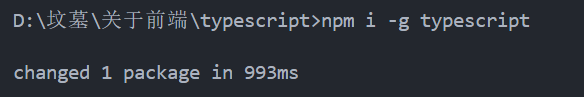
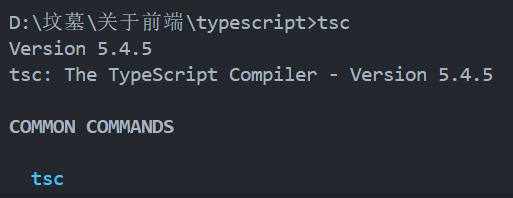
# TypeScript

Typescript是一个JavaScript的超集，如果理解的话就可以理解为Type+Script。Typescript出现的目的就是为了解决JavaScript的弱类型的问题。之前在JavaScript介绍时JavaScript是一种弱类型的，变量只有let const 和var，var还基本废弃了。这也就导致了他包容性很强。初学是没有任何问题，甚至可以说比其他强类型的语言来说，更使新手舒服。但是其弱类型也会使项目的维护出现很多的问题，维护成本甚至会高于开发成本。为了解决这个问题typescript诞生。Typescript是基于JavaScript进行的一个扩展。其通过npm包的形式，通过对typescript代码进行编译生成JavaScript。个人理解就是通过typescript这个过程把JavaScript中可能会出现的问题过滤掉然后再生成JS这样代码执行的还是JS但维护的却是Typescript，本来是没打算学了，但是vue3已经用typescript重写了。还是要学的，typescript更类似于Java。下面对typescript的学习进行一个记录。

安装typescript

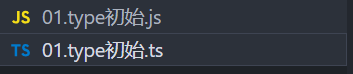




安装成功后就可以编写代码。

代码编译后可以生成JS文件，编译方法由于是-g所以很明显了。



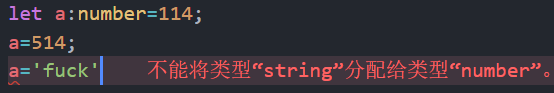
，

Typescript和JavaScript的关系和CSS与less的关系非常的像。

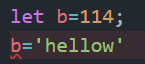
首先是用typescript解决JavaScript的类型问题。

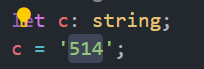


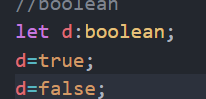
Typescript在JavaScript原有的number,string,null,undefind,bigInt,boolean,symbol，object几种数据类型上做出了一个新的扩展。重新规定了数据类型，如上图。下面一一通过代码示例进行示范。



这块代码里我们通过number类型实现了对a类型的规定，当我们对a进行赋值时不符合类型就会报错。其实他有自动解析功能。下面代码就实现了自动解析。所以我们如果声明的同时赋值了可以不写类型规定，他会自己解析出来。

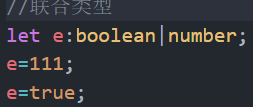
当然如果不改的话编译也是能进行的除非进行了一些编译设置。

string类型是这样的与number没有很大的区别。

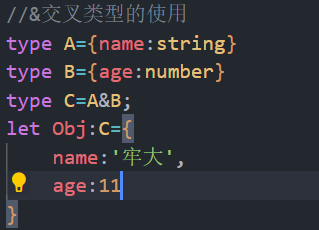
boolean值也是这样但如果我这样写就会出现问题了。

Typescript里面他把123解析成了number。但是我们在JavaScript里面123是等同于true的，但这里他不认识。所以尽量不要这样写。

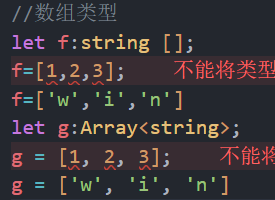
TS中提供了一个叫联合类型的东西，这个东西个人感觉是介于强弱类型，动态静态语言直接的一个东西。

