对于let 是否存在变量提升的思考?

> 链接：https://juejin.cn/post/6844904050614353928#comment

面试的时候，我们经常会被问到let 、const 和 var 之间的区别，我的回答无疑总是那几点：

- **变量提升**

  var会进行变量提升，let和const不会进行提升

- **暂存死区**

  因为var会进行变量提升，所以可以在声明之前访问，不会形成暂存死区。

  let 和const 不会进行变量提升，在声明之前不能使用，形成暂存死区

- **重复声明**

  var可以进行重复声明，但是let和const不能进行重复声明

- **块作用域**

  var不会形成块作用域，let和const可以形成块作用域

- **重新赋值**

  var和let声明的变量可以重新赋值，const不可以。

  如果const 声明的变量存储的是引用地址, 是可以修改这个引用对应的对象的值的，但是这个变量不能被赋予其他值

每次都是这个样回答，面试官也没有说什么。

但有一次面试的时候，一面面试官问到我，let的所谓的暂时性死区怎么解释？let和const到底有没有变量提升？我没回答出来，然后面试官和我解释了一下，并且推荐我看一个大神的这篇文章：zhuanlan.zhihu.com/p/28140450

通过这篇文章，以及这篇文章所提到的文章得出的结论就是：

let 的「创建」过程被提升了，但是初始化没有提升。

接下来就讲为什么？从变量的生命周期开始！

1、变量的生命周期

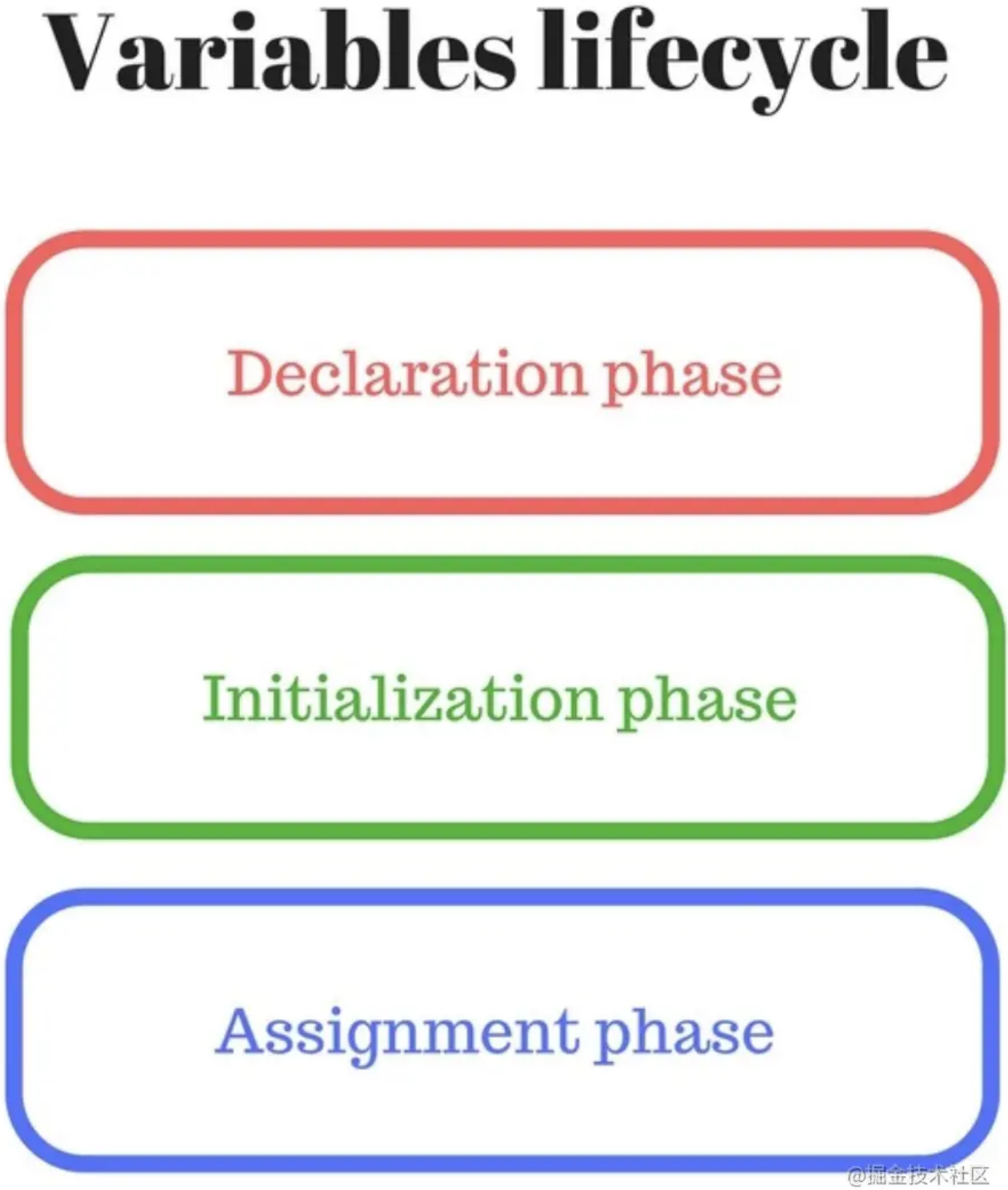
在这里我比较喜欢这篇文章：JavaScript Variables Lifecycle: Why let Is Not Hoisted

