaciona vrednost, `number` tip
- `firstName` – ime, `string` tip
- `lastName` – prezime, `string` tip
- `address` – adresa, `object` tip
- `adult` – da li je osoba punoletna, `boolean` tip

Kao što možete videti, sva svojstva osim svojstva `address` čuvaju podatke prostog tipa. Svojstvo `address` čuva referencu na objekat kojim se predstavlja nekoliko komponenata adrese:

- `street` – adresa, `string` tip
- `zipCode` – poštanski kod, `string` tip
- `city` – grad, `string` tip

Kako bi se ovakav objekat kojim se predstavlja jedna osoba serijalizovao, odnosno preveo u JSON tekstualni oblik, dovoljno je uraditi sledeće:

```
var json = JSON.stringify(person);
```

Metodi `JSON.stringify()` prosleđena je referenca na objekat koji je potrebno serijalizovati. Rezultat ovakve naredbe biće serijalizacija JavaScript objekta i smeštanje JSON teksta unutar promenljive `json`:

```
console.log(json);
```

Na kraju, unutar konzole se dobija:

```
{"id":34,"firstName":"Ben","lastName":"Lord","address":{"street":"Bos
En Lommerweg 65","zipCode":"1008 AG","city":"Amsterdam"},"adult":true}
```

Pitanje

Povratna vrednost `JSON.stringify()` metode je:

a) JSON u tekstualnom obliku
b) JavaScript objekat
c) JavaScript niz
d) boolean vrednost

Objašnjenje:

Povratna vrednost metode `JSON.stringify()` jeste JSON u tekstualnom obliku.

Osobine serijalizacije JSON.stringify() metodom

Upravo prikazani primer serijalizacije veoma je jednostavan, prevashodno zato što se obavlja serijalizacija podataka čiji tipovi su direktno podržani i unutar samog JSON formata. Takvi će biti i nekoliko narednih primera.

Serijalizacija niza

```
var json = JSON.stringify([13, 14, 15]); // '[13,14,15]'
```

Serijalizacija prostih vrednosti

```
var json = JSON.stringify(56.67); // '56.67'
```

Serijalizacija logičkih vrednosti

```
var json = JSON.stringify(false); // 'false'
```

Pored ovih bazičnih primera, veoma je bitno poznavati osobine `JSON.stringify()` metode i u nekim specifičnim slučajevima. Takvi slučajevi će biti prikazani u nastavku.

JSON.stringify() metodom ne mogu se serijalizovati funkcije

```
var person = {};
person.id = 34;
person.firstName = "Ben";
person.lastName = "Lord";
person.address = {};
person.address.street = "Bos En Lommerweg 65";
person.address.zipCode = "1008 AG";
person.address.city = "Amsterdam";
person.adult = true;

person.showInfo = function(){
    return firstName + " " + lastName;
}

var json = JSON.stringify(person);
console.log(json);
```

Sada je objektu koji predstavlja osobu dodato i svojstvo `showInfo`, čija vrednost je referenca na jednu metodu. U ispisu koji se dobija unutar konzole može se videti da se kreirana metoda `showInfo()` ignoriše.

Ukoliko se pokuša serijalizacija funkcije izvan objekta, dobija se `undefined`:

```
var json = JSON.stringify(function () { });
console.log(json);
```

Identična pravila važe i za vrednosti `undefined` i `Symbol` tipa – unutar objekata one su ignorišu, a kada su samostalne vrednosti, pretvaraju se u `undefined`.

Date objekti se u pozadini automatski konvertuju u string oblik, korišćenjem metode toJSON()

```
var json = JSON.stringify(new Date());
```

Primer proizvodi:

```
"2021-04-19T16:12:08.182Z"
```

Infinity, NaN i null se pretvaraju u null

```
var json = JSON.stringify([NaN, Infinity, null]);
```

Primer proizvodi:

```
"[null,null,null]"
```

Serijalizacija vrednosti BigInt tipa proizvodi grešku

```
var json = JSON.stringify(120n);
```

Primer proizvodi:

```
TypeError: Do not know how to serialize a BigInt
```

Specijalne vrste JavaScript kolekcija, kao što su Set i Map, ne mogu se serijalizovati

```
var json = JSON.stringify([new Set(["a", "b"]), new
Map([["a", "b"]])]);
console.log(json);
```

Primer proizvodi:

```
"[{},{}]"
```

Parametar replacer

Nešto ranije ste mogli da vidite da metoda `JSON.stringify()` pored jednog obaveznog parametra može da prihvati i dva opciona. Prvi opcioni parametar zove se *replacer* i njime je moguće uticati na osobine serijalizacije, definisanjem vrednosti koje će biti serijalizovane, kao i načina na koji će serijalizacija biti obavljena.

