【前端面试】JS如何将类数组转化为真正的...

前言

说起类数组，可能很多小伙伴脑袋都是一团浆糊。什么是类数组？它是数组吗？不是数组为什么要和数组扯上关系？...等等好多问题。其实类数组非常的简单，甚至在项目开发中我们可能遇到过，只是我们没有仔细去思考罢了。既然类数组带有数组两个字，那么它肯定和数组是有关系的，我们是可以将数组转为真正的数组的。

今天就来总结一些将类数组转为数组的方法！

1.什么是类数组?

民间流传着这样一句话：在 JS 中一切皆对象。这句话虽然不够严谨，但是还是有参考价值的。

我们使用的数组 Array 其实也是一个对象，只不过它稍微特殊一点。

我们平常使用的对象都是以键值对的形式出现的，我们也可以把数组想象成一个键值对对象，键就是索引，值就是具体对应的元素，它还带有 length 属性。

想象变为现实！就成了类数组。

类数组解释：

- 类数组是一个对象
- 属性名称即键值使用数字
- 带有length属性

上面三点确定的话基本上就可以认为这个对象是一个类数组，是不是和我们上面将数组想象成对象的形式一样啊！

类数组特点：

- 可以转化为真正的数组
- 没有数组方法，即不可调用数组的原型方法，如push、concat等等
- 可以像数组那样循环

我们通过一个实际的对比来看看它们真正的区别！

定义一个类数组对象，**代码如下：**

```
// 类数组对象
let objArray = {
  0: "小猪课堂",
```

```
  1: "小猪课堂",
  2: "会飞的猪",
  length: 3
}
```

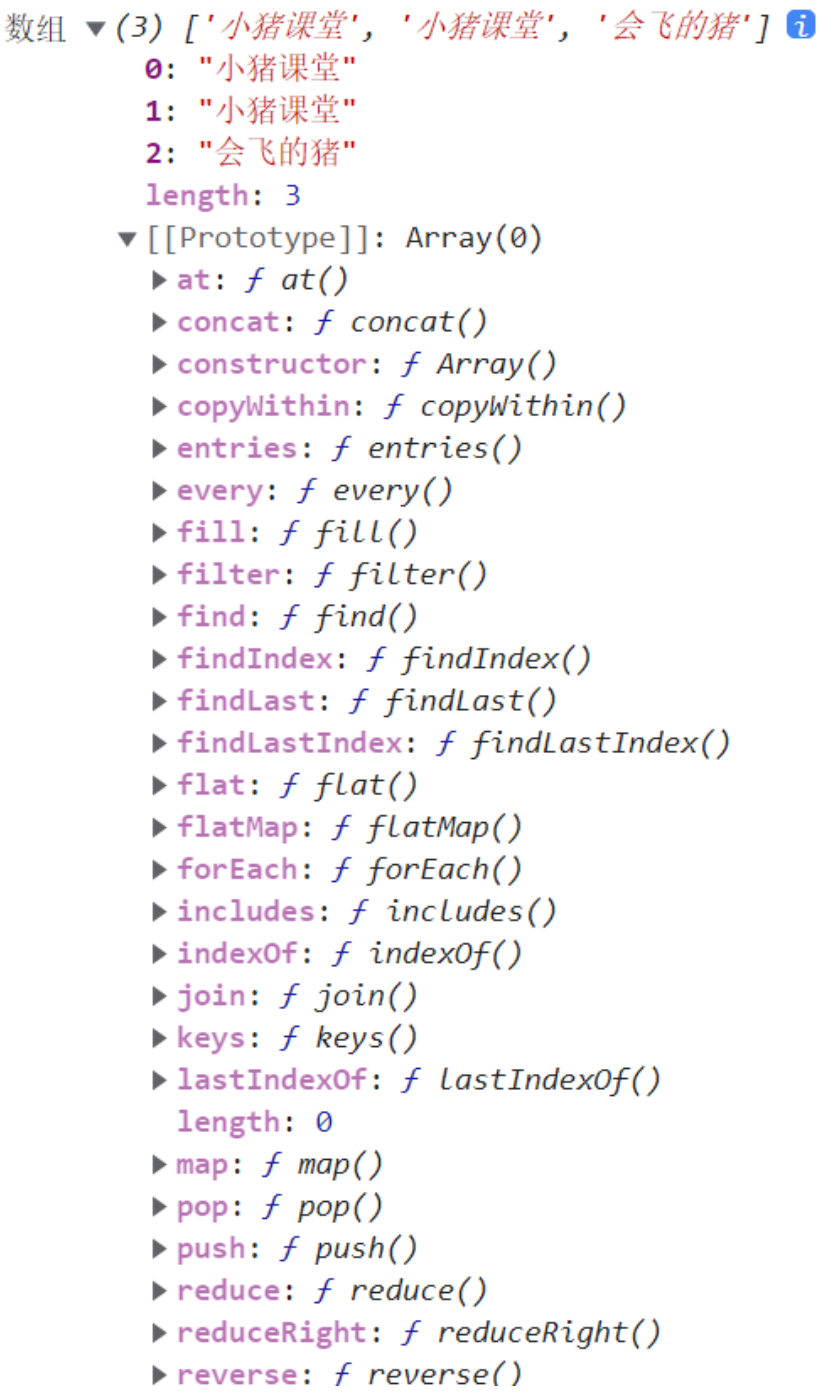
再定义一个真正的数组，**代码如下：**

```
let arr = ["小猪课堂", "小猪课堂", "会飞的猪"]
```