首先我们来学习一下变量的生命周期：

当引擎使用变量时，它们的生命周期包括以下几个阶段：

1. 声明阶段（`Declaration phase`）正在范围内注册变量。
2. 初始化阶段（`Initialization phase`）是分配内存并为作用域中的变量创建绑定。在此步骤中，变量将使用进行自动初始化undefined。
3. 分配阶段（`Assignment phase`）是为初始化变量分配一个值。

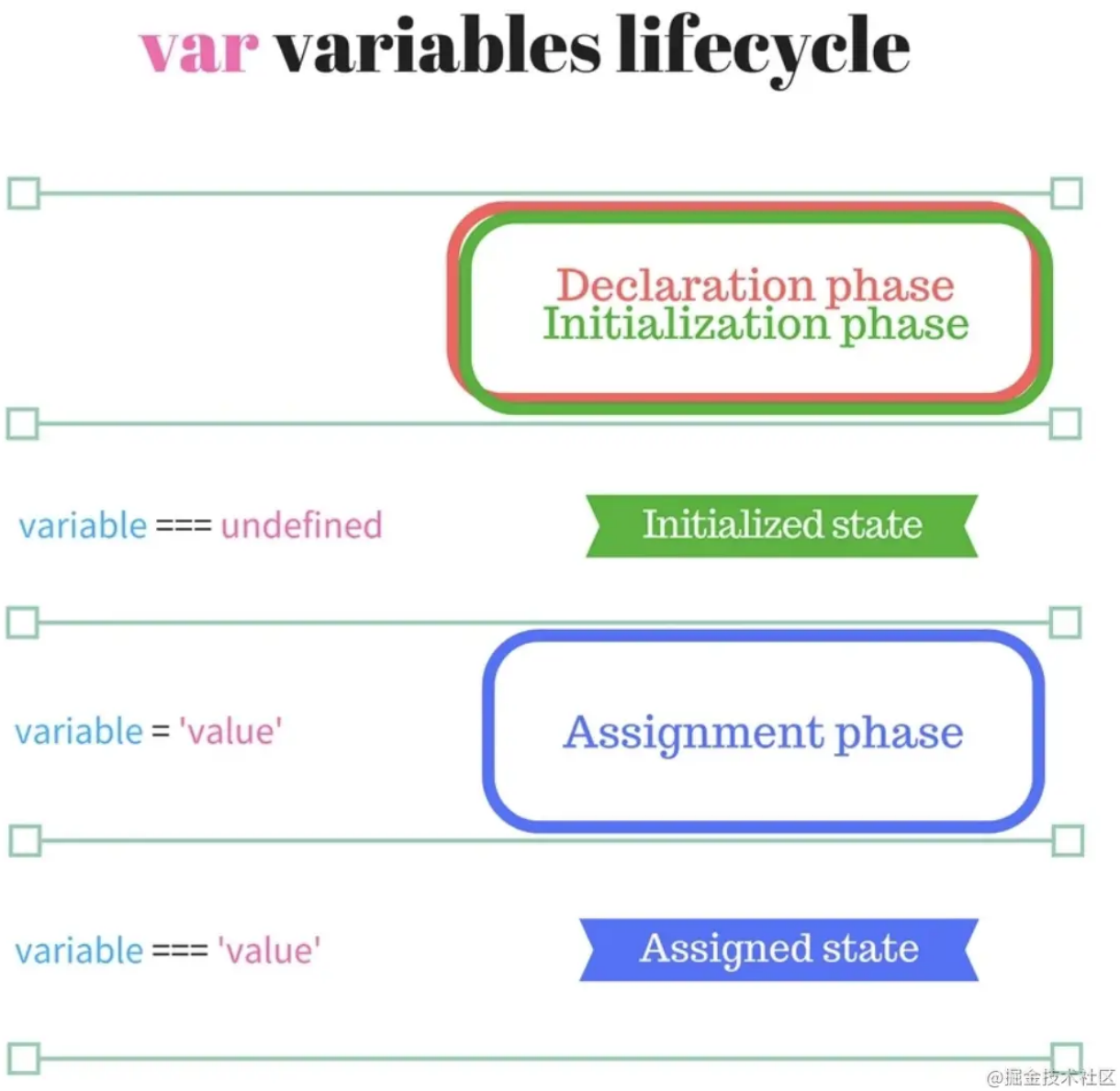
变量在通过声明阶段时已处于统一状态，但尚未达到初始化状态。



但是要注意的是，就变量生命周期而言，声明阶段与一般而言的变量声明是不同的术语。简而言之，引擎在三个阶段处理变量声明：声明阶段，初始化阶段和赋值阶段。

1.1 var变量的生命周期

var声明的变量在所在的作用域的开始处声明和初始化变量。声明和初始化阶段之间没有差距。




```
function var_variable(){