Vrednost parametra *replacer* se može definisati na dva načina, i to kao:

- funkcija – funkcija prihvata dva parametra, odnosno ključ i vrednost koji se serijalizuju; unutar tela funkcije je zatim moguće definisati na koji način će biti obavljena serijalizacija
- niz – niz koji sadrži nazive objektnih svojstava koja je potrebno serijalizovati

Parametar replacer kao funkcija

Prvo će biti prikazan jedan primer u kome će *replacer* parametar biti definisan kao funkcija. Primer će podrazumevati serijalizaciju nešto ranije prikazanog objekta koji predstavlja jednu osobu:

```
var person = {};
person.id = 34;
person.firstName = "Ben";
person.lastName = "Lord";
person.address = {};
person.address.street = "Bos En Lommerweg 65";
person.address.zipCode = "1008 AG";
person.address.city = "Amsterdam";
person.adult = true;

function replacer(key, value) {

    if (key === "id") {
        return undefined;
    } else if (key === "adult") {
        return (value === true) ? "yes" : "no";
    }

    return value;
}

var json = JSON.stringify(person, replacer);
console.log(json);
```

Replacer funkcija je definisana kao nezavisna funkcija imenovana baš tako – `replacer()`. Kao što je već rečeno, ona prihvata dva parametra, koja će tokom procesa serijalizacije automatski biti popunjena ključem i vrednošću. Funkcija `replacer()` prosleđuje se kao drugi parametar metodi `JSON.stringify()`. Na taj način, funkcija `replacer()` biće pozvana za svako svojstvo unutar objekta `person`, što nam omogućava da kontrolišemo proces serijalizacije.

Logikom koja je definisana unutar funkcije `replacer()` definisano je da svojstvo `id` uopšte neće biti serijalizovano i da će vrednosti svojstva `adult` da budu pretvorene u tekstualne vrednosti `yes` ili `no`, u zavisnosti od vrednosti. Tako primer ilustruje dve najčešće oblasti primene replacer funkcija – za kontrolisanje svojstava koja će biti serijalizovana i za transformisanje podataka pre nego što budu pretvoreni u tekst.

Iz replacer funkcije neophodno je emitovati neku povratnu vrednost. Takva povratna vrednost postaje vrednost konkretnog svojstva, osim u slučaju kada se kao povratna vrednost emituje `undefined`, čime se signalizira da je svojstvo i njegovu vrednost potrebno ignorisati prilikom serijalizacije. To je u prikazanom primeru učinjeno sa svojstvom `id`, pa na kraju njega nema u dobijenom JSON tekstu:

```
{"firstName":"Ben","lastName":"Lord","address":{"street":"Bos En
Lommerweg 65","zipCode":"1008 AG","city":"Amsterdam"},"adult":"yes"}
```

Parametar replacer kao niz

Parametar replacer može se definisati i u obliku niza i tada se takvim nizom kontrolišu svojstva koja će biti serijalizovana:

```
var person = {};
person.id = 34;
person.firstName = "Ben";
person.lastName = "Lord";
person.address = {};
person.address.street = "Bos En Lommerweg 65";
person.address.zipCode = "1008 AG";
person.address.city = "Amsterdam";
person.adult = true;

var replacer = ["firstName", "lastName", "address",
"street", "zipCode", "city", "adult"];

var json = JSON.stringify(person, replacer);
console.log(json);
```

Primer je sada modifikovan tako da `replacer` promenljiva čuva referencu na jedan niz. Unutar niza su definisani nazivi svih svojstava koja želimo da serijalizujemo. Drugim rečima, bilo koje svojstvo čiji naziv se ne navede unutar niza neće biti serijalizovano. Unutar niza nije navedeno svojstvo `id`, pa upravo zbog toga njega nema ni unutar JSON teksta koji se dobija:

```
{"firstName":"Ben","lastName":"Lord","address":{"street":"Bos En
Lommerweg 65","zipCode":"1008 AG","city":"Amsterdam"},"adult":true}
```

Bitno je da primetite i to da nije dovoljno navesti naziv nekog svojstva čija je vrednost objekat kako bi sva njegova svojstva bila serijalizovana. Neophodno je navesti i nazive svih njegovih svojstava, kako bi i ona bila serijalizovana. U prikazanom primeru, takva situacija postoji na primeru svojstva `address`.

