【前端面试】new做了哪些操作？手写一个n...

前言

在程序员的世界里怎么可能没有对象，没有对象我们new一个不就完事儿了吗？在Java这样的面向对象的语言里面，new的操作可以说是随处可见。在我们前端程序员的世界里，可能很多小伙伴还没有感受到new的魅力，但是随着TS、ES6等等应用广泛，new的操作也逐渐被应用起来了！

我们都说new一个对象，顾名思义new操作就是创建一个新对象，那么它是如何创建的？创建的对象有什么特点？创建的过程是什么？今天就来理一理！

1.代码分析

我们口说一千遍一万遍都不如一行代码来得实在！想要new的过程做了些什么，那么我们很有必要先搞清楚new的用法，都不会用new，那还何谈实现它呢？

示例代码：

```
<script>
  // 定义构造函数
  function Person(name, age) {
    this.name = name;
    this.age = age;
  }
  // 定义原型方法
  Person.prototype.say = function () {
    console.log("你好：", this.name)
  }
  // new一个对象
  let obj = new Person("小猪课堂", 26);
  obj.say(); // 你好：小猪课堂
  console.log(obj);
</script>
```

输出结果：

```
你好： 小猪课堂
▼Person {name: '小猪课堂', age: 26}
    age: 26
    name: "小猪课堂"
  ▼[[Prototype]]: Object
    ▶say: ƒ ()
    ▶constructor: ƒ Person(name, age)
    ▶[[Prototype]]: Object
```

上段代码是一个标准的使用new创建对象的过程，相信大家也非常熟悉。我们的Person就是大家常说的构造函数，其实它就是一个函数而已，只不过我们让他的作用和其它函数有一点不一样。其实我们平常声明class类也就是一个函数而已。

我们重点关注控制台的输出结果，我们有**两点重大发现**：

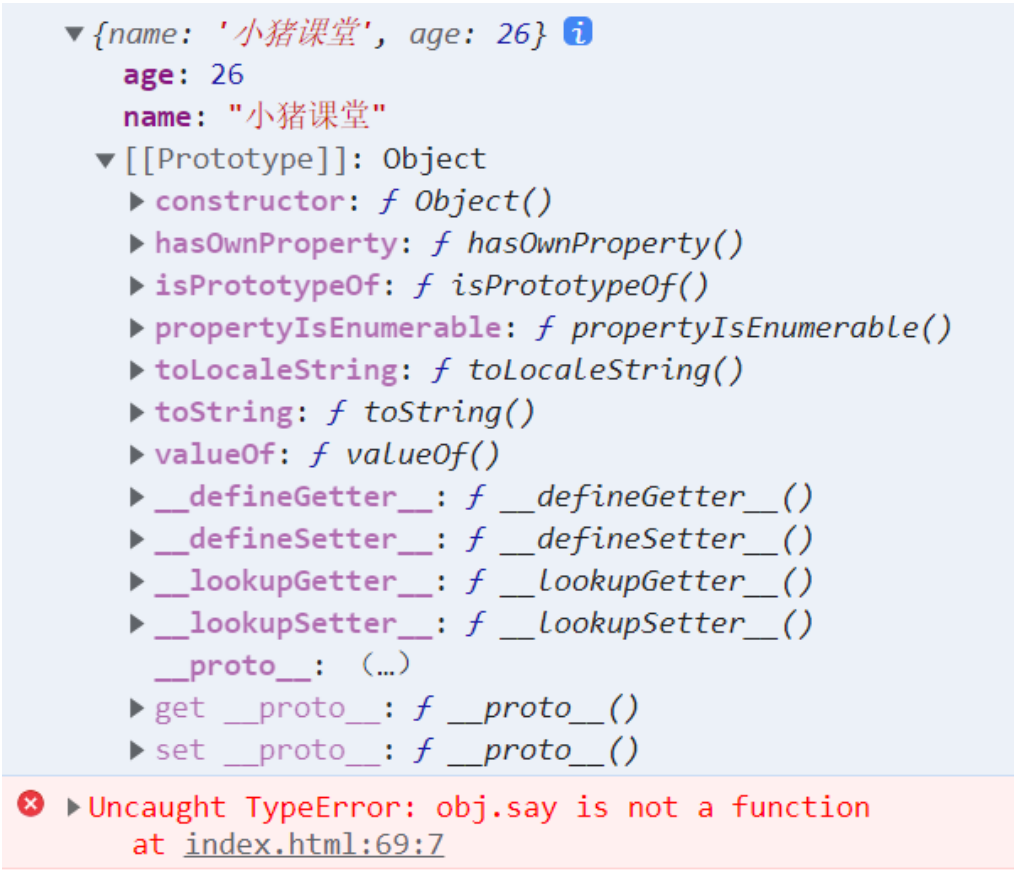
1. 我们new出来的对象可以调用Person的原型方法
2. 我们new出来的对象里面包含Person定义的属性