  console.log('在声明阶段之前',one_variable);
  var one_variable;
  console.log('在未赋值之前',one_variable);
  one_variable = 'be assigned';
  console.log('赋值之后', one_variable);
}
//console.log('在函数作用域外是否可以访问到var声明的变量', one_variable);//one_variable is
not defined
var_variable();
复制代码
```

运行结果：

```
在未声明阶段之前 undefined
在未赋值之前 undefined
赋值之后 be assigned
复制代码
```

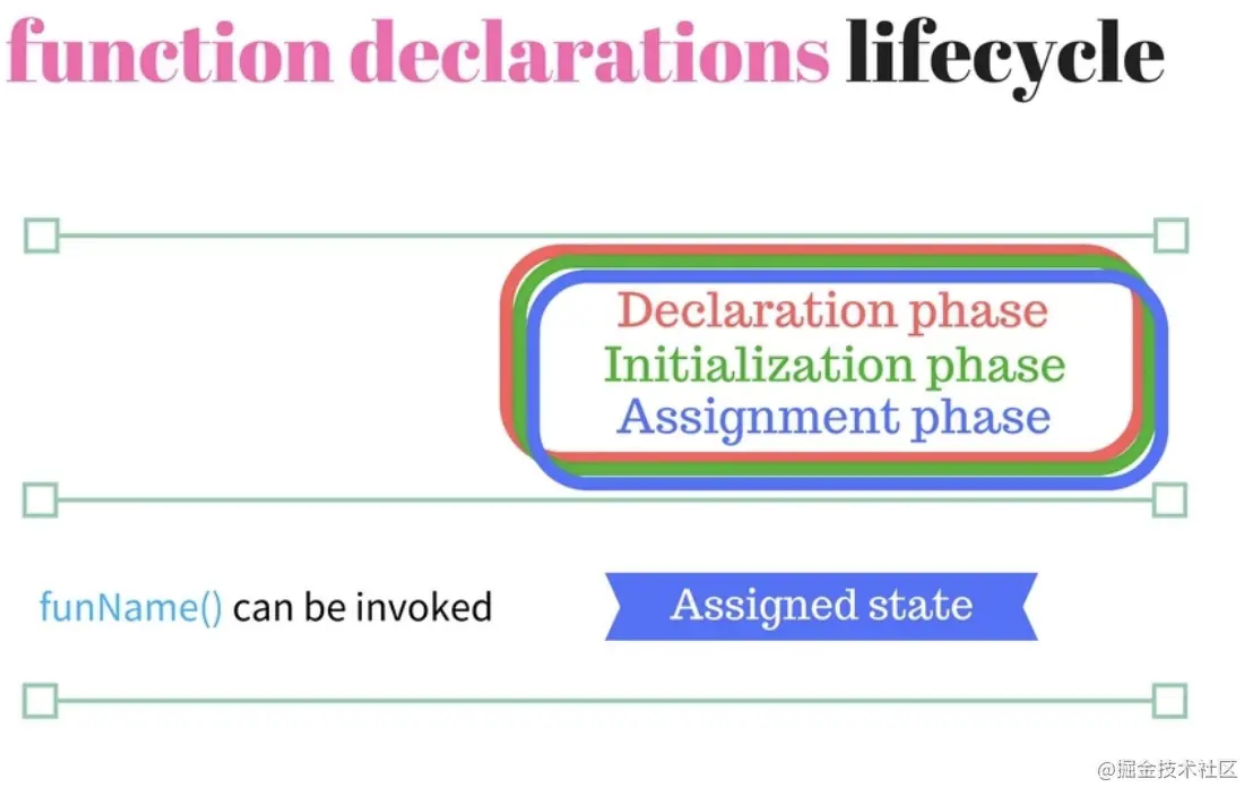
在执行任何语句之前，变量在作用域的开头通过声明阶段并立即初始化阶段（上图步骤1）。 var variable语句在函数作用域中的位置不影响声明和初始化阶段。