Korišćenje metode toJSON() za uticanje na osobine serijalizacije

U prethodnim redovima prikazan je jedan način za uticanje na osobine serijalizacije JavaScript objekata, prilikom korišćenja `JSON.stringify()` metode. Pored upotrebe replacer parametra, na osobine serijalizacije je moguće uticati i korišćenjem metode `toJSON()` unutar objekata koji se serijalizuju.

Kada objekat koji se serijalizuje poseduje metodu `toJSON()`, ona se koristi za definisanje osobina serijalizacije. Vrednost koju ona vrati kao povratnu biće serijalizovana od strane metode `JSON.stringify()`:

```
var person = {};
person.id = 34;
person.firstName = "Ben";
person.lastName = "Lord";
person.address = {};
person.address.street = "Bos En Lommerweg 65";
person.address.zipCode = "1008 AG";
person.address.city = "Amsterdam";
person.adult = true;

person.toJSON = function(name){
    return {
        first_name: this.firstName,
        last_name: this.lastName
    }
}
var json = JSON.stringify(person);
console.log(json);
```

Sada je unutar objekta koji predstavlja osobu dodato i svojstvo `toJSON`, čija je vrednost funkcija. Stoga će povratna vrednost funkcije biti ona koja će se serijalizovati od strane `JSON.stringify()` metode:

```
{"first_name":"Ben","last_name":"Lord"}
```

Primer ilustruje efikasan način kojim je korišćenjem metode `toJSON()` obavljena transformacija podataka koji će biti serijalizovani. Zapravo, primerom je obavljeno sledeće:

- definisano je da će biti serijalizovana samo dva svojstva (`firstName` i `lastName`);

obavljena je transformacija ključeva, pa je tako umesto `firstName` definisano `first_name`, a umesto `lastName` – `last_name`.

Naravno, ovo je samo jedan od načina na koji se može koristiti `toJSON()` metoda, zato što je unutar nje moguće definisati potpuno proizvoljnu logiku koja će transformisati i prilagoditi osobine objekta koji se serijalizuje.

U upravo prikazanom primeru, možete videti da metoda `toJSON()` prihvata i jedan ulazni parametar. On automatski dobija vrednost od JavaScript izvršnog okruženja i takva vrednost se može koristiti za razumevanje okruženja u kome se nalazi objekat koji se serijalizuje:

- kada se, baš kao i u prikazanom primeru, obavlja direktna serijalizacija objekta tako što se on prosledi metodi `JSON.stringify()`, metoda `toJSON()` kao vrednost svog parametra dobija prazan string: `""`;
- kada se objekat koji se serijalizuje nalazi unutar nekog drugog objekta, kao vrednost nekog od njegovih svojstava, metoda `toJSON()` kao svoj ulazni parametar dobija naziv svojstva kojem je takav objekat pridružen;
- kada je objekat koji se serijalizuje član nekog niza, vrednost ulaznog parametra `toJSON()` metode jeste vrednost indeksa u `string` formatu.

Za kraj priče o `toJSON()` metodi, evo još jednog primera, koji predstavlja nadgradnju prethodnog:

```
var person = {};
person.id = 34;
person.firstName = "Ben";
person.lastName = "Lord";

person.address = {};
person.address.street = "Bos En Lommerweg 65";
person.address.zipCode = "1008 AG";
person.address.city = "Amsterdam";
person.address.toJSON = function (name) {
    return {
        street: this.street,
        zip_code: this.zipCode,
        city: this.city
    }
}

person.adult = true;

person.toJSON = function (name) {
    return {
        first_name: this.firstName,
        last_name: this.lastName,
        address: this.address
    }
}

var json = JSON.stringify(person);
console.log(json);
```

S obzirom na to da je vrednost jednog od svojstava `person` objekta objekat sa podacima o adresi, sada je i unutar takvog objekta definisana `toJSON()` metoda, koja će odrediti način serijalizacije unutrašnjeg `address` objekta. Bitno je da primetite da će takva metoda kao vrednost parametra `name` sada dobiti vrednost `address`, s obzirom na to da je reč o objektu koji se ne serijalizuje direktno, već se nalazi unutar `person` objekta kao vrednost njegovog `address` svojstva.