看到这里正常人就都明白了，他就是一个动态静态语言之间的产物。这个|是介于两者之间的作用，可以理解为JS重的||但切记这俩不是一个东西。

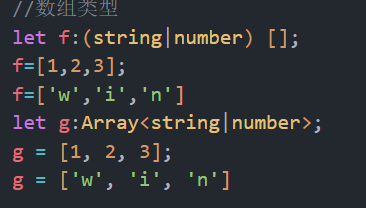
TS中的&也是一个数据类型，叫做交叉类型。主要是用于将两个类型合成一个类型的，代码示例如下：



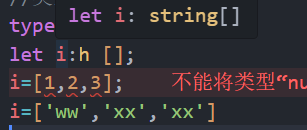
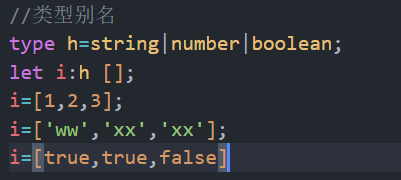
数组类型如下：

数组类型有两种写法第一种比较正常，第二种就更类似于Java中的泛型。

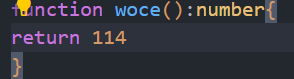
数组类型可以规定数组中数据的类型，但是我用number写的无法进去string数组。其实这里可以用到联合解决。如下面这样就不会报错了。

推荐使用类型 []写法

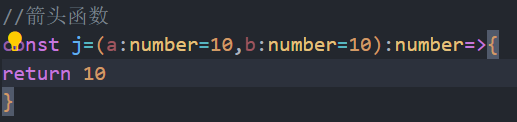
我们可以给类型起别名。代码如下：

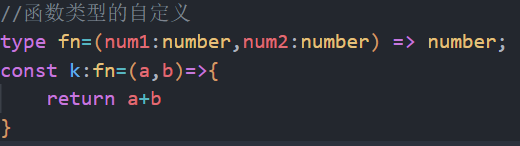
这里可以发现i是string类型的也就是说h===i了这就是起别名。如果只是这样写就有点多此一举了，所以他通常用于一堆一起写就像下面这样。多用于联合省事用。

他给函数也提供了一系列的规定，主要体现在返回值与参数里。

这个number是规定返回值的这样就实现了返回数据类型的确定。

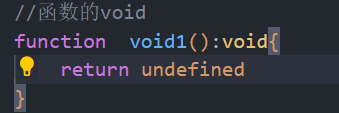
这么一对比就知道我们的返回值已经规定成功了。

箭头函数也可以设置。下面代码就是箭头函数的设置。



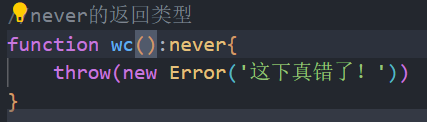
我们箭头函数是这样的。

函数的void返回值

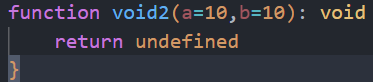
void的意思就是没有返回值，要么不写return要么return undefined。要么写个return 后面不跟值。

