Parsing Json in javascript

**URL**

A Uniform Resource Locator (URL), is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A web resource is any data that can be obtained via web, such as HTML documents, PDF files, PNG images, JSON data, or plain text.

A generic URL has the following form:

**scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]**

The square brackets indicate that the part is optional. A scheme is a way of addressing resources, such as http, ftp, mailto, or file.

The part following two slashes is called the authority part. The authority part contains 1) an optional authentication section of a user name and password, separated by a colon, followed by an at symbol (@) 2) a host, which is either a host name of or an IP address, 3) an optional port number, separated from the host by a colon.

**JSON**

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write and for machines to parse and generate. The official Internet media type for JSON is application/json. The JSON filename extension is .json.

In our examples, we use JSON data from http://time.jsontest.com.

{

"time": "11:27:26 AM",

"milliseconds\_since\_epoch": 1494934046126,

"date": "05-16-2017"

}

The GET request returns this JSON string.

**Step 1**

Pass the desired URL as an object:

URL url = new URL(“The required URL”);

**Step 2**

Type cast the URL object into a HttpURLConnection object. The benefit of doing this is that we will be able to harness the properties of the HttpURLConnection class to validate features. For example, set the request type or check the status of the response code:

HttpURLConnection conn = (HttpURLConnection)url.openConnection();

**Step 3**

Set the request type, as in, whether the request to the API is a GET request or a POST request.

conn.setRequestMethod(“GET”);

**Step 4**

Open a connection stream to the corresponding API.

conn.connect();

**Step 5**

Get the corresponding response code.

int responsecode = conn.getResponseCode();

**Step 6**

Now we need to perform a check so that if the response code is not 200, we throw a runtime exception, or otherwise carry on the rest of the procedure. The structure would be like this:

1

if(responsecode != 200)

throw new RuntimeException(“HttpResponseCode: “ +responsecode);

else

{

Next part of the functionality

}

**Step 7**

I have used the method scanner to read each line from the API and fetch the data in string format. Now, this part is inside else { } like I mentioned above.

Scanner sc = new Scanner(url.openStream());

while(sc.hasNext())

{

inline+=sc.nextLine();

}

System.out.println(“\nJSON data in string format”);

System.out.println(inline);

sc.close();

{

“results”:[

{

“address\_components”:[

{

“long\_name”:“Chicago”,

“short\_name”:“Chicago”,

“types”:[

“locality”,

“political”

]

},

{

“long\_name”:“Cook County”,

“short\_name”:“Cook County”,

“types”:[

“administrative\_area\_level\_2”,

“political”

]

},

{

“long\_name”:“Illinois”,

“short\_name”:“IL”,

“types”:[

“administrative\_area\_level\_1”,

“political”

]

},

],

“status”:“OK”

}

**Step 8**

Declare an instance of the JSONParser:

**JSONParser parse = new JSONParser();**

**Step 9**

Convert the string objects into JSON objects:

**JSONObject jobj = (JSONObject)parse.parse(inline);**

{

"results" : [

{

"place\_id" : "ChIJ7cv00DwsDogRAMDACa2m4K8",

"types" : [ "locality", "political" ]

} ]

}

I would now like to get the corresponding values under the results array.

**Step 10**

First, convert the JSON object into JSONArray object like this:

JSONArray jsonarr\_1 = (JSONArray) jobj.get(“results”);

**Step 11**

Once the JSON objects are stored in the array, read the corresponding JSONArray objects, and convert it to JSON objects again so you get the elements within the results array. Here is how you do it

//Get data for Results array

for(int i=0;i<jsonarr\_1.size();i++)

{

//Store the JSON objects in an array

//Get the index of the JSON object and print the values as per the index

JSONObject jsonobj\_1 = (JSONObject)jsonarr\_1.get(i);

System.out.println(“Elements under results array”);

System.out.println(“\nPlace id: ” +jsonobj\_1.get(“place\_id”));

System.out.println(“Types: ” +jsonobj\_1.get(“types”));

}

**Navigating Programatically**

**What is routing?**

Routing is the capacity to show different pages to the user. That means the user can move between different parts of an application by entering a URL or clicking on an element.