Primer će proizvesti sledeći izlaz:

```
{"first_name":"Ben","last_name":"Lord","address":{"street":"Bos En
Lommerweg 65","zip_code":"1008 AG","city":"Amsterdam"}}
```

Kombinovanje replacer parametra i toJSON() metode

U prethodnim redovima su prikazana dva načina koja se mogu koristiti za uticanje na proces serijalizacije – replacer parametar `JSON.stringify()` metode i `toJSON()` metoda samih objekata. Moguće je čak i kombinovati oba ova pristupa za uticanje na serijalizaciju. Tada je bitno znati da se prvo obavlja logika `toJSON()` metode, a zatim i filtriranje korišćenjem replacer parametra:

```
                var person = {};
                person.id = 34;
                person.firstName = "Ben";
                person.lastName = "Lord";
                person.address = {};
                person.address.street = "Bos En Lommerweg 65";
                person.address.zipCode = "1008 AG";
                person.address.city = "Amsterdam";
                person.address.toJSON = function (name) {
                    return {
                        street: this.street,
                        zip_code: this.zipCode,
                        city: this.city
                    }
                }
                person.adult = true;
                person.toJSON = function (name) {
                    return {
                        first_name: this.firstName,
                        last_name: this.lastName,
                        address: this.address
                    }
                }
                var replacer = ["first_name", "last_name", "address",
    "street", "zip_code",];
                var json = JSON.stringify(person, replacer);
                console.log(json);
```

Sada je u primer dodat i niz koji će biti korišćen kao replacer parametar. Serijalizacija započinje izvršavanjem `toJSON()` metoda objekata koji se serijalizuju, pa se nakon toga obavlja filtriranje nizom koji je definisan kao replacer parametar:

```
{"first_name":"Ben","last_name":"Lord","address":{"street":"Bos En
Lommerweg 65","zip_code":"1008 AG"}}
```

Kao što vidite, sve ovo za efekat ima to da se u finalnom JSON tekstu ne nađe svojstvo `city`, iako je definisano u povratnoj vrednosti `toJSON()` metode `address` objekta. Takvo svojstvo se nakon završetka logike `toJSON()` metode filtrira od strane replacer niza.

Formatiranje JSON teksta korišćenjem space parametra

Nešto ranije, prilikom prikaza sintakse metode `JSON.stringify()`, prikazano je da ona može da prihvati ukupno tri parametra, od kojih je samo prvi obavezan:

```
JSON.stringify(value[, replacer[, space]])
```

O poslednjem, space parametru još uvek nije bilo reči u dosadašnjem toku ove lekcije. Reč je o parametru kojim je moguće uticati na formatiranje JSON-a. U dosadašnjim primerima ste mogli da vidite da smo uvek dobijali JSON tekst u identičnom obliku – odnosno, bez ikakvih praznih mesta (prelazaka u novi red ili razmaka). Tako se može reći da se, podrazumevano, korišćenjem metode `JSON.stringify()` dobija maksimalno optimizovan oblik JSON teksta, koji zauzima minimalno memorije i kao takav je pogodan za čuvanje i distribuciju preko mreže. U većini slučajeva to će biti ponašanje koje ćemo želeti – s obzirom na to da nećemo imati potrebu za samostalnim, ručnim pregledom JSON teksta koji će se deliti između različitih aplikativnih komponenata. Ipak, u nekim situacijama, među kojima je i demonstracija JSON formata u procesu upoznavanja njegove strukture, dobro je dobijeni JSON tekst prikazati u obliku koji je maksimalno čitljiv ljudima. Upravo to omogućava poslednji space parametar `JSON.stringify()` metode.

Space parametar može da prihvati vrednost u dva različita oblika, i to:

- broj koji se odnosi na količinu razmaka (space karaktera) koji će biti postavljeni pre svakog pojedinačnog nivoa u JSON tekstu;
- tekst koji će biti postavljen pre svakog pojedinačnog nivoa u JSON tekstu.

Vrlo je bitno reći da definisanjem vrednosti space parametra koja se razlikuje od praznog stringa (""), `null` ili `undefined`, `JSON.stringify()` automatski počinje da prelama JSON tekst u nove redove. Svaki par naziva i vrednosti smešta se u novi red. Korišćenjem parametra space zapravo možemo da utičemo samo na uvlačenje svakog pojedinačnog nivoa koji se na taj način dobija, odnosno na karaktere koji će biti dodati kao prefiks svakom paru naziva i vrednosti.