函数never返回值

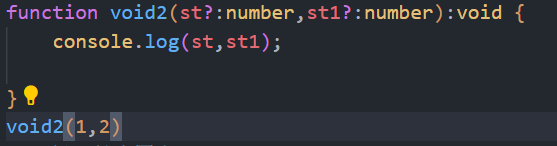
这下是真的永远都不用返回了

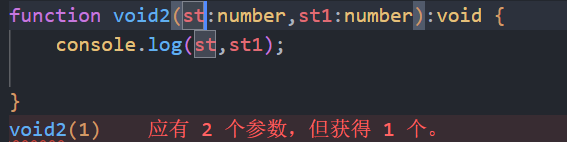
这种完全不需要返回值的用never就行了。

函数有可选赋值的也有默认值

默认值老生常谈不说了

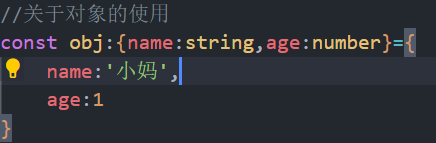
可选值





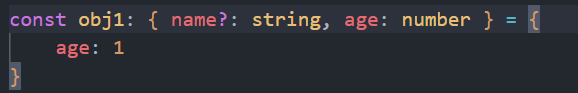
可选值是指在变量后面加问号，这样的方法可以使参数传入不再受很大的限制。

关于对象的使用



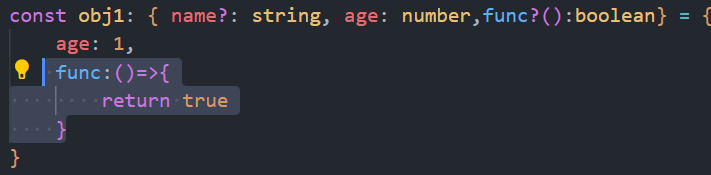
对象的用法就是声明键对类型，赋值键对值。

同样可以与可选值结合

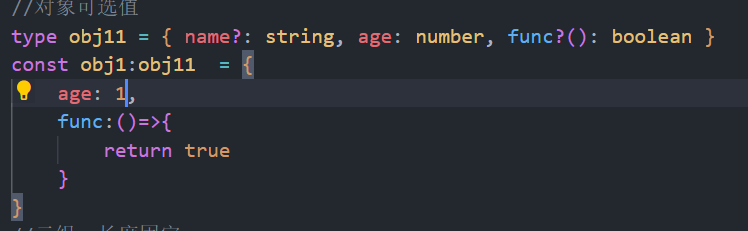


这样可以实现对象可选值。

也可以加入方法。通过函数名加返回值的方法加入方法。

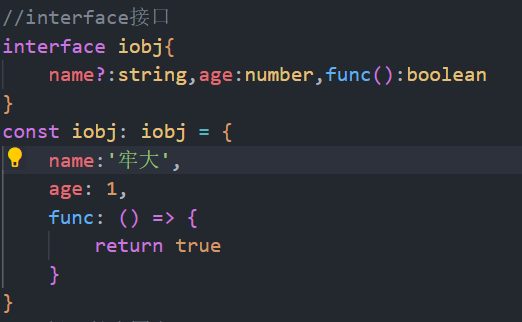


其实这种一大串的类型就可以用自定义解决。



接口

接口这个概念在Java里面也是有的， 他与用type定义对象别名很像。



非常像啊。与type定义对象别名差别就一点点了。

但如果细说的话接口与对象别名还是有一定的区别的：

相同点

1. 都可以给对象指定类型

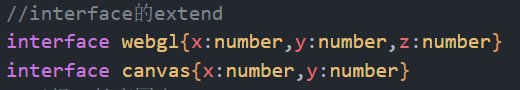
不同点

接口只能给对象用，且他可以继承

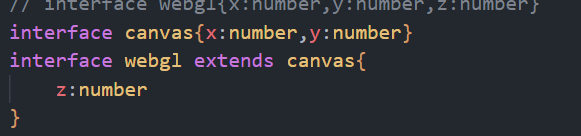
对象别名不但可以给对象用，可以给任何类型用，且他不能继承。

接口继承

在多个interface里面有很多相同的属性，重复写就很麻烦。



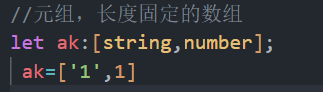
像这样xy就都重复了。



