DAY21 对象详解和深浅拷贝

2023年5月17日 17:14

JSON对象的概述

一.JSON对象的概述

1.JSON使用类似JavaScript语法来描述数据对象，但不是JavaScript独有的，其支持许多不同的编程语言.

2.JSON是一种轻量级的数据交换格式，里面字符串需要添加双引号

3.JSON不能包含JavaScript相关的语法.

4.JSON拥有独立的文件扩展名(.json)

注意：json里面的数据是字符串或者数字。

```
{
  "success": 1,
  "result": {
    "status": "ALREADY_ATT",
    "phone": "13800138000",
    "area": "010",
    "postno": "100000",
    "att": "中国,北京",
    "ctype": "中国移动138卡",
    "par": "1380013",
    "prefix": "138",
    "operators": "中国移动",
    "style_simcall": "中国,北京",
    "style_citynm": "中华人民共和国,北京市"
  }
}
```

二.提供了两个非常重要的静态方法

1.JSON.parse():将json格式的字符串转换成对象格式(具有json检测功能)

```
console.log(JSON.parse(obj));
```

2.JSON.stringify():将对象转换成json格式的字符串。

```
const obj1 = {
  name: 'zhangsan',
  age: 18,
  sex: '男'
}
console.log(obj1);
console.log(JSON.stringify(obj1));//{"name":"zhangsan","age":18,"sex":"男"}
```

三.简单的应用

1.比较两个对象里面的内容是否相等。

```
const obj1 = {
  name: 'zhangsan'
};
const obj2 = {
  name: 'zhangsan'
};
console.log(obj1 == obj2);//false 比较的地址
console.log(JSON.stringify(obj1) === JSON.stringify(obj2));//true
```

2.本地存储存储对象格式的数据

```
const obj1 = {
  name: 'zhangsan',
  age: 18,
  sex: '男'
}
localStorage.setItem('objData', JSON.stringify(obj1));//存储
console.log(JSON.parse(localStorage.getItem('objData')));//获取
```

3.利用两个方法实现最简单的深拷贝

值传递和引用传递

1.值传递：基本类型遵循的

```
let a = 1;
let b = a;//将a的值复制一份传递给b，a和b是两个变量
b++;
console.log(a);//1
console.log(b);//2
```

2.引用传递：其实就是对象间的赋值，引用类型遵循引用传递，也叫地址传递

```
let arr1 = [1, 2, 3];
let arr2 = arr1;//引用传递，将arr1的地址给arr2，也就是arr1和arr2指向同一地址，地址指向内容，操作的也就是同一块内容。
arr2.push(4, 5, 6);
console.log(arr1);//[1, 2, 3, 4, 5, 6]
console.log(arr2);//[1, 2, 3, 4, 5, 6]
```

如何解决引用传递带来的影响，可以采用深浅拷贝来实现

浅拷贝

那么如何解决对象间的赋值问题，通常使用浅拷贝或者深拷贝了来进行对象间的赋值操作。

浅拷贝：拷贝一层。

一.浅拷贝的实现

```
const obj = {
  name: 'zhangsan',
  age: 18,
  sex: '男'
};
```

1.遍历实现

```
function shallowCopy(target) {//target:被拷贝的对象
  if (typeof obj !== 'object' || obj === null) return;  // 如果传入的值不是对象类型或者传入null都直接结束
  // 判断拷贝的是数组还是自定义对象
  let result = Array.isArray(obj) ? [] : {};//判断拷贝的是数组或对象
  for (let key in obj) {
    if (obj.hasOwnProperty(key)) {//判断防止继承原型链上的其他属性(hasOwnProperty判断括号里面的属性是否是对象自身下面的属性，返回布尔值)
      result[key] = obj[key];
    }
  }
  return result;
}
let result = shallowCopy(obj);
result.name = '尼古拉斯赵四';
console.log(obj);
console.log(result);
```

2.利用Object.assign方法

```
function shallowCopy(obj) {
  if (typeof obj !== 'object' || obj === null) return;  // 如果传入的值不是对象类型或者传入null都直接结束
  let result = Array.isArray(obj) ? [] : {};//判断拷贝的是数组或对象
  Object.assign(result, obj);
  return result;
}
let result = shallowCopy(obj);
result.age = 111111;
console.log('obj', obj);
console.log('result', result);
```

3.利用扩展运算符

```
const obj = {
  name: 'zhangsan',
  age: 18,
  sex: '男'
};
const obj1 = { ...obj };
obj.name = '王五';
console.log('obj', obj);
console.log('obj1', obj1);
```