在声明和初始化之后，但是在赋值阶段之前，该变量具有undefined值并且可以被使用。

在赋值阶段 variable = 'value'，变量将接收其初始值（上图步骤2）。

1.2 函数声明生命周期

函数声明：`function funName() {...}`



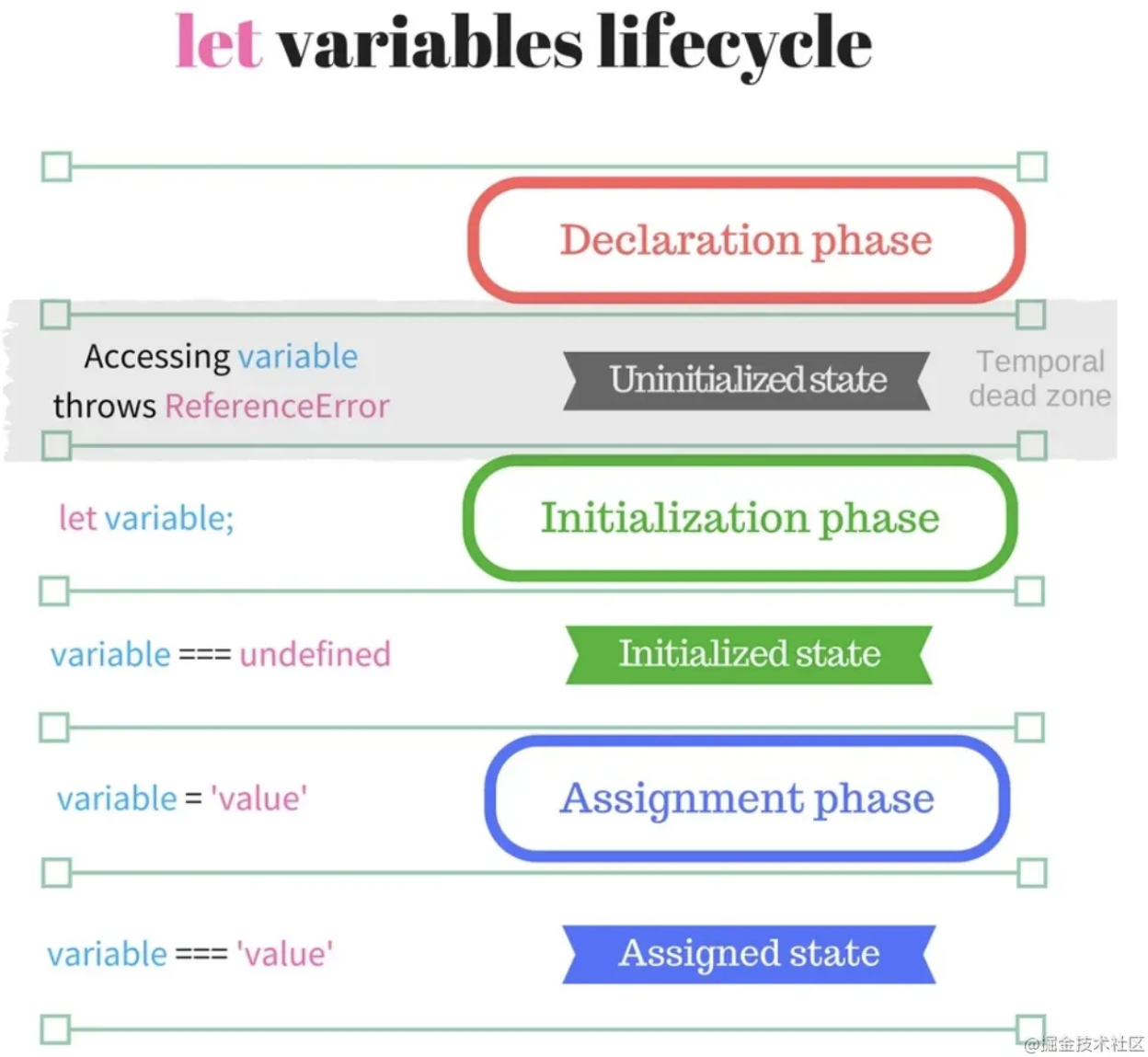
function 声明会在代码执行之前就「创建、初始化并赋值」。

```
function sumArray(array) {
  return array.reduce(sum);
  function sum(a, b) {
    return a + b;
  }
}
sumArray([5, 10, 8]); // => 23
复制代码
```