通过extends继承可以少些重复项。提高代码的复用性。

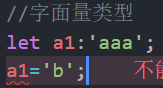
元组Tuple

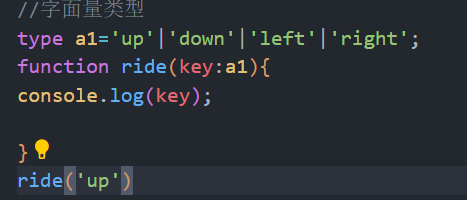
JS中的数组没有固定长度的，如果我们数组长度已经固定可以用元组优化效率。

这就是一个元组，个人感觉就是一一对应。

字面量类型

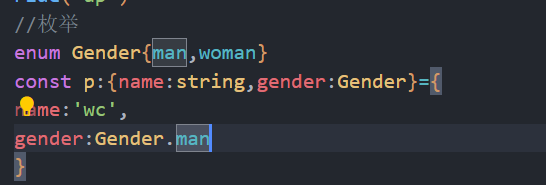
这个玩意通常和联合用。与const非常像

，一但赋值给字面量类型就不能再改了。可以看成const a1=’aaa’；

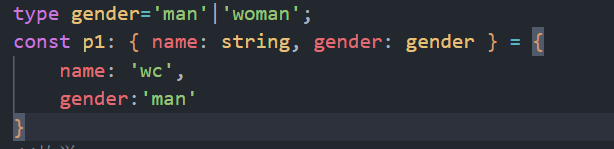
通常用于可限定范围内的东西。给特定值。

枚举enum

他的功能好像大部分都被type替代了。但还是有一些特定作用没被取代。



枚举例子如上。

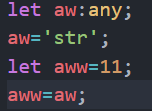


Type字面量配联合变量例子如上

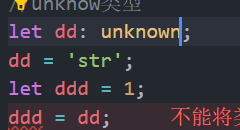
同样的效果用字面量配上联合写可读性更强一点。

any类型

JavaScript中的数据类型就是any类型的如果我们定义变量没有确定类型。就会默认变成any类型。这样typescript就会变成anyscript变得没意义了。同时any还很有感染力。

这串代码不报错，但明眼人都能看出来我把str赋值给了一个number变量。纯纯的抽象写法。如果真不知道类型可以用unknown来代替。至少他不会感染其他类型的变量。造成抽象结果。

Unknown类型实例如下

unknow类型自己玩自己的不去霍霍别人。

类型断言as

很多时候程序员比typescript懂数据类型。比如说我们通过DOM操作拿到一个元素，他是没有默认数据类型的。这时候我们就要用类型断言给他加一个数据类型，这样写代码更加严谨。

