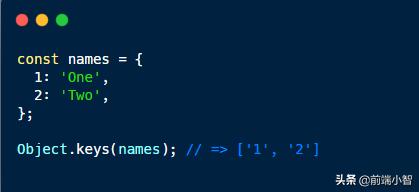
JS 普通对象 {key: 'value'} 用于存放结构化数据。但有一件事我觉得很烦:对象键必须是字符串(或很少使用的 symbol)。

如果将数字用作键会怎样？在这种情况下不会有错误：



JS 会隐式地将对象的键转换为字符串，这种默认行为丢失了类型的一致性，要解决也挺棘手的。但 ES6 中的Map 对象可以帮我们解决这类的问题，Look See See。

**1. Map 接受任何类型的键**

如前所述，如果对象的键不是 string 或 symbol，JS 将隐式地将其转换为字符串。

幸运的是，map 的键类型没有问题



1 和 2 是 numbersMap 中的键，这些键的类型（数字）保持不变。

可以在 mpa 中使用任何键类型:数字、布尔值、字符串和 symbol。



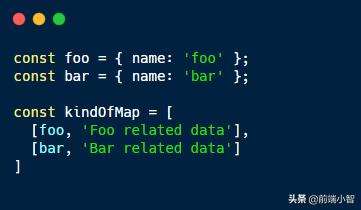
booleansMap 使用 booleans 作为键，没有问题。相反，布尔键在普通对象中不起作用。

来突破一下想象:是否将整个对象作为 map 的键，答案：可以的。

**1.1 对象作为键**

假设你需要存储一些与对象相关的数据，而不需要将这些数据附加到对象本身。使用普通对象是不可能的。

解决方法是使用对象-值元组数组：



kindOfMap 是一个数组，包含对象和关联值的对。

这种方法最大的问题是按键访问值的复杂度O(n)，咱们必须遍历整个数组才能通过键获得所需的值。



WeakMap (Map的一个专门版本)不需要这么麻烦就能做到上面的事情:**它只接受对象作为键**。

**Map** 和 **Weakmap** 之间的主要区别是，**Weakmap** 允许对键对象进行垃圾收集，从而防止内存泄漏。

好了，用 WeakMap 重构上面的代码就变得很简单了：

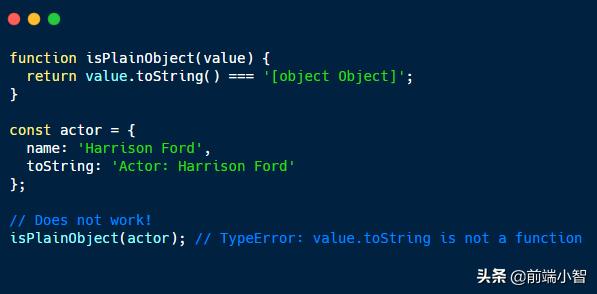


与 Map 相反，WeakMap 只接受对象作为键，并少了一些方法。

**2. map 对键名没有限制**

JS 中的任何对象都从原型对象继承属性，普通对象也是如此。

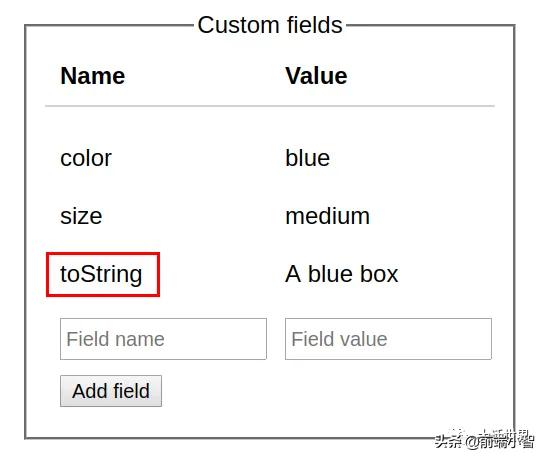
如果重写从原型继承的属性，则可能会破坏依赖这些原型属性的代码：



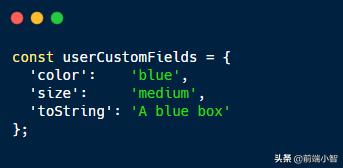
在对象参与者上定义的属性 toString 覆盖从原型继承的 toString() 方法。这中断了isObject()，因为它依赖于 toString() 方法。

检查普通对象从原型继承的属性和方法的列表, 避免使用这些方法名定义自定义属性。

例如，假设有一个管理某些自定义字段的用户界面。用户可以通过指定名称和值来添加自定义字段：



将定制字段的状态存储到普通对象中会很方便：



但是用户可能会选择一个自定义字段名称，例如toString（如示例中所示），构造函数等，这可能会破坏咱们的对象。

不要使用用户输入的值作为普通对象上键。

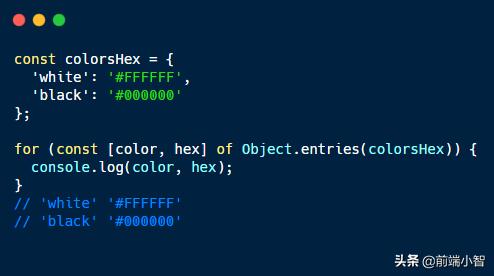
map 没有这个问题，键值名称不受限制：



不管 actorMap 有一个名为toString的属性，toString()方法都可以正常工作。

**3. map 是可迭代**

为了遍历普通对象的属性，必须使用其他的辅助静态函数，如 Object.keys()或Object.entries():



Object.entries(colorsHex) 返回从对象提取的键值对数组。

但是，map 本身是可迭代的：

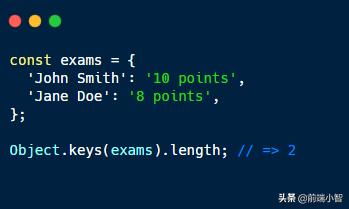


colorsHexMap是可迭代。可以在任何接受迭代的地方使用它:for()循环，展开运算符[...map]。

map 提供了返回可迭代方法：map.keys() 遍历键，map.values() 遍历值

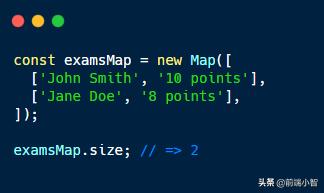
**4. map 的大小**

普通对象的另一个问题是，您无法立马知道它包含的属性的数量。



要确定 exams 的大小，必须通过所有键来确定它们的数量。

map 提供了 size 属性，表示属性的数量。



确定 map 的属性的数量更加简单：examsMap.size。

**总结**

普通 JS 对象通常可以很好地保存结构化数据，但它们也有一些局限性:

1. 只能用字符串或 sybmol 作为键
2. 自己的对象属性可能会与从原型继承的属性键冲突（例如toString，constructor等）。
3. 对象不能用作键

所有这些问题都可以通过 map 轻松解决。而且，它们提供了诸如迭代器和易于进行大小查找之类的好处。

不要将 map 当作普通对象的替代品，而应视为是普通对象补充。