JSON

JSON file

When you want to transport say a complicated object or an array of object to another compute using network. You cannot transport a complicated object just directly as "it is". You must serialize it which essentially turns the object into a stream of bytes, a string (since string can be represented using binary easily). Then send the stream of bytes over the wires, and the receiver can then reconstruct the object using the stream of bytes that you have sent.

JSON file syntax

- JSON file can either be an array in which it starts with hard bracket, and it defines an array of objects in it
- JSON file can also be an object directly in which it starts with soft brackets and it defines the property of the object
- Each key of the object in JSON must be in quotation marks! No question!
- Each key can map to another string, a float, boolean, an array, or another dictioanry
- Each element in either object or array must be separated by a comma

JSON.stringify

This is the function that will translate a given object into a JSON. The resulted string is called *JSON-encoded* or *serialized object* or *marshalled object* that is ready to be send over the wire since it is a plain string of bytes.

`JSON.stringify` is able to support the conversion of

- Objects
- Arrays
- strings, numbers, boolean values, and even null

Into JSON

However, here are some of the excluded items that are excluded for the conversion

- Object methods
- Symbolic keys and values
- Properties that are mapped to undefined

```
let user = {
  sayHi() { // ignored
    alert("Hello");
  },
  [Symbol("id")]: 123, // ignored
  something: undefined // ignored
};

alert( JSON.stringify(user) ); // {} (empty object)
```

Nested objects

Nested objects are handled automatically by `JSON.stringify` so we don't have to worry about that!

Circular references

`JSON.stringify` does not support circular references! It will be an error during conversion!

Extra parameter

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

1. `value`: Is the object that we are encoding
2. `replacer`: It can either be an array of property that we specifically ask to encode and ignore the rest, or it can be a mapping function. The function has to take in `function(key, value)`. The function will be called on for every `(key, value)` pair of the object and return the `replace` value that will be used for doing the encoding instead of the original. Return `undefined` if you don't want to encode a specific property. The reason why this is done to give us a finer control on how we want the `stringify` to be carried out
3. `space`: Specifies how many spaces to use for the output encoded string

Custom "toJSON"

If an object implements it's own `toJSON` method for string conversion `JSON.stringify` will automatically call it if it is available to do the encoding.

JSON.parse

On the other hand, once you receive the JSON encoded string you would want to decode it back into an object so you can process it. You can do that with the `JSON.parse` method

```
let value = JSON.parse(str, [receiver]);
```

- `str`: The JSON-string to parse

- `receiver`: An optional `function(key, value)` that will be called for each `(key, value)` pair and can be used to transform the value after it has been parse

> There is some peculiarity with the receiver function, mainly in that it will be calling on each key and value pair, but at the last final iteration it will also called on the entire object where the key is an empty string and the value is the original object/array itself.

Revision #1
Created 2022-12-23 21:04:41 UTC by Tamarine
Updated 2022-12-24 02:52:49 UTC by Tamarine