输出结果对比：

类数组

数组



乍一看两者很类似，但是仔细看就会发现它们的原型方法是不一样的，数组的原型方法有很多，及门上我们都使用过。

2.常见的类数组

我们在实际开发中可能遇到过很多的类数组，只是我们没有注意罢了。

2.1 arguments

我们都知道每个函数都有一个argumens属性，它代表了函数接收的参数集合，其实它就是一个类数组对象。

示例代码：

```
<script>
  function fn1(num, str, obj) {
```

```
    console.log("arguments", arguments)
  }
  fn1(12, "小猪课堂", { name: "小猪课堂" });
</script>
```

输出结果：

```
arguments ▼Arguments(3) [12, '小猪课堂', {…}, callee: ƒ, Symbol(Symbol.iterator): ƒ]
            0: 12
            1: "小猪课堂"
          ▶ 2: {name: '小猪课堂'}
          ▶ callee: ƒ fn1(num, str, obj)
            length: 3
          ▶ Symbol(Symbol.iterator): ƒ values()
          ▶ [[Prototype]]: Object
```

从输出结果来看它非常符合我们对类数组对象的定义。

2.2 HTMLCollection

以前我们有一篇文章里面就提到过HTMLCollection，它是Element元素的集合，比如我们使用Element.children获得的就是HTMLCollection集合。

示例代码：

```
<script>
  let box1 = document.getElementById("box1");
  console.log("children", box1.children);
</script>
```

输出结果：

```
children ▼HTMLCollection(2) [p, span]
           ▶ 0: p
           ▶ 1: span
             length: 2
           ▼ [[Prototype]]: HTMLCollection
             ▶ item: ƒ item()
               length: 2
             ▶ namedItem: ƒ namedItem()
             ▶ constructor: ƒ HTMLCollection()
             ▶ Symbol(Symbol.iterator): ƒ values()
               Symbol(Symbol.toStringTag): "HTMLCollection"
             ▶ get length: ƒ length()
             ▶ [[Prototype]]: Object
```

上面的输出结果基本和我们对类数组的定义一致，出了children属性外，DOM操作中还有很多放多都是获得类数组，比如document.getElementByTagName、document.getElementByClassName等等。

