嗨，你知道 let 和 const 吗?

前言

如果有人问你：知道 let 和 const 吗 答案是肯定的，还能回答的头头是道：

- let 和 const 不就是 es6 新增的声明变量的关键字嘛
- let 是用来声明变量的，const 是用来声明常量的
- 与 var 不同，let、const 不存在变量提升
- 暂时性死区
- 不绑定顶级对象

再问：

- let 和 const 真的不存在变量提升吗？ 为什么不存在
- 暂时性死区又是如何形成的?
- let、const 声明的全局变量不在 全局对象中，那它存在于哪里呢?

前言 2

我是一个前端菜鸟，经常使用 let、const，我知道 let (let 和 const 的基本特性一致，后面就不带 const 了) 不存在变量提升，变量操作必须在声明之后，否则就会报错；还知道 let 声明的变量存在于块级作用域中，块外无法访问块内使用 let 声明的变量；我还知道 let 声明的变量不会被挂载到全局对象中(刚知道)；

很久以前，我知道了执行上下文中的变量对象这个东西，理解了变量对象，继而理解了变量提升的过程，后来心中一直有一个疑惑，使用 var 声明的变量会被创建在 变量对象中，也就是所谓的变量提升，let 又为什么不会变量提升呢；

1. var 的变量提升规则

> 复习变量提升的概念，其中涉及到 JS 的 **执行上下文、变量对象** 等知识。

```
var a = "global";
function bar(a) {
  console.log(temp); // undefined
  var temp = "local";
}

bar();
复制代码
```

我们都知道在 js 中，代码执行时会把当前作用域中的所有的使用 var 声明的变量以及 function 声明的函数提升到作用域的顶部(变量提升声明，不提升赋值；函数提升函数体，不提升调用)。所以我们在 var 声明变量之前可以访问 temp，结果是 **undefined**

让我们从 js 代码的运行角度去看这个过程，如下：

> **<1>** 执行流进入全局执行环境

> **<2>** 创建全局执行环境的变量对象, 并将变量 a 的声明、函数 bar 的声明添加到变量对象中
>
> **<3>** 进入全局代码执行阶段
>
> **<4>** 对变量 a 赋值为 'global'
>
> **<5>** 遇到 函数 bar 的调用
>
> **<6>** 进入函数 bar 内部，创建 bar 的执行环境，压入执行环境栈
>
> **<7>** 同样的，创建函数 bar 执行环境的活动对象，将变量 temp 在 活动对象内创建，key 为 temp，初始化值为 undefined
>
> **<8>** 进入代码执行阶段，遇到 打印输出 a，此时 a 存在于活动对象中，值为 undefined
>
> **<9>** 继续执行，为 temp 赋值为 'local'
>
> **<10>** bar 函数内代码执行完毕，执行环境栈谈出 bar 执行环境，执行流回到 global 执行环境中继续执行...

小结

> js 在代码预编译阶段，会创建一个变量对象，变量对象中有一个属性，属性名称为 temp，属性值为 undefined，所以在变量声明之前打印 temp，值为 undefined；这也就是 var 的变量提升规则

2. let 的提升规则

死缓区？ 暂存死区？ TDZ (Temporal dead zone)?

如果你能理解 var 的提升规则，那么理解 let 的提升将会变得很轻松。 在 MDN 关于 let 的文档中，有这么一句话：

> **The other difference between var and let is that the latter is initialized to value only when parser evaluates it (see below).**

ps: 我的英语极差，MDN 的翻译是这样的：

> var 和 let 的不同之处在于后者是在编译时才初始化（见下面）

我使用 有道翻译 是酱紫的：

> var 和 let 之间的另一个区别是，后者只有在解析器对其求值时才初始化为 value(参见下面的内容)。

我预感到 MDN 的中文翻译并不准确(因为 var 是在编译时初始化的，可见翻译有问题)，再结合 有道的硬核翻译，我将其理解为：

> **var 和 let 的不同之处在于后者是在运行时才初始化的**

在 MDN 上点击 (见下面) 看到了 暂存死区的概念：

> 与通过 var 声明的有初始化值 undefined 的变量不同，通过 let 声明的变量直到它们的定义被执行时才初始化。在变量初始化前访问该变量会导致 ReferenceError。该变量处在一个自块顶部到初始化处理的“暂存死区”中。

结合例子：

例 1.

```
function foo() {
  console.log(a); // Uncaught ReferenceError: Cannot access 'a' before initialization
  let a = 123;
}
foo();
复制代码
```

> 典型的 let 声明变量的例子：
>
> 与 var 不同的是，我们无法在 let 声明之前获取该变量；
>
> 否则，报错：Cannot access 'a' before initialization；
>
> 报错信息给到我们，硬核翻译为：无法在初始化之前访问'a'；
>
> 从 “无法在初始化之前访问'a'” 是否能够看出写端倪呢？

例 2.

```
let a = 3;
let b;
(function() {
  console.log(a); // Uncaught ReferenceError: Cannot access 'a' before initialization
  console.log(b);
  let a = (b = 3);
  console.log(a);
  console.log(b);
})();
console.log(a);
console.log(b);
复制代码
```