这里我把上面的呢个any类型的数据断言成了string。类型断言就是这么用的。

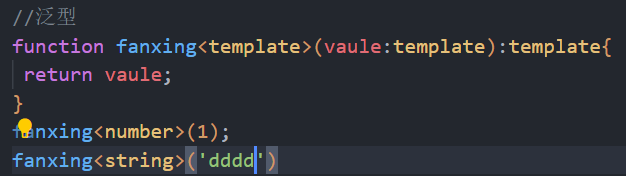
Typeof

根据变量值反向拿到变量类型，比如下面这块代码

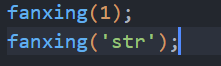
这里就拿到了aaa的类型，算是一个反向操作。

泛型

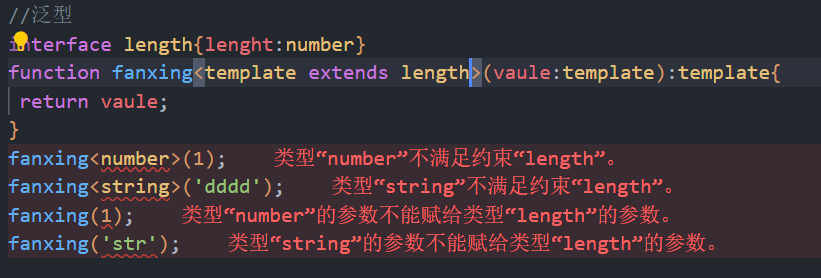
其实和Java的泛型完全一样，类似于cpp的模板。



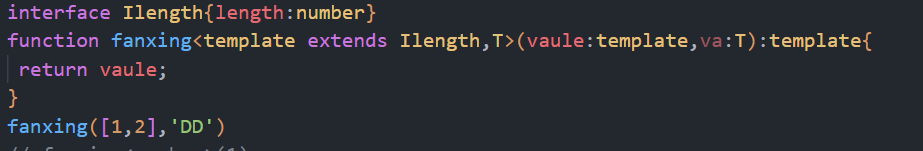
这是基础写法。其实可以用类型推断来减缓泛型。

他自己会推断出数据类型，可以用这个办法简化。

泛型约束可以对传入的泛型约束的接口帮助约束。

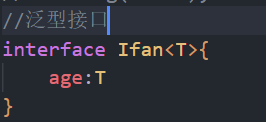


这样就约束了泛型必有length属性。其实就是约束成了数组。



可以同时用多个泛型。

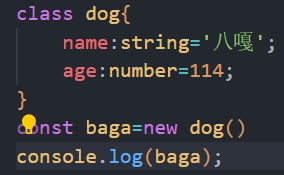
泛型接口

这就是泛型接口了，泛型运用于接口就叫泛型接口。

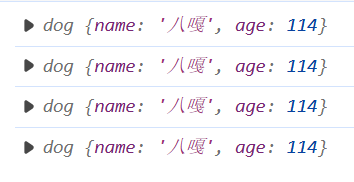
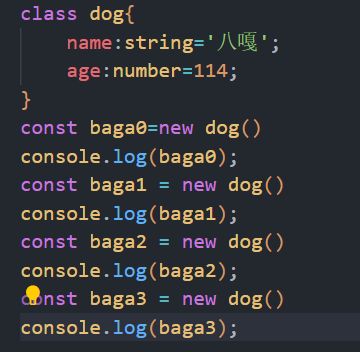
**面向对象**

这个东西其实是所有编程语言里应该都有，面向对象的语言里都是万物皆对象。其实学过很多语言，面向对象这个概念其实很简单。就是把现实里的东西里给放到程序里。现实里的人放入游戏里变成角色。那么角色就会是一个对象，对象必有两个东西。属性，方法，我们的程序正是对于这个对象进行操作进行的。浏览器可以看做是一个报纸的对象，这样就很明了了。