从上面不难看出，new操作无非就是新创建了一个对象，有些小伙伴可能会觉得会是Person返回的对象，那么我们是不是可以直接在Person里面return一个对象呢？我们实际操作一下。

修改代码如下：

```
<script>
  // 定义构造函数
  function Person(name, age) {
    this.name = name;
    this.age = age;

    return {
      name: this.name,
      age: this.age
    }
  }
  // 定义原型方法
  Person.prototype.say = function () {
    console.log("你好：", this.name)
  }
  // new一个对象
  let obj = new Person("小猪课堂", 26);
  console.log(obj);
  obj.say();
</script>
```

输出结果：



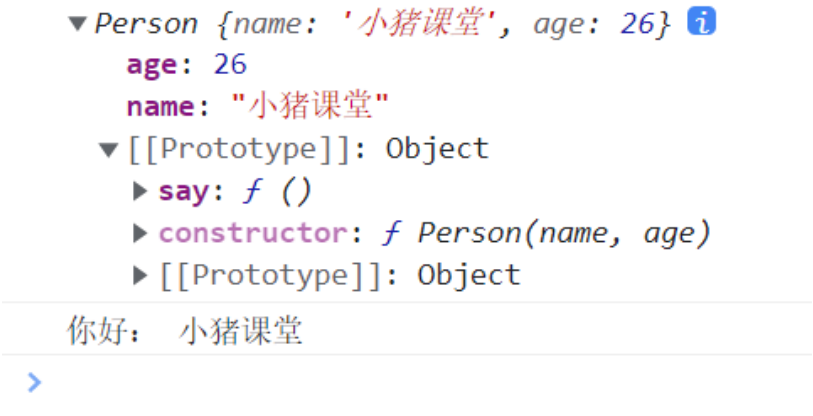
上段代码我们也没做什么修改，就在构造函数内部return了一个对象而已，可以结果却发生了巨大的变化。我们new出来的对象没有继承Person的原型方法了，new出来的对象也只是一个普通的对象了。

那么我们继续探索，如果在Person中返回一个值类型会是什么样的呢?

修改后代码如下：

```
<script>
  // 定义构造函数
  function Person(name, age) {
    this.name = name;
    this.age = age;

    return this.name + ':' + this.age
  }
  // 定义原型方法
  Person.prototype.say = function () {
    console.log("你好：", this.name)
  }
  // new一个对象
  let obj = new Person("小猪课堂", 26);
  console.log(obj);
  obj.say();
</script>
```

输出结果：



我们到这里又会发现一切都回来了！这说明我们返回非对象类型时，不会对new操作造成任何影响。

2.总结new做了啥?

看了上面的代码分析，我相信你对new的操作过程有了一定的感悟了吧！虽然你可能还有点模糊，但是不用着急，我们把new的这些特定以此总结出来，我相信你一定就不会感到模糊了！可能这就是只可意会不可言传吧！

老规矩，我们先来看看官方是如何解释new的。

官方解释：

> new 运算符创建一个用户定义的对象类型的实例或具有构造函数的内置对象的实例。

其中官网还对new关键字所作的操作做出了如下的解释：

> 1. 创建一个空的简单JavaScript对象（即{}）；
> 2. 为步骤1新创建的对象添加属性__proto__，将该属性链接至构造函数的原型对象；
> 3. 将步骤1新创建的对象作为this的上下文；
> 4. 如果该函数没有返回对象，则返回this。

官网的解释其实还是比较通透明了了，但是为了让小伙伴们更加容易理解，我们可以结合我们上面的代码在总结一下。

咱们的总结：

> 1. 首先会创建一个新的空对象，如我们上面的obj。
> 2. obj对象会继承构造函数即Person的原型，即obj.__proto__ === Person.prototype，这样obj就可以调用Person上的原型方法了。
> 3. 将Person的this指向obj，也就是将Person上的属性添加到新的obj对象上去，obj则可以调用Person中定义的属性了。
> 4. 如果Person构造函数没有返回对象，则返回新创建的对象（即this），否则返回return的对象。