问题：

- 但是上面的函数无法进行深层次的拷贝，这个时候我们可以使用深拷贝来完成
- 所谓深拷贝，就是能够实现真正意义上的数组和对象的拷贝

深拷贝

一.深拷贝的简单的实现方式

```
const obj1 = {
  name: 'zhangsan',
  info: {
    width: 173,
    height: 200,
    address: {
      add1: '中国杭州',
      add2: '中国北京'
    }
  }
}

function deepCopy(obj) {//obj:拷贝的对象
  return JSON.parse(JSON.stringify(obj));
}
const obj2 = {
  name: 'zhangsan',
  info: {
    width: function () { console.log('1111') },
    height: undefined,
    address: {
      add1: '中国杭州',
      add2: '中国北京'
    }
  }
}

console.log(JSON.stringify(obj2));//这里自动将函数和undefined给去掉
const result = deepCopy(obj1);
result.info.address.add1 = 'hangzhou china';
console.log('obj', obj1);
```

```
console.log('result', result);
```

弊端：如果被拷贝对象的属性值是函数或者undefined,相关的属性就无法拷贝，最终会消失。

二.深拷贝 - 递归遍历方式

```
const obj = {
  name: 'zhangsan',
  age: 18,
  info: {
    width: 173,
    height: 200,
    address: {
      add1: '中国杭州',
      add2: '中国北京'
    }
  }
}

function deepCopy(obj) {//obj:被拷贝的对象
  if (typeof obj !== 'object' || obj === null) return  // 判断传入的是一个数组或者自定义对象，不能是null
  let result = Array.isArray(obj) ? [] : {};//准备一个接收的对象，这个对象必须是数组或者自定义对象
  for (let key in obj) {//遍历对象
    if (obj.hasOwnProperty(key)) {//判断属性来自对象自身，不包括原型链。
      // if (typeof obj[key] === 'object' && obj[key] !== null) {//证明对象的这一项还是对象，递归操作
      //   result[key] = deepCopy(obj[key]);
      // } else {//不是对象，直接赋值操作
      //   result[key] = obj[key]
      // }
      result[key] = (typeof obj[key] === 'object' && obj[key] !== null) ? deepCopy(obj[key]) : obj[key];
    }
  }
  return result;
}

let result = deepCopy(obj);
result.info.address.add2 = 'beijing china';
console.log('obj', obj);
console.log('result', result);
```

Object.defineProperty

1.Object.defineProperty()方法会直接在一个对象上定义一个新属性，或者修改一个对象的现有属

性，并返回此对象。

2.Object.defineProperty语法

Object.defineProperty(obj, prop, desc)

- obj 需要定义属性的当前对象
- prop 当前需要定义的属性名
- desc 属性描述符,对象格式

3.属性描述符(四个属性，两个方法)

3.1.value： 属性对应的值,可以使用任意类型的值，默认为undefined

3.2.writable：属性的值是否可以被重写(修改)。true可以被重写，false不能被重写，默认为false

3.3.configurable：描述属性是否配置，以及可否删除，true可以，false不能，默认为false

3.4.enumerable：描述属性是否会出现在for in 或者 Object.keys()的遍历中

```
const obj = {
  name: 'zhangsan'
};

新增一个属性
Object.defineProperty(obj, 'age', {
  value: 18,
  writable: true,
  configurable: true,
  enumerable: true
});

修改属性
obj.age = 250;

删除属性
delete obj.age;

遍历对象
for (let key in obj) {
  console.log(key + '----' + obj[key]);
}
console.log(obj);
```

4.存取器描述：属于属性描述里面的内容，定义属性如何被存取。

注意：当使用了getter或setter方法，不允许使用writable和value这两个属性(如果使用，会直接报错)

get 是获取值的时候的方法，类型为 function，获取值的时候会被自动调用，不设置时为undefined

set 是设置值的时候的方法，类型为 function，设置值的时候会被自动调用，不设置时为 undefined

```
const obj = {
  name: 'zhangsan'
};
```

```
let num;
Object.defineProperty(obj, 'age', {
  get: function () {//获取属性值，return的结果才是获取的属性值
    // console.log('获取属性值会自动触发');
    return num;
  },
  set: function (v) {//设置属性值  自定义一个参数，可以获取设置的值
    console.log('设置属性值会自动触发', v);
    num = v;
  }
});
obj.age = 100;//自动调用了set方法，并且set方法的第一个参数就是设置的值
console.log(obj.age);//自动调用get方法，并且get的返回值就是这里的值
```

Model-View-ViewModel（简称为MVVM）

1.Model是指数据模型(数据)

2.View层是视图层，也就是用户界面，前端主要由HTML和CSS来构建。

3.ViewModel是视图模型层。将视图和数据进行双向数据绑定

MVVM是一种软件设计模式

MVVM的出现促进了前端开发与后端的分离，极大提高了前端的开发效率。

MVVM的核心是ViewModel层。就像一个中转站，负责Model中的数据对象让数据变得更容易管理和使用，该层向上和视图层进行双向数据绑定，向下和Model层通过接口进行数据交互，起到承上启下的作用。