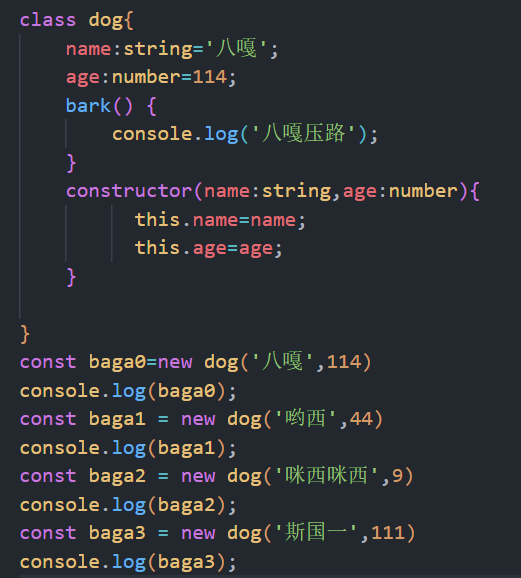
创建类

这里就算创建了一个日本狗的对象并且进行实例化编译后在控制台打印。这就是打印结果。

但是作为类我们实例化的对象都是同一只日本狗肯定是不行的。



这里我们就需要构造函数这个东西帮我们创建不同的日本狗，其实这个是ES6的语法typescript更强了，之前没写就是了。

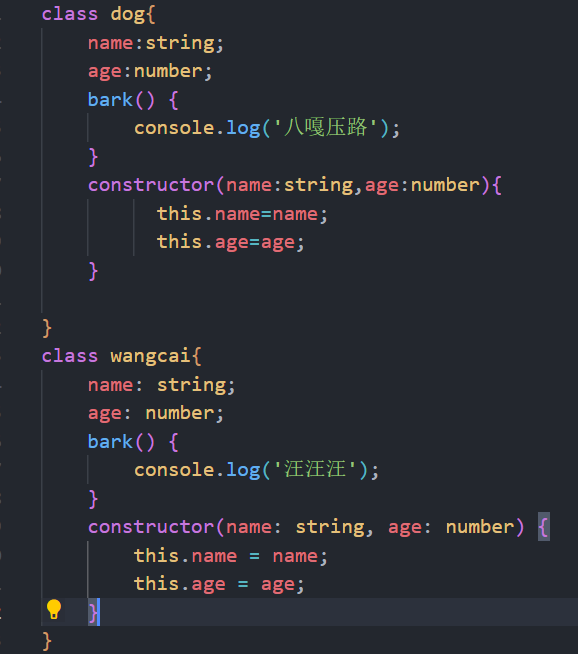


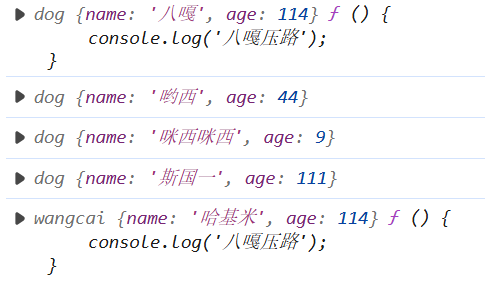
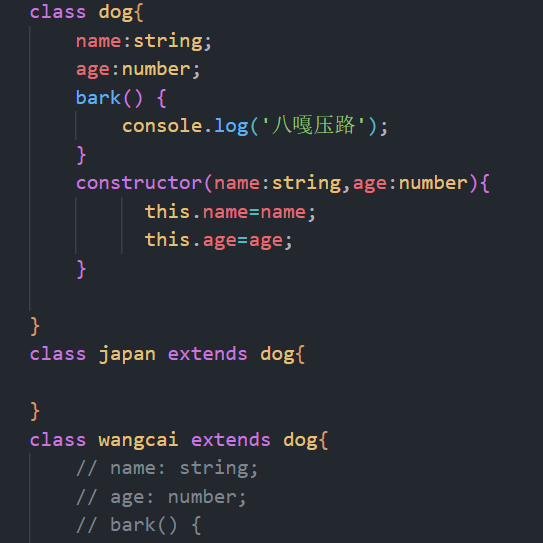
我们通过构造函数传参实现了日本狗实例的创建。而且是四个完全不同的日本狗。

我们也可以使用.方法实现日本狗狗叫的的调用。非常的方便。上面代码出现了this的指向，this指向在任何地方都是非常敏感的也是面试高频，这里需要写一下this指向性的问题，这里的this指向的是实例化对象的本身。比如说baga0里面的this就是指向的baga0所以我们用this.name来改实例对象中的值。

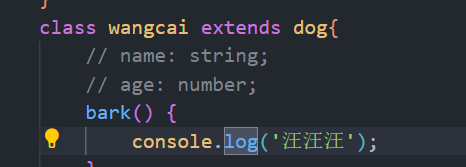
关于继承

上面写了一个日本狗，但现实里狗的品种不止日本狗一种。

上面的日本狗和旺财的内部构造基本是一样的，这时候如果多个狗我们每个都单独写一次复用性就太差了这时候继承就要发力了。

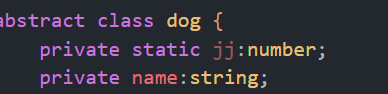


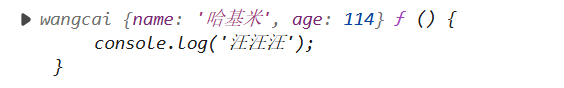
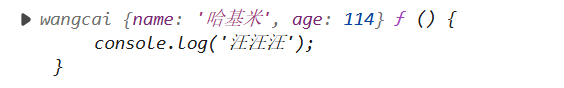
这段代码里我们用了继承的方法这样的话我们通过继承创建了两个对象，代码复用率这就高起来了。但是有一点不够完美就是旺财说的话是狗话不是狗都不如的语言。所以我们需要在接受继承的类里面对狗叫进行重写。

