在学习编程的时候，你一定或多或少了解结过oop（Object-oriented programming，面向对象编程），它是一种编程思想。指的是：将一类东西的一些共同特性或行为抽象出来，封装到一个object中。进而提高代码的可复用性，灵活性和可扩充性。大学期间的Java课本上说oop编程思想具有：继承、封装、多态三大特性。这些概念都是比较抽象的东西，言语上简短的解释很难让你认识到它究竟是个什么东西，下面我们通过Javascript的代码来体现oop思想的三大特性。

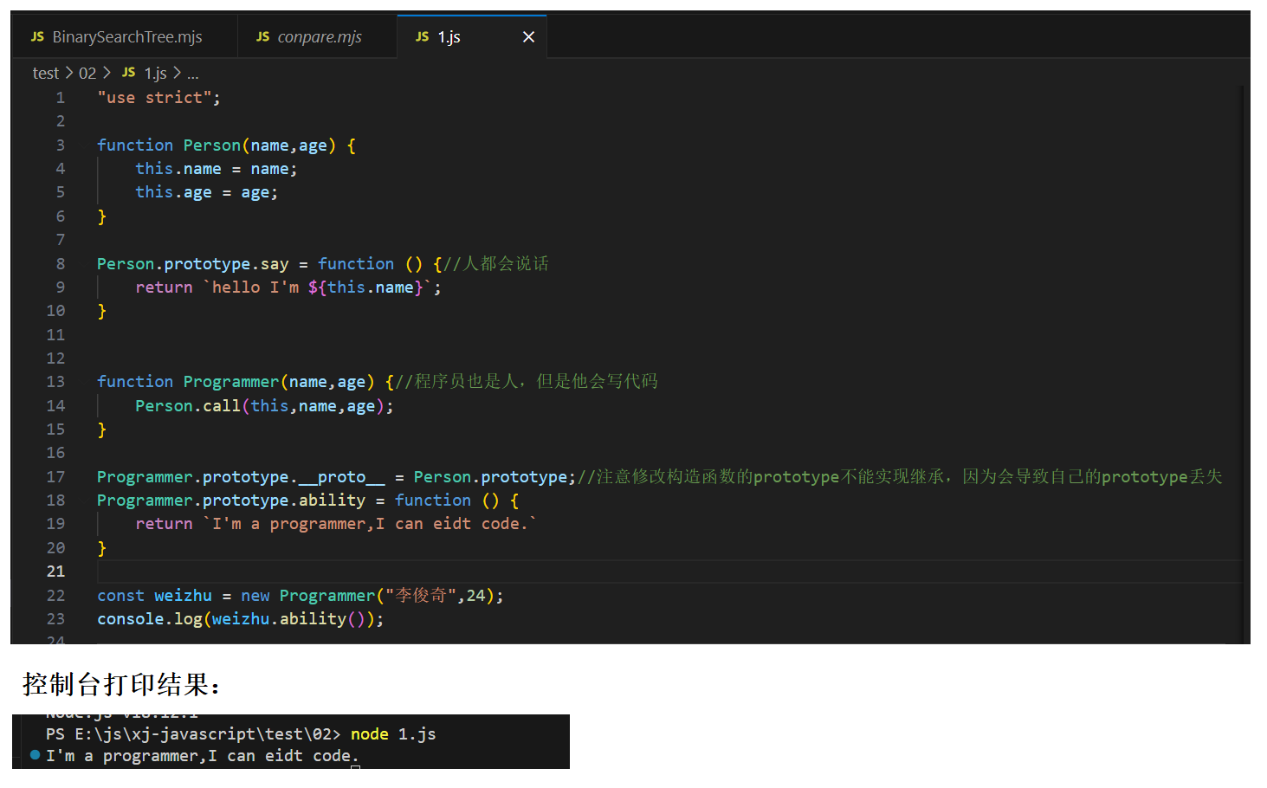
值得注意的是在es6之前，Js是通过prototype原型链的方式来实现oop的。我们知道在es6之后，其实你可以不必这么麻烦。但是prototype仍旧是Js的一个比较重要的概念。

使用Js实现oop思想三大特性中的“封装”



简单的封装出一个Person构造函数，在Js中构造函数是实现oop编程思想的基础。

使用Js实现oop思想三大特性中的“继承”