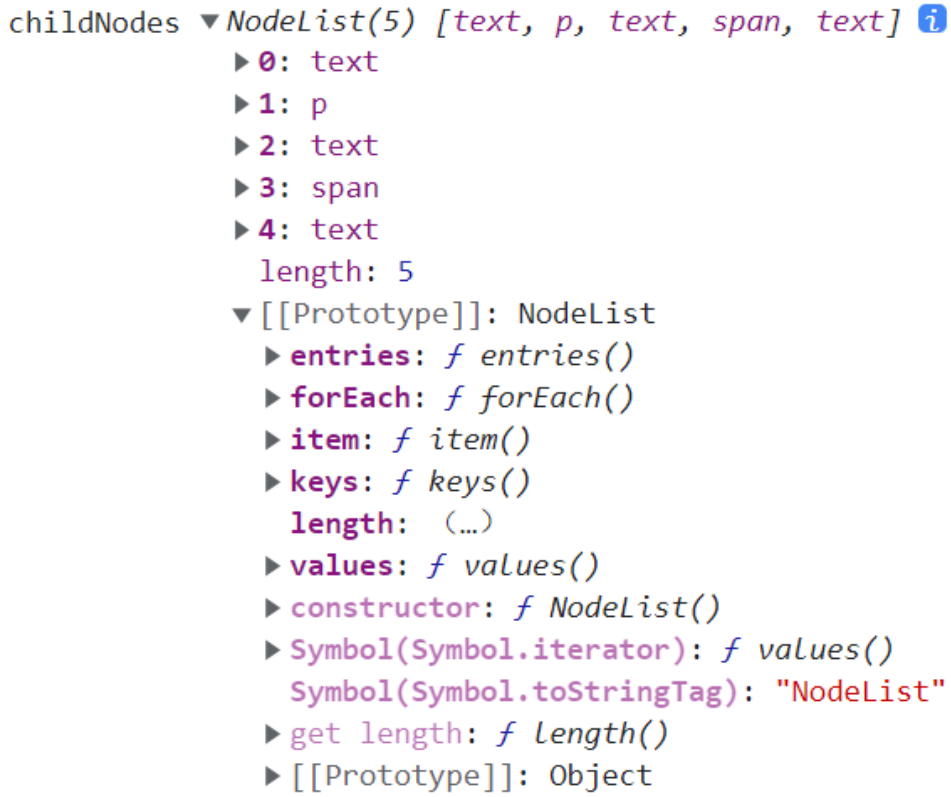
2.3 NodeList

NodeList集合是HTMLCollection集合的父级，它可以返回所有子节点，包括文本节点、注释、元素等等，它也是一个类数组，它可以通过Node.childNodes获得。

示例代码：

```
<script>
  let box1 = document.getElementById("box1");
  console.log("childNodes", box1.childNodes);
</script>
```

输出结果：



出了NodeList外，还有类似于document.querySelectorAll等方法也可以返回类数组对象。

2.4 其它

类数组还有很多很多，毕竟大家都说JS中一切皆对象。

String：

string其实也可以称为类数组，因为它可以通过str.length的方式得到长度，通过str[0]的方式获取字符。

LocaStorage和sessionSotrage:

这两个API主要是用来设置浏览器存储的，我们将它们打印出来，会发现它也基本符合类数组的定义，但是我们在实际情况基本不考虑。

示例代码：

```
localStorage.setItem('title', "小猪课堂");
localStorage.setItem("name", "小猪课堂");
console.info("localstorage", localStorage)
```

输出结果：

```
localstorage ▼Storage {title: '小猪课堂', num: '0', name: '小猪课堂', length: 4}
                name: "小猪课堂"
                num: "0"
                title: "小猪课堂"
                key: <不可读>
                length: 4
              ▶[[Prototype]]: Storage
```

3.类数组转为数组

我们了解了类数组和数组之间的关系之后，就可以考虑转换的问题了，因为在很多时候，我们都知道操作数组很便利，毕竟有很多原生API，既然类数组这么像数组，那么我们把它转成真正的数组不久可以享用数组里面的方法了。

将类数组转为数组的最大目的：可以调用数组的原型方法。

3.1 Array.from()

这个方法专门用来将类数组转为真正的数组，非常的好用。

官网的解释：

> Array.from() 方法对一个类似数组或可迭代对象创建一个新的，浅拷贝的数组实例。

想要详细了解Array.from方法的小伙伴可以移步官网：Array.from()