重写之后终于说对了话。

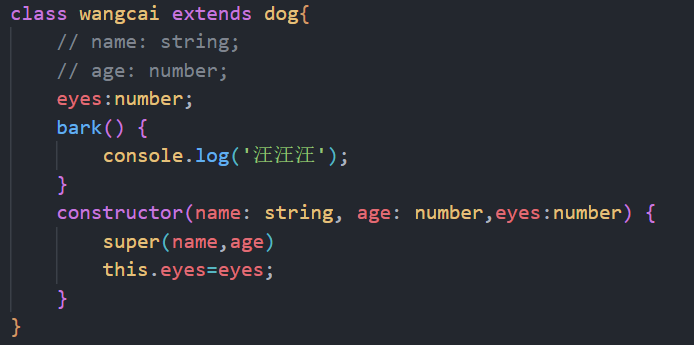
有些东西不能继承就像是有**static的静态属性无法继承。**



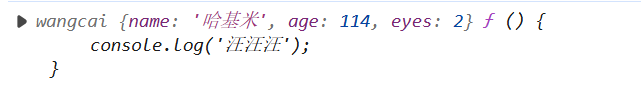




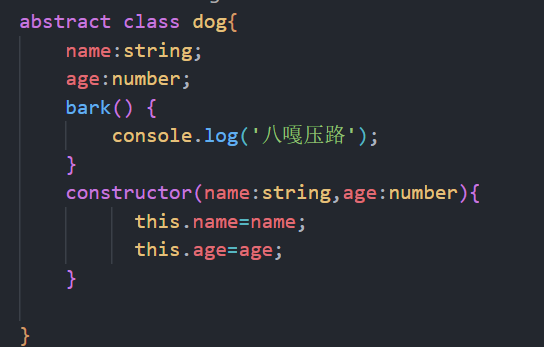
但是相似归相似但是总有不同的，总不能有一点不一样就换父类，这样太过麻烦所以我们用constructor来解决这个问题。





Constructor配合super实现了一个不完全与父类相同的子类。属性参数多了一个，控制台也不一样了。

很多时候父类是不参与实例化对象的，那我们不参与实例化对象的类最好给他锁死，只用于当父类用，这时候就出现了抽象类的概念，通过抽象类我们就可以把一个类锁死。



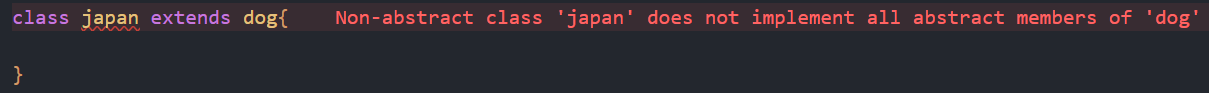


我们把类变成抽象类之后就没有办法创建实例了，也就是锁死了只用于当父类继承。

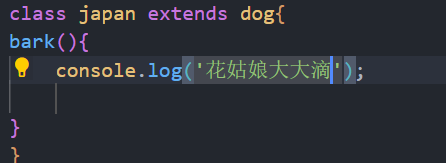




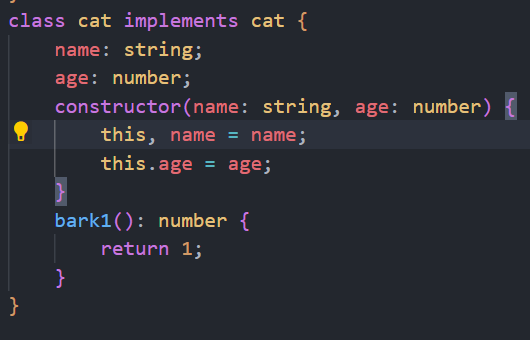
其实abstract也可以给方法使用，使用后方法必须在子类中重写，否则必将报错。