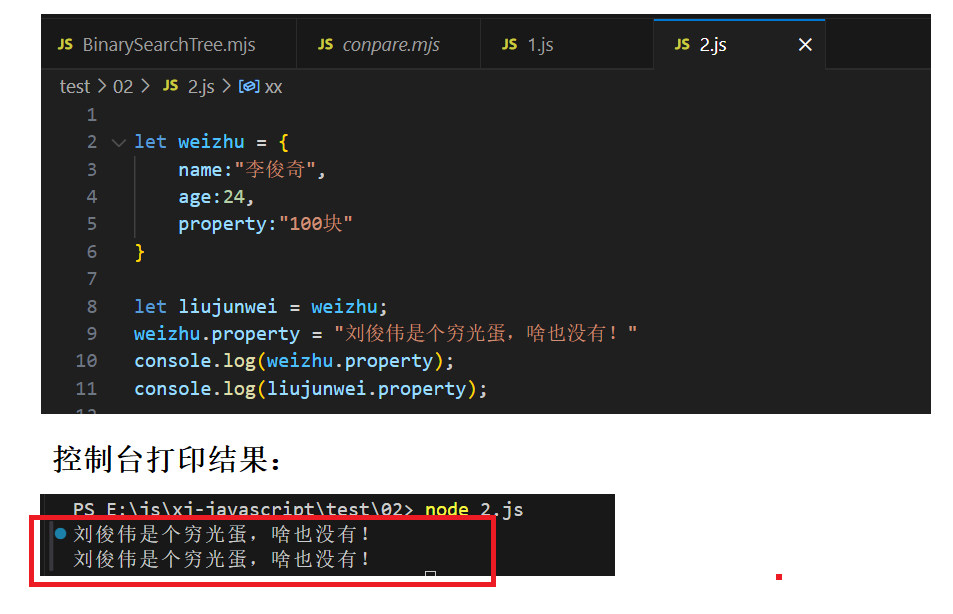


通过将一个构造函数的原型\_\_proto\_\_指向另一个构造函数的prototype的方式实现继承。

在Js中普通函数和构造函数的区别很模糊，因为Js给每个函数都设置了一个construtor，只有当你使用了new关键字的时候，才会用到其中的constructor。一个大家都遵守的规矩是，构造函数的首字母大写，而普通方法的方法名则不用大写。

看完上图的代码后，相信你肯定很好奇为什么是通过将构造函数的prototype.\_\_proto\_\_指向另一个构造函数的prototype来实现，而不是直接将一个构造函数的prototype指向另一个构造函数的prototype来实现的呢？道理很简单，因为“prototype”指向的是一个对象（这个用词不是很准确，有点怪，准确的说应该是一个“{}”封装的内容，这里我们就称它为对象好了），这个对象中有这个构造函数的constructor和你封装的属性和方法，如果你直接使用“=”赋值操作，就相当于将这个构造函数的prototype指向了另一个构造函数的prototype,这样会导致自己的prototype里的属性和方法丢失，这也不是真正的继承。

我举个简单的例子你就能明白：



在Js中对象是弱引用类型的数据，使用“=”并不能将一个对象真正的复制给另一个指针，也就是说liujunwei与weizhu指向的是同一块内存空间，所以打印出来的结果自然是都相同的。有没有办法可以实现：“不改动原有对象，同时将另一个对象的内容复制过来呢（哈哈哈，其实这样挺无礼的）？”有！比较复杂，这里我们就不讨论了（es6后Js给提供了一个api叫structuredClone(),可以实现对象的复制）。但在es6之前，Js给我们提供了一个\_\_proto\_\_,可以快速实现这种类似的功能（注意\_\_proto\_\_不是复制），\_\_proto\_\_的作用就是允许你在调用一个对象”{}”的方法的时候，如果这个对象没有，向上攀升（向你使用\_\_proto\_\_指向的个对象）寻找。我们来举个简单的例子：