示例代码：

```
<script>
  // 将字符串转化为真正数组
  let str = "小猪课堂";
  console.log("str", Array.from(str)); // ['小', '猪', '课', '堂']

  // 将NodeList转为真正数组
  let box1 = document.getElementById("box1");
  console.log("childNodes", Array.from(box1.childNodes)); // [text, p, text,

  // 将其它标准类数组转为真正数组
  let objArray = {
    0: "小猪课堂",
    1: "小猪课堂",
    2: "会飞的猪",
    length: 3
  }
  console.log("childNodes", Array.from(objArray)); // ['小猪课堂', '小猪课堂',
</script>
```

输出结果：

```
str ▼(4) ['小', '猪', '课', '堂']
        0: "小"
        1: "猪"
        2: "课"
        3: "堂"
        length: 4
      ▶[[Prototype]]: Array(0)
```

```
childNodes ▼(5) [text, p, text, span, text]
             ▶0: text
             ▶1: p
             ▶2: text
             ▶3: span
             ▶4: text
               length: 5
             ▶[[Prototype]]: Array(0)
```

```
childNodes ▼(3) ['小猪课堂', '小猪课堂', '会飞的猪']
               0: "小猪课堂"
               1: "小猪课堂"
               2: "会飞的猪"
               length: 3
             ▶[[Prototype]]: Array(0)
```

3.2 扩展运算符

扩展运算符是ES6新增的语法，它的原理就是将参数中的可遍历属性浅拷贝到当前对象中，它可以作用于对象和数组，因为总体而言这两者都是对象。我们可以利用它的这个特性，将对象中遍历出来的属性放到我们的数组中去，从而得到一个真正的数组。

示例代码：

```
// 将字符串转化为真正数组
let str = "小猪课堂";
console.log("str", [...str]); // ['小', '猪', '课', '堂']

// 将NodeList转为真正数组
let box1 = document.getElementById("box1");
console.log("childNodes", [...box1.childNodes]); // [text, p, text, span, te
```

输出结果：

```
str ▼(4) ['小', '猪', '课', '堂']
        0: "小"
        1: "猪"
        2: "课"
        3: "堂"
        length: 4
      ▶[[Prototype]]: Array(0)
```

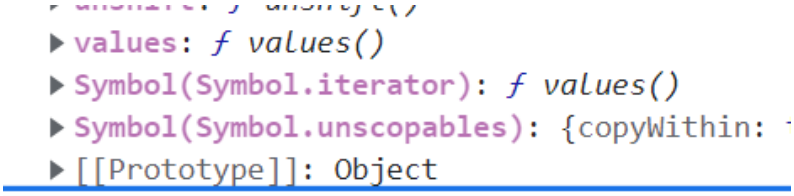
```
childNodes ▶(5) [text, p, text, span, text]
```

上段代码中，大家会发现我为什么没有objArray 类数组了。因为它不是一个可迭代对象，扩展运算符内部其实是调用了 Iterator 接口，我们声明的objArray类数组对象什么都没组，它是不可迭代的。

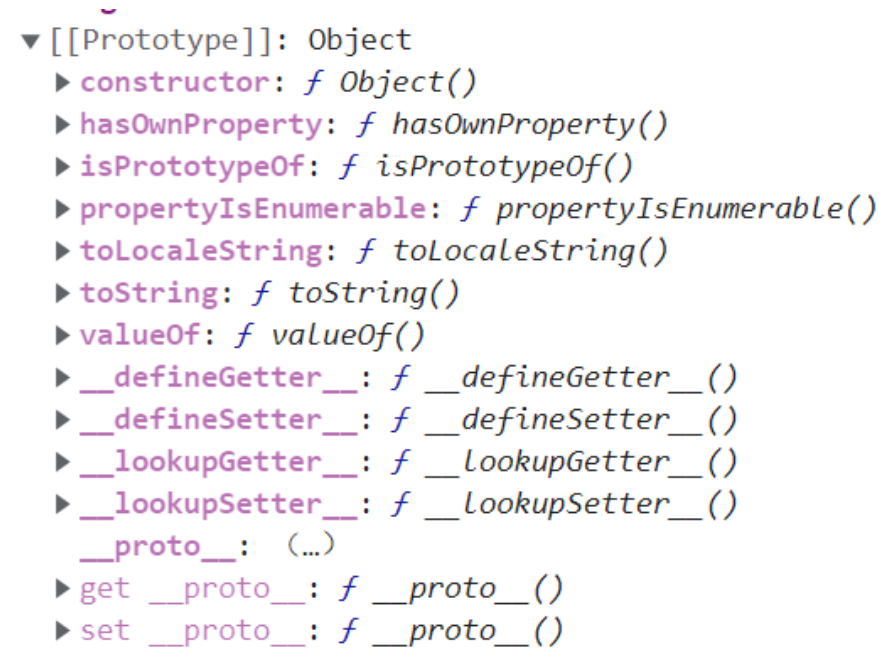
对于可迭代对象和迭代对象的转换，大家可以去官网详细了解，这里不展开说，官网地址：iterator。