Prvo će biti prikazano korišćenje space parametra sa numeričkom vrednošću:

```
var person = {};
person.id = 34;
person.firstName = "Ben";
person.lastName = "Lord";
person.address = {};
person.address.street = "Bos En Lommerweg 65";
person.address.zipCode = "1008 AG";
person.address.city = "Amsterdam";
person.adult = true;

var json = JSON.stringify(person, null, 6);
console.log(json);
```

Napomena

Kao što možete videti, space parametar se ne može definisati bez prethodnog definisanja replacer parametra. Zbog toga je u primeru za vrednost replacer parametra postavljena vrednost `null`.

Kao vrednost poslednjeg space parametra, metodi `JSON.stringify()` prosleđena je vrednost 6. Time je rečeno da želimo da svaki pojedinačni nivo, bude uvučen 6 praznih mesta više od prethodnog nivoa (slika 6.1).

```
{
	"id": 34,
	"firstName": "Ben",
	"lastName": "Lord",
	"address": {
		"street": "Bos En Lommerweg 65",
		"zipCode": "1008 AG",
		"city": "Amsterdam"
	},
	"adult": true
}
```

Slika 6.1. Primer formatiranja JSON teksta

Na slici 6.1. možete videti kako izgleda formatiranje JSON teksta nakon definisanja numeričke vrednosti 6 kao vrednosti space parametra. Svaki pojedinačni nivo uvučen je za šest dodatnih praznih mesta u odnosu na prethodni. Tako su svojstva `person` objekta (`id`, `firstName`, `lastName`...) uvučena šest praznih mesta u odnosu na vitičaste zagrade kojima se definiše objekat. Svojstva objekta `address` (`street`, `zipCode` i `city`) uvučena su za šest dodatnih praznih mesta, odnosno ukupno 12.

Napomena

Maksimalni broj praznih mesta koja se mogu koristi za uvlačenje je 10. Bilo koja numerička vrednost veća od 10 biće tretirana kao 10.

Kao što je nešto ranije rečeno, vrednost space parametra može biti i `string`. Najčešće se `string` vrednost definiše kao specijalni tab (`\t`) karakter, čime se svaki sukcesivni nivo uvlači za jedan tab karakter:

```
var person = {};
person.id = 34;
person.firstName = "Ben";
person.lastName = "Lord";
person.address = {};
person.address.street = "Bos En Lommerweg 65";
person.address.zipCode = "1008 AG";
person.address.city = "Amsterdam";
person.adult = true;

var json = JSON.stringify(person, null, "\t");
console.log(json);
```

Na ovaj način se dobija efekat kao na slici 6.2.

```
{
    "id": 34,
    "firstName": "Ben",
    "lastName": "Lord",
    "address": {
        "street": "Bos En Lommerweg 65",
        "zipCode": "1008 AG",
        "city": "Amsterdam"
    },
    "adult": true
}
```

Slika 6.2. Primer formatiranja JSON teksta (2)

Greške do kojih može doći prilikom korišćenja JSON.stringify() metode

Za kraj priče o serijalizaciji JavaScript objekata korišćenjem `JSON.stringify()` metode biće razmotrene i neke situacije koje mogu da proizvedu pojavu grešaka prilikom obavljanja takve operacije. Naime, pojava greške, odnosno emitovanje izuzetka od strane `JSON.stringify()` metode, retko se događa, ali postoje dve situacija u kojima do toga može doći:

- prilikom serijalizacije objekta sa cirkularnom referencom dolazi do emitovanja izuzetka tipa `TypeError: Converting circular structure to JSON`;
- prilikom pokušaja serijalizacije `bigint` vrednosti dolazi do emitovanja izuzetka `TypeError ("BigInt value can't be serialized in JSON")`.