As you may already know, by default, React comes without routing. And to enable it in our project, we need to add a library named react-router.

**Setting up the router**

To enable routing in our React app, we first need to import BrowserRouter from react-router-dom.

In the App.js file, enter the following:

import React, { Fragment } from "react";

import "./index.css"

import { BrowserRouter as Router } from "react-router-dom";

export default function App() {

return (

<Router>

<main>

<nav>

<ul>

<li><a href="/">Home</a></li>

<li><a href="/about">About</a></li>

<li><a href="/contact">Contact</a></li>

</ul>

</nav>

</main>

</Router>

);

}

This should hold everything in our app where routing is needed. That means, if we need routing in our entire app, we must wrap our higher component with BrowserRouter.

**Router Hooks**

Router hooks make things much easier. Now you can access the history, location, or parameters in an easy and elegant way.

**useHistory**

The useHistory hook gives us access to the history instance without pulling it from props.

import { useHistory } from "react-router-dom";

const Contact = () => {

const history = useHistory();

return (

<Fragment>

<h1>Contact</h1>

<button onClick={() => history.push('/') } >Go to home</button>

</Fragment>

)

};

**useParams**

This hook helps us get the parameter passed on the URL without using the props object.

import { BrowserRouter as Router, Route, Link, Switch, useParams } from "react-router-dom";

export default function App() {

const name = 'John Doe'

return (

<Router>

<main>

<nav>

<ul>

<li><Link to="/">Home</Link></li>

<li><Link to={`/about/${name}`}>About</Link></li>

</ul>

</nav>

<Switch>

<Route path="/" exact component={Home} />

<Route path="/about/:name" component={About} />

</Switch>

</main>

</Router>

);

}

const About = () => {

const { name } = useParams()

return (

// props.match.params.name

<Fragment>

{ name !== 'John Doe' ? <Redirect to="/" /> : null }

<h1>About {name}</h1>

<Route component={Contact} />

</Fragment>

)

};

**useLocation**

This hook returns the location object that represents the current URL.

import { useLocation } from "react-router-dom";

const Contact = () => {

const { pathname } = useLocation();

return (

<Fragment>

<h1>Contact</h1>

<p>Current URL: {pathname}</p>

</Fragment>

)

};

**Styling React Using CSS**

There are many ways to style React with CSS, this tutorial will take a closer look at three common ways:

**Inline styling**

CSS stylesheets

CSS Modules

Inline Styling

To style an element with the inline style attribute, the value must be a JavaScript object:

**Example:**

Insert an object with the styling information:

const Header = () => {

return (

<>

<h1 style={{color: "red"}}>Hello Style!</h1>

<p>Add a little style!</p>

</>

);

}

**JavaScript Object**

You can also create an object with styling information, and refer to it in the style attribute:

Example:

Create a style object named myStyle:

const Header = () => {

const myStyle = {

color: "white",

backgroundColor: "DodgerBlue",

padding: "10px",

fontFamily: "Sans-Serif"

};

return (

<>

<h1 style={myStyle}>Hello Style!</h1>

<p>Add a little style!</p>

</>

);

}

**CSS Modules**

Another way of adding styles to your application is to use CSS Modules.

CSS Modules are convenient for components that are placed in separate files.

The CSS inside a module is available only for the component that imported it, and you do not have to worry about name conflicts.

Create the CSS module with the .module.css extension, example: my-style.module.css.

**Create a new file called "my-style.module.css" and insert some CSS code in it:**

my-style.module.css:

.bigblue {

color: DodgerBlue;

padding: 40px;

font-family: Sans-Serif;

text-align: center;

}

**Import the stylesheet in your component:**

**Car.js:**

import styles from './my-style.module.css';

const Car = () => {

return <h1 className={styles.bigblue}>Hello Car!</h1>;

}

export default Car;

Import the component in your application:

**index.js:**

import ReactDOM from 'react-dom';

import Car from './Car.js';

ReactDOM.render(<Car />, document.getElementById('root'));

----------------------------------------------------------------------------------------------------Link: [JavaScript - reading JSON from URL with Fetch API, JQuery, XMLHttpRequest (zetcode.com)](https://zetcode.com/javascript/jsonurl/)