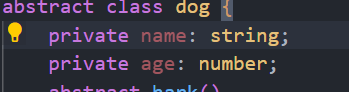
像这样就是没重写，重写之后就什么都解决了。如下：



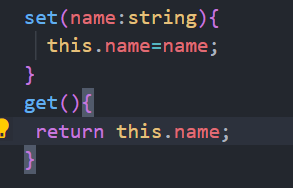
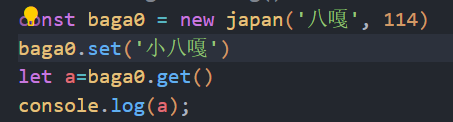
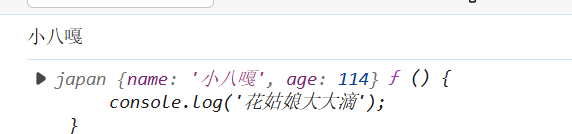
接口也可以用于继承。

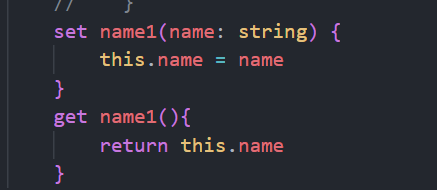
implement继承了接口的属性必须一一对应不然不行。

最后一点是属性封装，其实这个东西类似于JS的闭包，外部变量不能随意改变量内部的值，但是并不是改不了。这个知识点还涉及了public，private，protect这三个点。下面就是代码实例了。Public全局访问，private自己类内部访问，protect自己，子类孙子类访问。

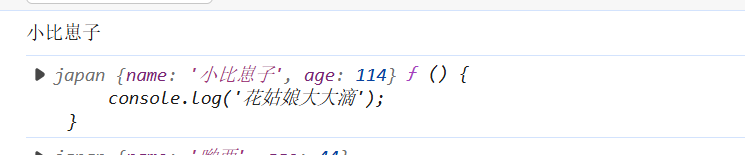
我们这里给属性加上了private属性。我们如果在外面改是没戏的

这就是一个报错。解决这个问题就得通过属性封装，封装成一个方法，通过调用方法来实现。所以说类似于闭包。

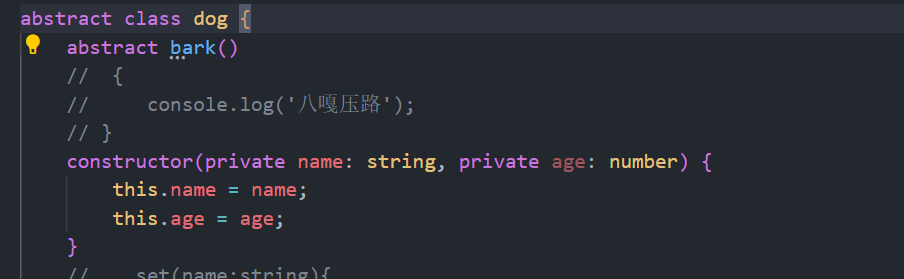
通过这两个方法，我们在外面可以读取改变内部变量。自然而然的实现了读写操作。他还有简写方式是typescript给我们提供的。





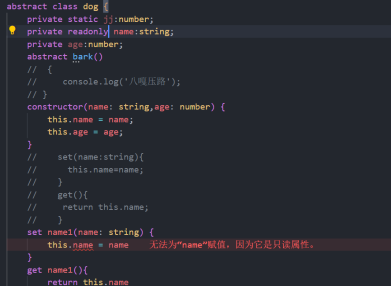


写起来不方便多少但是操作起来方便多了。可以直接.放大拿到或者更改。



可以把属性直接写在constructor里面相当于一种语法糖。

Readonly为只读

只读属性是不可更改的。