Cirkularno referenciranje

Cirkularna referenca nastaje kada dva objekta međusobno referenciraju jedan drugog ili kada jedan objekat referencira samog sebe. Šta ovo praktično znači, biće prikazano na sledećem primeru:

```
var person = {};
person.name = "Ben";

var dog = {};
dog.name = "Jackie";

person.dog = dog;
dog.owner = person;
```

U primeru se kreiraju dva objekta – jednim se predstavlja osoba, a drugim pas. Objekat kojim se predstavlja pas postavljen je za vrednost svojstva `dog`, `person` objekta, dok je objekat koji predstavlja osobu postavljen kao vrednost svojstva `owner`, `dog` objekta. Na taj način je stvoreno cirkularno referenciranje između dva objekta:

```
var person = {};
person.name = "Ben";

var dog = {};
dog.name = "Jackie";

person.dog = dog;
dog.owner = person;

var json = JSON.stringify(person);
```

Pokušaj serijalizacije bilo kog od ova dva objekta stvoriće izuzetak:

```
Uncaught TypeError: Converting circular structure to JSON
```

Cirkularne reference su generalno loš način dizajniranja objektno orijentisane strukture, pa ih je najbolje izbegavati. Najlakše se prevazilaze davanjem većeg značaja jednom od objekata, pri čemu je potrebno ukloniti referencu iz objekta manje važnosti. To bi u prikazanom primeru praktično značilo sledeće – iz objekta `dog` uklonilo bi se svojstvo `owner`, odnosno referenca na objekat `person`:

```
var person = {};
person.name = "Ben";

var dog = {};
dog.name = "Jackie";

person.dog = dog;
//dog.owner = person;

var json = JSON.stringify(person);
console.log(json);
```

Sada se serijalizacija obavlja bez problema:

```
{"name":"Ben","dog":{"name":"Jackie"}}
```

bigint serijalizacija

Još jedan scenario u kome može doći do pojave izuzetka prilikom serijalizacije korišćenjem metode `JSON.stringify()` jeste pokušaj serijalizacije podataka bigint tipa:

```
var json = JSON.stringify(100n);
```

Ovakva naredba proizvodi:

```
Uncaught TypeError: Do not know how to serialize a BigInt
```

Najbolji način za prevazilaženje ovakvog problema jeste korišćenje replacer parametra sa odgovarajućom logikom koja će detektovati `bigint` vrednosti i obaviti njihovu konverziju u `string`:

```
var json = JSON.stringify(100n, (key, value) =>
    typeof(value) === 'bigint' ? value.toString() : value
);
console.log(json);
```

Sada je replacer parametar definisan kao anonimna Arrow funkcija, unutar koje se obavlja detekcija tipa vrednosti koja se serijalizuje. Ukoliko je tip vrednosti `bigint`, vrednost se konvertuje u `string`. U protivnom, vrednost se isporučuje u izvornom obliku. Na ovaj način, prevazilazi se problem pojave izuzetka prilikom serijalizacije `bigint` tipova.

Rezime

- Proces konvertovanja JavaScript objekata u JSON drugačije se naziva JSON serijalizacija.
- Metoda `JSON.stringify()` koristi se za serijalizaciju JavaScript objekata u JSON tekst.
- Prvim parametrom metode `JSON.stringify()` definiše se JavaScript objekat ili vrednost koju je potrebno serijalizovati.
- `JSON.stringify()` metodom ne mogu se serijalizovati funkcije.
- Prilikom serijalizacije `JSON.stringify()` metodom, `Date` objekti se u pozadini automatski konvertuju u `string` oblik, korišćenjem metode `toJSON()`.
- Prilikom serijalizacije `JSON.stringify()` metodom, `Infinity`, `NaN` i `null` se pretvaraju u `null`.
- Specijalne vrste JavaScript kolekcija, kao što su `Set` i `Map`, ne mogu se serijalizovati korišćenjem `JSON.stringify()` metode.
- Drugi parametar `JSON.stringify()` metode zove se replacer i njime je moguće uticati na proces serijalizacije.
- Na osobine serijalizacije je moguće uticati i korišćenjem metode `toJSON()` unutar objekata koji se serijalizuju.
- Kada objekat koji se serijalizuje poseduje metodu `toJSON()`, ona se koristi za definisanje osobina serijalizacije; vrednost koju ona vrati kao povratnu biće serijalizovana od strane metode `JSON.stringify()`.
- Poslednjim, trećim parametrom metode `JSON.stringify()` moguće je uticati na formatiranje JSON teksta.
- Prilikom serijalizacije objekta sa cirkularnom referencom dolazi do emitovanja izuzetka tipa `TypeError: Converting circular structure to JSON`.
- Prilikom pokušaja serijalizacije `bigint` vrednosti dolazi do emitovanja izuzetka `TypeError`.