执行JavaScript时sumArray([5, 10, 8])，它将进入sumArray函数范围。在此范围内，紧接着执行任何语句之前，sum将通过所有三个阶段：声明，初始化和赋值。 这种方式甚至array.reduce(sum)可以sum在其声明语句之前使用function sum(a, b) {...}。

1.3 let变量生命周期

let变量的处理方式不同于var。主要区别是声明和初始化阶段是分开的。




```
let a = 1;
{
  // 初始化前无法访问“a”，在这里的a指的是下面的a，不是全局的a，此时a还没有被初始化，
  //所以在这里log会报错，因为在这里是暂时性死区
  // console.log(a);


  let a;
```

```
  //解释器进入包含let variable语句的块范围的情况。变量立即通过声明阶段，在范围内注册其名称，
  //然后解释器继续逐行解析块语句。

  console.log('########## - a', a);//初始化，对其进行访问的结果为undefined
  a='被重新赋值了'
  console.log('========= - a', a);

  console.log('########## - b', b);
  var b;
  b='b';
  console.log('========= - b', b);
}
console.log('全局的a',a);
console.log('全局的b', b);
复制代码
```

运行结果：

```
########## - a undefined
========= - a 被重新赋值了
########## - b undefined
========= - b b
全局的a 1
全局的b b
复制代码
```

我认为：在块级作用域中，从块级作用域中的第一行开始，到用`let variable`声明变量这一行之前，这一段区域是let的暂时性死区。

这个暂时性死区就是在上图的`Uninitialized state`阶段，如果在这一阶段的时候，访问这个变量就会报错。

在初始化阶段`Initialized state`之后就可以访问这个`let`变量了。

1.3.1 为什么`Hosting`在生命周期中无效

如上所述，提升是变量在顶部的耦合声明和初始化。 但是`let`声明的变量，他的生命周期使声明和初始化阶段脱钩。解耦消除了`Hosting`的术语`let`。 这两个阶段之间的间隙创建了临时死区，无法访问该变量。

总结

`let`的「创建」过程被提升了，但是初始化没有提升。因为声明和初始化阶段是分离的，所以提升对于`let`变量（包括`for const`和`class`）无效。在初始化之前，该变量位于时间死区中并且不可访问。

> 在这里再提一下：
>
> 如果`let x`的初始化过程失败了，那么`x`变量就将永远处声明（`Declaration`）状态。 你无法再次对`x`进行初始化（初始化只有一次机会，而那次机会你失败了）。 由于 x 无法被初始化，所以`x`永远处在暂时死区.

作者：烟暖雨收

链接：https://juejin.cn/post/6844904050614353928

来源：掘金