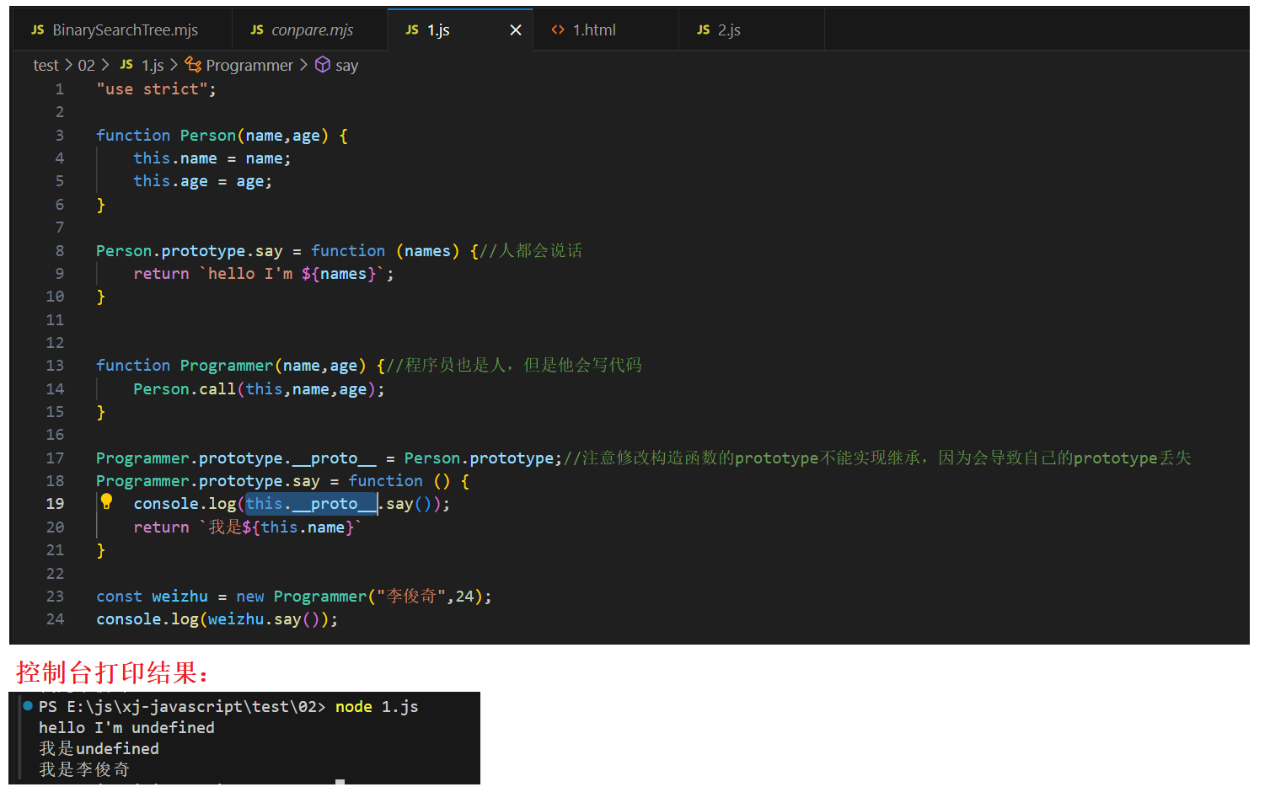


上图是\_\_proto\_\_用法的浅析

同时你观察上图，是不是发现super关键字可以与this.\_\_proto\_\_替换？哈哈哈，相信通过这个例子你能对\_\_proto\_\_和super关键字有一个浅显的认识。当然super关键字在es6 class中的用法与这里不太一样。

使用Js实现oop思想三大特性中的“多态”

对于oop思想的多态，我的理解是每个通过new关键字生成的实例，它们的方法和属性都是互不相干的(相当于生成了一个独立的对象object),这个比较好理解，另一方面我认为则是子类可以复写父类的方法。接下来我们通过代码演示一下：



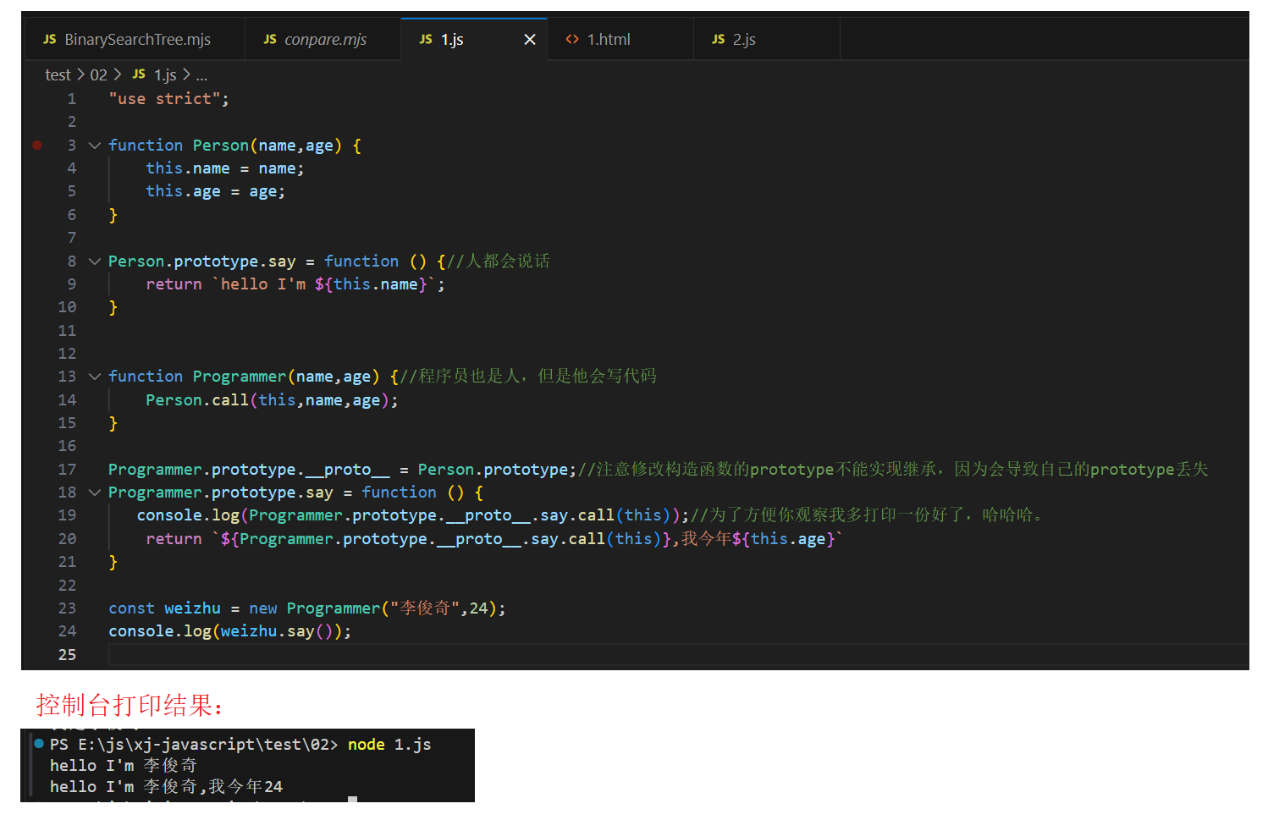
看到打印结果，相信你一定很奇怪，你并没有实现你所说的“子类可以复写父类中的方法”啊。为什么通过this.\_\_proto\_\_调用原型链上的方法会出现这样的结果？别急，这里我是先埋了一个坑，帮助你对Js的oop思想的实现方式有更深的理解。