> 结果为：报错 - Uncaught ReferenceError: Cannot access 'a' before initialization；
>
> 与 例 1 报的同样的错误；
>
> 我们试想下：
>
> 在匿名函数内，第一个访问的 a，为什么不能获取到外部环境的 a = 3；而是在这里就早早的报错了呢？
>
> 估计很多人就会说：这不就是因为 let 的暂时性死区特性嘛；
>
> 在 阮大神的 《es6 入门》 中，对于暂时性死区这样写到：
>
> > 只要块级作用域内存在 let 命令，它所声明的变量就“绑定”（binding）这个区域，不再受外部的影响。
>
> 以及后面的介绍，在我看来只是声明式的告诉了我：在作用域内，无论代码先后顺序，只要在代码中出现了 let a，那么 a 变量就 **“绑定”** 了这个作用域，有没有疑问？ 为什么我声明变量的代码明明还在下面，上面的代码就被绑定了呢，js 不是顺序执行的吗？
>
> OK，走到了现在，结合 var 的提升规则，是不是可以将 let 理解为：
>
> - 使用 let 声明的变量实际上也存在提升，但是与 var 的提升规则不同：
> - var 声明的变量是在代码预编译阶段 被创建在了 执行环境的变量对象中，并且将其初始化，初始化值为 undefined；

> - 而 let 声明的变量，在执行环境预编译阶段，被提升到了一个叫做 **“暂存死区”** 的地方，并且没有对其进行**初始化**；
> - 所以，无论使用 let 声明的变量，声明在代码的任何位置，这个变量一开始就被 **“绑定”** 在了当前执行环境中；
> - 在进入执行阶段时，只有在 let 声明被执行后，才对这个变量进行了初始化；
> - 所以，在执行到 let 声明之前，是无法访问这个被 **“绑定”** 的变量的。

进一步验证

```
let a = 3;
let b;
(function() {
  let a;
  console.log(a); // undefined
  console.log(b); // undefined
  b = 3;
  a = b;
  console.log(a); // 3
  console.log(b); // 3
})();
console.log(a); // 3
console.log(b); // 3
复制代码
```

> 代码顺利执行
>
> 我们将变量 a 的声明提升到了函数的顶部，在进入函数执行阶段，第一句执行的就是 let a; 声明了 a 变量，并将其值初始化为 undefined
>
> 所以在下面打印 a 为 undefined
>
> 后续代码就不说了...

小结

> 1. 其实 let 和 const 也是存在变量提升的，只不过和 var 的提升规则不同，let 是将变量提升到了一个叫做 **“暂存死区”** 的地方，在提升时并没有对其进行初始化，如果去访问 **“暂存死区”** 中的变量，就会报错；
> 2. 由于 let 的变量提升是在 “预编译” 阶段完成的，所以在进入执行环境后，无论声明代码在何处，这个被声明的变量就被绑定了，在声明之前访问这个变量，就会报错，也就会形成 代码的 **“暂时性死区”**
> 3. 在执行环境的执行阶段，在当执行到 let 声明时，那个变量才会 从 **“暂存死区”** 中移除，并对齐初始化为 undefined，所以 使用 let 声明的变量只能在其声明后访问。
> 4. 其实在我的理解中，**“暂存死区”** 和 **“暂时性死区”** 并不是一个概念，**“暂存死区”** 亦可称为 **“死缓区”** (死亡缓存区域？ 随便怎么叫了)，**“暂存死区”** 是一个保存 let 声明的变量的地方，正是因为有这个区域的存在，才使得 let 声明的变量能够保证不能再声明前访问变量，继而也就形成了 **“暂时性死区”**

3. 全局环境下的 let

在 《es6 入门》 中如是写到：

> 顶层对象，在浏览器环境指的是 window 对象，在 Node 指的是 global 对象。ES5 之中，顶层对象的属性与全局变量是等价的。

> let 命令、const 命令、class 命令声明的全局变量，不属于顶层对象的属性。

在 es5 时期，全局声明的变量是被挂载在顶级对象下的，在 es6 时期，使用 let 声明的全局变量，并不存在于顶级对象下

> 那么，使用 let 声明的变量存在于哪里呢?

```
var a = 1;
let b = 2;

console.log(window.a); // 1
console.log(window.b); // undefined
复制代码
```

> 下列涉及到作用域、作用域链生成规则等知识, 如果不理解作用链的生成规则可能会对下面产生疑惑。

友情链接：了解作用域 | 深入作用域链

```
let str = "global Str";
function bar() {
  let str = "local Str";
}

console.dir(bar);
复制代码
```

输出结果：

> 上图中的 圈红部分是函数的作用域链(在 谷歌浏览器可以看到如上的打印输出，在 火狐 ie 下均没有)
>
> 可以看到，在作用域链中存在两项：
>
> - **[[Scopes]][0]: Script** 和顶级对象平行的一个作用域，可以看到里面貌似有熟悉的东西
> - **[[Scopes]][1]: Global** 也就是顶级作用域的变量对象，在浏览器中就是 window

小结

> - let 定义的全局变量并不存在于 顶级对象中，而是存在于和顶级对象平行的一个全局作用域中

> - 亦或者可以说 let 和 const 定义的变量是存在于作用链的顶端的，根据作用域链的访问规则，可以访问到全局变量
> - 至于这个 作用域链顶端的 Script 到底是什么，我也说不清楚，我目前没有仔细去找解释这个 **[[Scopes]][0]: Script** 的文档，大致的找了下，并没有找到，还望有了解的大佬解释下这个 作用域链 顶端的 Script 到底是个什么东西。

干货总结

- let 和 const 也存在变量提升，预编译阶段提升，所以能绑定整个作用域
- let 和 const 将变量提升到了一个称为 “死缓区” 的地方，尝试访问 “死缓区” 内容将会报错，所以形成 “暂时性死区”
- let 和 const 在提升变量时不会对其初始化操作
- let 和 cosnt 声明的全局变量在 作用域链的顶端，一个叫 Script 的作用域里，根据作用域链规则，可以访问到其定义的全局变量