当然如果非要只用扩展运算符转化objArray，可以先将它转为可迭代对象。

可迭代对象：



不可迭代对象：



3.3 借用数组slice方法

slice方法是数组的一个原型方法，它可以在不改变原数组的情况下返回数组中的某些元素并形成新的数组。

官网解释：

> slice() 方法返回一个新的数组对象，这一对象是一个由 begin 和 end 决定的原数组的浅拷贝（包括 begin，不包括end）。原始数组不会被改变。

如果还不熟悉slice的用法的小伙伴请参考官网：slice

示例代码：

```
const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));
```

```
// expected output: Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));
// expected output: Array ["camel", "duck"]
```

那么它与我们的类数组转换有什么关系呢，为什么要叫做借用呢?

先来看看slice实现原理：

```
Array.prototype.myslice = function (start, end) {
    var result = new Array();
    start = start || 0;
    end = end || this.length; // this指向调用的对象
    for (var i = start; i < end; i++) {
        result.push(this[i]);
    }
    return result;
};
```

原理也非常的简单，但是大家注意里面有一个this，如果我们不改变this指向，它就是指向的调用对象。但是如果我们将指向的对象改为了我们的类数组对象会发生什么变化呢?

首先我们类数组对象有length，而且类数组对象也可以通过obj[0]的方式获取元素，在对照上述代码，我们就可以将巧妙地将类数组对象转为数组。

示例代码：

```
// 将字符串转化为真正数组
let str = "小猪课堂";
console.log("str", Array.prototype.slice.call(str)); // ['小', '猪', '课', '

// 将NodeList转为真正数组
let box1 = document.getElementById("box1");
console.log("childNodes", Array.prototype.slice.call(box1.childNodes)); // [

// 将其它标准类数组转为真正数组
let objArray = {
  0: "小猪课堂",
  1: "小猪课堂",
  2: "会飞的猪",
  length: 3
}
console.log("childNodes",Array.prototype.slice.call(objArray)); // ['小猪课堂
```

输出结果：

```
str ▶ (4) ['小', '猪', '课', '堂']
childNodes ▶ (5) [text, p, text, span, text]
childNodes ▶ (3) ['小猪课堂', '小猪课堂', '会飞的猪']
```

理解slice转化类数组的原理，主要就是理解this指向，以及slice是如何实现的。

补充：

通过上面使用slice转化类数组方法后，我们可以发散一下思维，那么是否数组的其它原型方法是否也可以借用呢?

答案是肯定的!

比如下面的操作方法：

- Array.prototype.push.call()
- Array.prototype.splice.call()
- Array.prototype.apply.call()
- ...

只是具体需要传递什么参数需要大家下来自己思考!

总结

我们这里总结了大概3种方法实现类数组对象转化为数组对象，但是如果大家仔细思考就会发现，远远不止三种。就比如使用call这种方式，我们就可以衍生出很多的方式，需要大家自己去实践。

转换方法	注意事项
Array.from()	ES6提供的方法，只能实现浅拷贝，比较推荐使用。
扩展运算符	不能使用在非迭代对象上，操作起来比较简单。
slice（call方法）	主要理解this指向问题，可以衍生出很多转化方法。