Date
23/02/2017

The PHP tags are enclosed in single quotes inside the JSON.parse function call since JSON uses double quotes for names & values. If you leave out the quotes or use double quotes, Javascript errors will be triggered.

JSON to PHP using json_decode()

PHP's json_decode function takes a JSON string and converts it into a PHP variable. Typically, the JSON data (i.e. JSON string) will represent a Javascript array or object literal which json_decode will convert into a PHP array or object. The following 2 examples demonstrate, first with an array & the 2nd with an object:

```
$json = '["apple", "orange", "banana", "strawberry"]';
$ar = json_decode($json);
//access first element of $ar array:
echo $ar[0]; //apple
```

By default, objects are converted to standard objects by json_decode:

```
$json = '{
            "title": "title1",
            "author": "author1",
            "edition" : 6
}';
$book = json_decode($json);
// access title of $book object
echo $book->title; // title1
```

Date
23/02/2019

The json_decode function provides an optional 2nd argument to convert object to associative array. On above example, now title & other elements can be accessed using array syntax:

```
// $json same as example object above
// pass true to convert objects to associative arrays:
$book = json_decode($json, true);
// access title of $book array
echo $book['title']; // title1
```

On previous page, a multidimensional array $books has been defined:

By default the result of json_decode will be a numerically indexed array of objects:

```
$json = '[
    {
        "title": "title1",
        "author": "author1"
    },
    {
        "title": "title2",
        "author": "author2"
    },
    {
        "title": "title3",
        "author": "author3"
    }
]';
$books = json_decode($json);
```

Date
23/02/2017

// access property of object in array
echo $books[1]->title; //title2

If we pass true as the 2nd argument to json_decode, the result is a multidimensional array that is numerically indexed at the outer level and associative at the inner level:

// $json same as example object above
// pass true to convert objects to associative arrays
$books = json_decode($json, true);
// numeric/associative array access
echo $books[1]['title']; // title2

Pass variables & data from PHP to JavaScript

There are actually several approaches to do this. Some require more overhead than others and some are considered better than others.

1. Use AJAX to get the data you need from the server.
2. echo the data into the page somewhere, and use Javascript to get the information from the DOM.
3. echo the data directly to Javascript

1. Use AJAX to get the data you need from the server.

This method is considered the best, because your server side and client side scripts are completely separate.

Pros:

1. Better separation between layers → If tomorrow

Date
23/02/2017

you stop using PHP, and want to move to a servelet, a REST API, or some other service, you don't have to change much of the Javascript code.

2. More readable → Javascript is Javascript, PHP is PHP. Without mixing the two, you get more readable code on both languages.

3. Allows for async data transfer → Getting the information from PHP might be time/resources expensive. Sometimes you just don't want to wait for the information, load the page, and have the information reach whenever.

4. Data is not directly found on the markup → This means that your markup (i.e. HTML) is kept clean of any additional data, and only Javascript sees it.

This means that, the data through AJAX can be seen only through Chrome developer tool box (which is also an event driven javascript box) & can not be seen through page view source (i.e. ctrl+u).

Cons:

1. Latency → AJAX creates an XML HTTP request & XML HTTP request are carried over network and have network latencies.

2. Put & get from DOM

This method is less preferable to AJAX, but it still has its advantages. It's still relatively

Date
23/02/2017

separated between PHP and JAVAscript in a sense that there is no PHP directly in the Javascript.

Pros:

Fast → Dom operations are often quick and you can store & access a lot of data relatively quickly.

Cons:

1. <input type="hidden"> can be used to store information, but doing so, means that we have a meaningless element in your HTML.
2. Data that PHP generates is outputted directly to the HTML source, meaning that we get a bigger & less focused HTML source.
3. Harder to get structured data
4. Tightly couples PHP to your data logic

Example: index.php

```
<div id="dom-target" style="display:none;">
  <?php $output=42; echo htmlspecialchars($output);
  // we have to escape because the result will not
  // be valid HTML otherwise
  ?>
</div>
<script>
  var div = document.getElementById("dom-target");
  var myData = div.textContent;
</script>
```

3. echo the data directly to Javascript:

This is probably the easiest to understand, and