记住一句话：数据驱动视图

数据指的是渲染的普通数据

视图(HTMLDOM模板,渲染出来的结构)

```
<script src="https://cdn.bootcdn.net/ajax/libs/vue/3.2.47/vue.global.js"></script>

<body>
  <div id="app">
    <input type="text" v-model="msg">
    {{msg}}
    <hr>
    <ul>
      <li v-for="item in list">{{item}}</li>
    </ul>
  </div>
</body>

  const { createApp } = Vue;
  const app = createApp({
```

```
    data() {
      return {
        msg: '我是一段数据',
        list: [
          1111,
          2222,
          3333,
          4444,
          5555
        ]
      }
    },
  });
  const vm = app.mount('#app')
```

数据劫持

数据劫持：其实就是数据代理(将原始的数据复制一份，通过复制的数据操作原始数据)。

具体指的是在访问或者修改对象的某个属性时，通过一段代码拦截这个行为，进行额外的操作或者修改返回结果。

形成双向数据绑定的思路

```
<body>
  <input type="text" id="inp1">
  <input type="text" id="inp2">
  <input type="text" id="inp3">
  <div id="box"></div>
</body>
```

```
  const obj = {//数据
    name: 'zhangsan',
    age: 18,
    sex: '男'
  }
```

正常的DOM渲染，如果数据发生变化，视图不会改变，必须重新渲染

```
  const box = document.querySelector('#box');
  box.innerHTML = `我的名字叫${obj.name}，我今年${obj.age}岁，我是${obj.sex}的`;

  obj.name = '尼古拉斯赵四';
  box.innerHTML = `我的名字叫${obj.name}，我今年${obj.age}岁，我是${obj.sex}的`;
```

封装函数实现数据劫持

```
  const obj = {//原始对象
    name: 'zhangsan',
    age: 18,
    sex: '男'
```

```
}
function observer(obj, fn) {//obj: 原始对象    fn:DOM操作，传递函数
  let result = {};//复制对象
  for (let key in obj) {
    Object.defineProperty(result, key, {
      get() {//获取值触发，返回值就是获取的结果(将原始的数据复制一份)
        return obj[key]
      },
      set(v) {//设置值触发，获取设置的值
        obj[key] = v;
        fn && typeof fn === 'function' && fn();//fn存在，同时是函数，才调用函数
      }
    });
  }
  return result;
}
let result = observer(obj, render);
function render() {

box. innerHTML = `我的名字叫${result.name}，我今年${result.age}岁，我是${result.sex}
的`;
}
render();
inp1.oninput = function () {
  result.name = this.value;
};
inp2.oninput = function () {
  result.age = this.value;
};
inp3.oninput = function () {
  result.sex = this.value;
};
```

proxy

1.数据双向绑定原理： vue 2.x 使用的是Object.defineProperty()(Vue 在 3.x 版本之后改用 Proxy 进行实现)

2.数据代理： 通过一个对象代理对另一个对象中属性的操作(读/写)

可以理解为现实生活中的中介，比如，我们要租一个房子，我们和中介谈价钱、谈条件，那么，实际的房主就是被代理的对象，中介公司就是数据代理。

3.区别

proxy代理整个对象，Object.defineProperty通过代理对象里面的属性。

4.Proxy的语法

Proxy是ES6中的一个构造函数，我们使用时需要通过new关键字来创建Proxy的实例

Proxy构造函数接收两个参数：target和handler

- target表示所要拦截的目标对象，
- handler参数也是一个对象，用来定制拦截的行为，比如重写get或set函数

```
const obj = {//被代理的对象
  name: 'zhangsan',
  age: 18,
  sex: '男'
}

let result = new Proxy(obj, {//obj:被代理的对象    result:生成的代理对象
  get(target, prop) {// target就是上面的obj,表示被代理的对象，prop被代理的对象所有的属性(只要获取属性值触发get方法)
    return target[prop]; //自带遍历的
  },
  set(target, prop, value) {// value设置的值    设置属性值，触发set方法
    target[prop] = value;
    box.innerHTML = `我的名字叫${result.name}，我今年${result.age}岁，我是${result.sex}的`;
  }
});

console.log('obj', obj);
console.log('result', result);
inp1.oninput = function () {
  result.name = this.value;
};
box.innerHTML = `我的名字叫${result.name}，我今年${result.age}岁，我是${result.sex}的`;
```

数据劫持和数据代理 - 回答面试题

1.vue2.0双向数据绑定原理：Object.definePropery数据劫持(getter和setter) + 发布订阅模式

2.vue3.0双向数据绑定原理：Proxy

区别：

中使用了 es6 的 Proxy API 对数据代理，通过 reactive() 函数给每一个对象都包一层 Proxy，通过 Proxy 监听属性的变化，从而 实现对数据的监控。