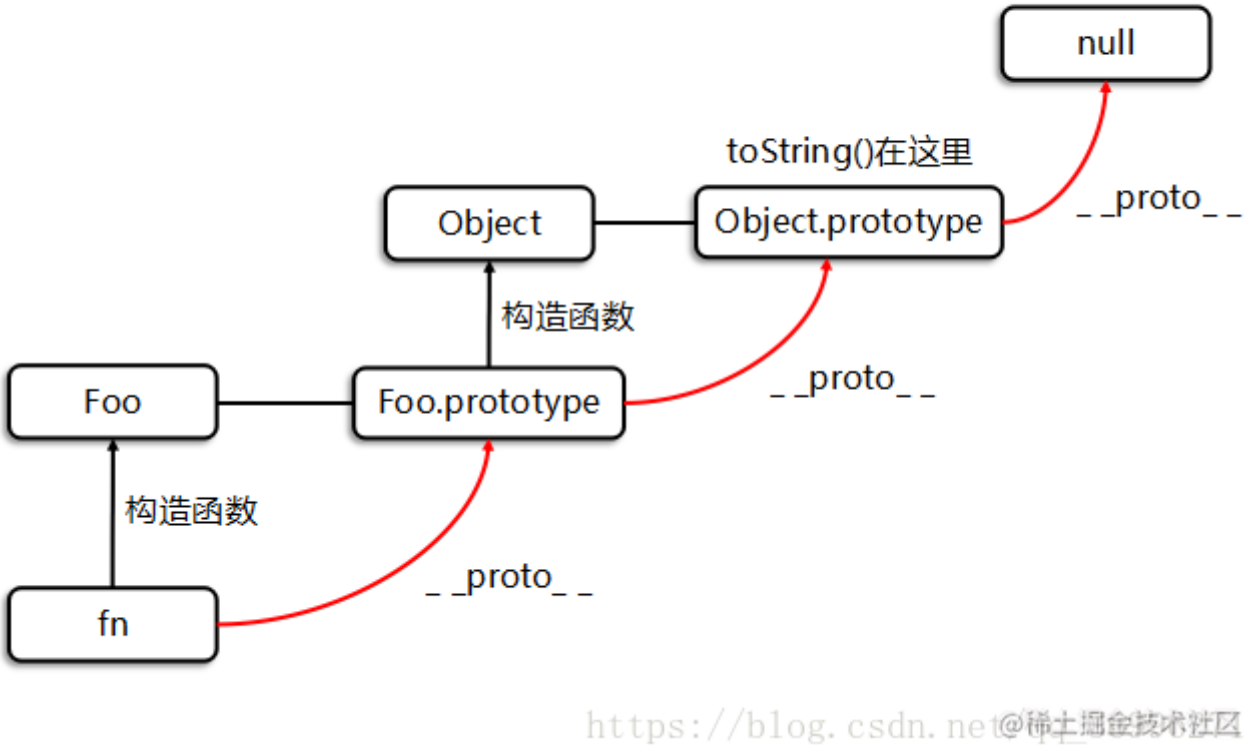
前端面经复习20220809

1.JS原型与原型链

参考链接



blog.csdn.net/qq_36996271...~~all~~baidu_landing_v2~default-5-82527256-null-null.142^v39^pc_rank_34_ecpm0,185^v2^control&utm_term=JS%E5%8E%9F%E5%9E%8B%E4%B8%8E%E5%8E%9F%E5%9E%8B%E9%93%BE&spm=1018.2226.3001.4187

2.深拷贝与浅拷贝

浅拷贝只复制指向某个对象的指针，而不复制对象本身，新旧对象还是共享同一块内存。但深拷贝会另外创造一个一模一样的对象，新对象跟原对象不共享内存，修改新对象不会改到原对象。

2.1浅拷贝的实现方式：

2.1.1 Object.assign

`Object.assign()` 方法可以把任意多个的源对象自身的可枚举属性拷贝给目标对象，然后返回目标对象。但是 `Object.assign()` 进行的是浅拷贝，拷贝的是对象的属性的引用，而不是对象本身。

```ini 复制代码
var obj = { a: {a: "kobe", b: 39} };
var initalObj = Object.assign({}, obj);
initalObj.a.a = "wade";
console.log(obj.a.a); // wade
```

当boject只有一层的时候，是深拷贝。

```ini 复制代码
let obj = {
   username: 'kobe'
};
let obj2 = Object.assign({},obj);
obj2.username = 'wade';
console.log(obj);//{username: "kobe"}
```

2.1.2 Array.prototype.concat()

```ini 复制代码
let arr = [1, 3, {
   username: 'kobe'
}];
let arr2=arr.concat();
arr2[2].username = 'wade';
console.log(arr);
```

2.1.3 Array.prototype.slice()

```ini 复制代码
let arr = [1, 3, {
   username: ' kobe'
}];
let arr3 = arr.slice();
arr3[2].username = 'wade'
console.log(arr);
```

2.2深拷贝的实现方式

2.2.1 JSON.parse(JSON.stringify())

```ini
let arr = [1, 3, {
   username: ' kobe'
}];
let arr4 = JSON.parse(JSON.stringify(arr));
arr4[2].username = 'duncan';
console.log(arr, arr4)
```

这种方法虽然可以实现数组或对象深拷贝，但是不能处理函数。

2.2.2手写递归方法

（待添加）

2.2.3函数库lodash

```ini
var _ = require('lodash');
var obj1 = {
   a: 1,
   b: { f: { g: 1 } },
   c: [1, 2, 3]
};
var obj2 = _.cloneDeep(obj1);
console.log(obj1.b.f === obj2.b.f);
// false
```

3.防抖和节流

3.1防抖

函数防抖：触发高频事件后n秒内函数只会执行一次，如果n秒内高频时间再次被触发，则重新计算时间。

实现方式：每次触发事件设置一个延迟调用方法，并且取消之前的延时调用方法。
缺点：如果事件在规定的时间间隔内被不断的触发，则调用方法会不断的延迟。

```xml
//防抖debounce代码：
  <body>
    <input type="text" />
  </body>
  <script>
    let inp = document.querySelector('input')
    inp.oninput = debounce(function () {
      console.log(this.value);
    }, 500)

    function debounce(fn, delay) {
      let t = null;
      return function () {
        if (t !== null) {
          clearTimeout(t);
        }
        setTimeout(() => {
          fn.call(this);
        }, delay)
      }
    }
  </script>
```

3.2节流

函数节流：高频事件触发，但在n秒内只会执行一次，所以节流会稀释函数的执行频率。
实现方式：每次触发事件时，如果当前有等待执行的延时函数，则直接return。

```ini
//节流代码：
   let flag = true;
    window.onscroll = function(){
        if(flag){
            setTimeout=>((){
                console.log("111");
                flag = true;
            },500)
        }
        flag = false;
    }
```

3.3总结

函数防抖：将多次操作合并为一次操作进行。原理是维护一个计时器，规定在delay时间后触发函数，但是在delay时间内再次触发的话，就会取消之前的计时器而重新设置。这样一来，只有最后一次操作能被触发。
函数节流：使得一定时间内只触发一次函数。原理是通过判断是否有延迟调用函数未执行。
区别：函数节流不管事件触发有多频繁，都会保证在规定时间内一定会执行一次真正的事件处理函数，而函数防抖只是在最后一次事件后才触发一次函数。比如在页面的无限加载场景下，我们需要用户在滚动页面时，每隔一段时间发一次 Ajax 请求，而不是在用户停下滚动页面操作时才去请求数据。这样的场景，就适合用节流技术来实现。