JSON.stringify

</> Medium 20mins 1.46k done

Implement a function `jsonStringify`, similar to `JSON.stringify` that converts a JavaScript value into a JSON string.

- Only JSON-serializable values (i.e. boolean, number, `null`, array, object) will be present in the input value.
- Ignore the second and the third optional parameters in the original API.

Examples

```
jsonStringify({ foo: 'bar' }); // '{"foo":"bar"}'
jsonStringify({ foo: 'bar', bar: [1, 2, 3] }); // '{"foo":"bar","bar":[1,2,3]}'
jsonStringify({ foo: true, bar: false }); // '{"foo":true,"bar":false}'
```

Other types

```
jsonStringify(null); // 'null'
jsonStringify(true); // 'true'
jsonStringify(false); // 'false'
jsonStringify(1); // '1'
jsonStringify('foo'); // '"foo"'
```

Asked at these companies

Snap Amazon Google

Solution

Because non-primitive values can contain both primitive and non-primitive values, we have to use a recursive solution.

We can think of the `value`s like a tree. Primitive values are nodes that do not have children, and array/object types are nodes that have children of any value type. We can convert each node into a tree by converting each of its children into strings. We can work upwards from the "leaf" nodes, which in this case is primitive values because they cannot contain any children, and build the string up from the leaf nodes all the

primitive values because they cannot contain any children, and build the string up from the leaf nodes all the way to the root.

Let's define how to stringify each value type:

- `null`: Directly convert it into the string `null`
- **Boolean**: Directly convert `true` / `false` into a string via `String()`
- **Numbers**: Directly convert into a string via `String()`
- **Strings**: We have to wrap the value with double quotes as strings use double quotes
- **Arrays**: Recursively stringify each child item, then concatenate them with a comma, and wrap them in square brackets: `[` and `]`
- **Objects**: Convert each key/value pair (also called an entry) into a `"key":{stringifiedValue}` format by recursively stringifying the values, concatenate them with a comma, and wrap them in braces: `{` and `}`

Now that we know how to stringify each value, we just have to know how to determine the type of value. Since `null`, `boolean`, `number`s can produce the desired string just by using `String()`, we can handle them together as the default case at the bottom. To determine the types:

- **Arrays**: Use `Array.isArray()`
- **Objects**: Use `typeof value === 'object' && value !== null`. The check for `!== null` is important because `typeof null` is `'object'` but we have to handle `null`s differently from objects
- **Strings**: Use `typeof value === 'string'`

Here's the code that determines the type of a value and stringify each value type appropriately.

JavaScript TypeScript

```
/**
 * @param {*} value
 * @return {string}
 */
export default function jsonStringify(value) {
  if (Array.isArray(value)) {
    const arrayValues = value.map((item) => jsonStringify(item));
    return `[${arrayValues.join(',')}]`;
  }

  if (typeof value === 'object' && value !== null) {
    const objectEntries = Object.entries(value).map(
      ([key, value]) => `"${key}":${jsonStringify(value)}`
```

```
    );
    return `{${objectEntries.join(',')}}`;
  }

  if (typeof value === 'string') {
    return `"${value}"`;
  }

  return String(value);
}
```

Here's an alternative that does the typechecking in a cleaner fashion by using `switch` cases:

```
function getType(value) {
  if (value === null) {
    return 'null';
  }

  if (Array.isArray(value)) {
    return 'array';
  }

  return typeof value;
}

/**
 * @param {*} value
 * @return {string}
 */
export default function jsonStringify(value) {
  const type = getType(value);

  switch (type) {
    case 'array':
      const arrayValues = value.map((item) => jsonStringify(item)).join(',');
      return `[${arrayValues}]`;
    case 'object':
      const objectValues = Object.entries(value)
```

```
      .map(([key, value]) => `"${key}":${jsonStringify(value)}`)
      .join(',');
    return `{${objectValues}}`;
  case 'string':
    return `"${value}"`;
  default:
    // Handles null, boolean, numbers.
    return String(value);
 }
}
```

Limitations

The code above is a simplified version of `JSON.stringify` and doesn't handle many other types available in JavaScript nor support the API's replacer and formatting options. Some other cases:

- Cyclic references within the objects.
- Doesn't handle other types like `undefined`, `Function`, `Map`, `Set`, `Symbol`, `RegExp`, `Date`, and more.
- Double quotes and other special characters like backslashes, tabs, within strings should be escaped.