接下来的知识，需要你对Js的callstack有一定的了解（这个不是什么很难理解的东西，事实上Java也有这个东西，它就是一种stack结构，遵循“后进先出”的原理，用于解决函数之间的调用关系和执行上下文的）

首先你要搞懂，log中的this指向谁？（图中第19行代码） this指向的是weizhu,那么weizhu.\_\_proto\_\_指向的是谁呢？是Programmer.prototype!（实例的\_\_proto\_\_指向的是构造函数的prototype,构造函数也是个对象），所以这里遇到weizhu.say()的时候，Js会先在调用栈中压入这个say（weizhu.say()）方法，然后检查函数中是否有调用其它方法和可执行代码。当碰到console.log(this.\_\_proto\_\_.say())的时候，它又会在调用栈中压入一个方法，是Programmer.prototype.say()（哈哈哈，你可能又好奇了，这tm不成了自己调用自己，不成了回调地狱了？），如果你有这种想法说明你对Js调用栈和this的指向还不太了解。（普通函数的this指向，与函数的调用方式有关）,Programmer.prototype.say(),会在调用栈中又压入了一个say()方法，而其中的this应当指向的是Programmer.prototype，那它其中又有你写的log代码,所以它又会向调用栈中压入一个this.\_\_proto\_\_.say()方法,但此时调用的就是Person.prototype.say()了（因为先前你设置过Programmer.prototype.\_\_proto\_\_指向Person.prototype）,所以其中this的指向是Person.prototype。又由于Js调用栈“后进先出”的特性，所以它们打印的顺序应当是上图所示结果一致。我们来画个图方便你理解：



通过上面的分析你应该能知道，如果子构造函数（子类）上的方法与父构造函数（父类）同名时，子构造函数（子类）的实例会先从自己的构造函数的prototype上找有没有这个方法，如果有，就调用自己的构造函数的prototype上的方法，而不会去调用父构造函数prototype的同名方法。但有有些人会好奇，难道就无法实现在子构造函数（子类）的prototype方法中调用父构造函数的prototype上（父类）的方法吗，要怎么才能实现呢？万一我就是想像java一样，通过调用父类的方法得到的结果，然后再重构子类方法呢？哈哈哈哈，当然可以！我们来修改一下上图的代码就可以轻松实现！



通过call方法改变this指向的方式实现在子构造函数（子类）的方法中调用父构造函数（父类）中的同名方法。

讲到这里其实已经基本上讲完了在Js中是如何实现oop思想的三大特性的了，其实它的关键在于你对Js原型链以及callstack的理解。我很想一次性把这两个概念讲清楚，但是受限于篇幅问题，加之本文主旨并不是讨论Js原理的文章，所以这里就草草带过了。但是在后续写tank项目的时候，你大可不必这样麻烦，因为Es6给我们提供class语法糖，方便你快速实现这些操作，加之我们有ts给我们提供的静态类型支持，可以让你快速的完成tank大战项目的开发。