到这儿我们应该就大概明白了一new的操作过程主要做了哪些步骤，如果你还有点云里雾里，这里在给大家说一下另一种理解：new String()大家应该都知道是在做什么吧，它返回了一个新的string实例对象，我们的new Person()也是返回一个新的person实例对象，只不过String是内置的罢了，所以我们new操作就是创建一个用户自己定义的对象类型的实例，只有如下两步：

1. 通过编写函数来定义对象的类型，比如new String()定义string类型。
2. 通过new来创建对象的实例。

3.实现一个new方法

核心原理我们都已经了解了，那还有什么理由不去实现它呢？我们根据前面总结的特点一步一步去实现它。

代码框架：

```
<script>
  // 手动实现new方法
  function myNew(constructor) {
    if (typeof constructor !== "function") {
      throw "myNew方法的第一个参数必须是一个方法";
    }

    // 返回新对象
    return newObj;
  }

  let obj = myNew(Person, '小猪课堂', 26);
  console.log(obj);
  obj.say();
</script>
```

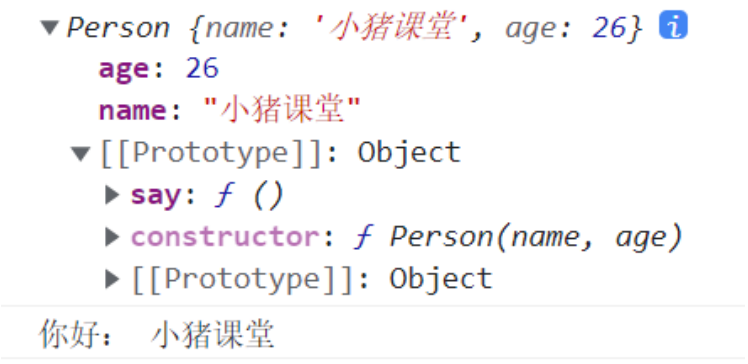
管它三七二十一，我们先把架子搭起来，我们的myNew方法最终肯定是要返回一个对象，因为new操作就是返回一个新的对象。

代码填充后（完整代码）：

```
<script>
  // 定义构造函数
  function Person(name, age) {
    this.name = name;
    this.age = age;

    return this.name + ':' + this.age
  }
  // 定义原型方法
  Person.prototype.say = function () {
    console.log("你好：", this.name)
  }

```

```
  // 手动实现new方法
  function myNew(constructor) {
    if (typeof constructor !== "function") {
      throw "myNew方法的第一个参数必须是一个方法";
    }

    // 基于constructor的原型创建一个全新的对象
    let newObj = Object.create(constructor.prototype);

    // 获取传入的参数
    let args = Array.from(arguments).slice(1);

    // 执行constructor函数，获取结果，并将属性添加到新对象newObj上
    let result = constructor.apply(newObj, args); // 将this指向newObj

    // 判断result类型，如果是object或者function类型，则直接返回结果
    let originType = Object.prototype.toString.call(result); // 获取内部属性值
    let isObject = originType === '[object Object]';
    let isFunction = originType === '[object Function]';
    if (isObject || isFunction) {
      return result;
    } else {
      // 返回新对象
      return newObj;
    }
  }

  let obj = myNew(Person, '小猪课堂', 26);
  console.log(obj);
  obj.say();
</script>
```

输出结果：



可以看到上面的输出结果和我们使用new关键字输出的结果一致了。想要理解上面代码，我们需要将new操作的步骤和我们的代码一一对应，确定我们每一步都在做什么。

这里有几点难理解或者需要注意的地方：

1. Object.create()方法：该方法会创建一个新的对象，传入的参数是我们需要绑定的原型，上段代码中的这步操作其实就是实现了new关键词的两步操作：首先创建新对象，然后赋值原

型给新对象。

2. constructor.apply()方法：该方法用来改变this指向，也就是我们将this指向了我们新创建的对象，这样Person方法中的this其实就是newObj，这样newObj就会拥有Person方法的属性了。
3. Object.prototype.toString.call()方法：这个方法主要是用来判断Person方法执行后返回的结果是什么，我们在前面实验过，new操作的时候，如果构造函数返回的是一个对象，则直接返回return的那个对象，初次之外，function也是一种特殊的对象。所以我们这一步就是为了解决这个问题。

总结

new了这么多次对象了，我们今天总算搞明白它的原理了。从我们实现结果来看，稍微难理解的就是原型和this指向这两块问题。

所以说强烈建议小伙伴们好好学一学原型、原型链和this指向。