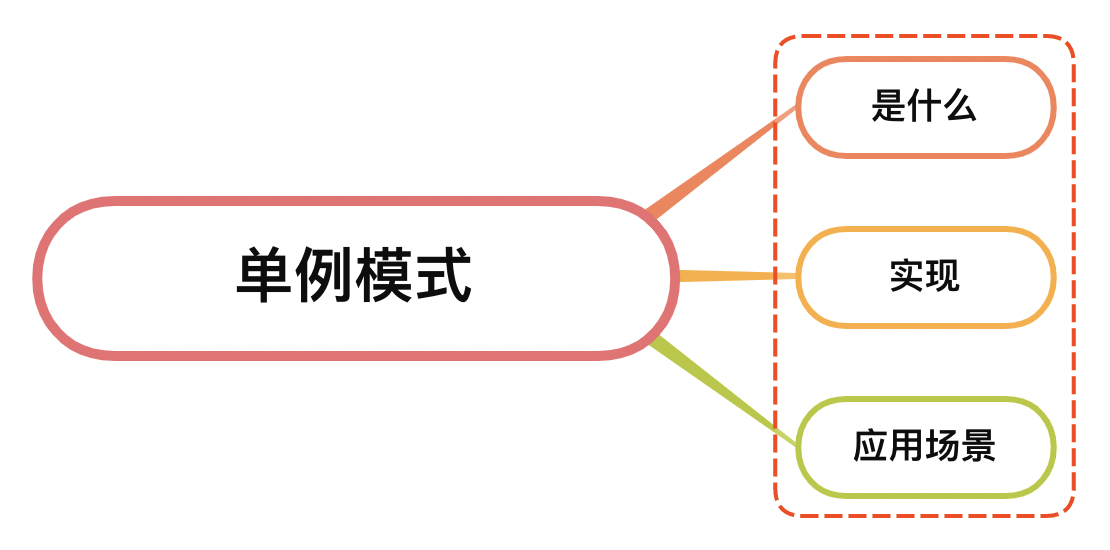
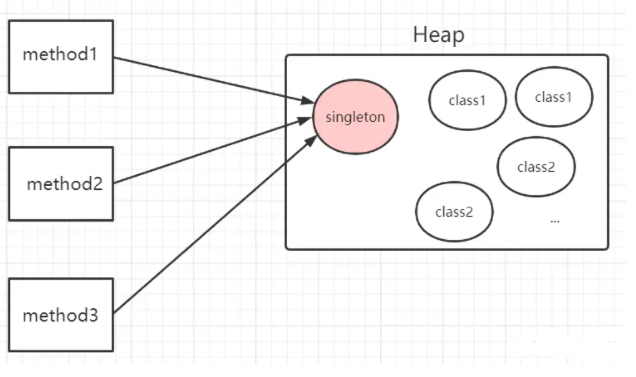
# 面试官：说说你对单例模式的理解？如何实现？



## 一、是什么

单例模式（Singleton Pattern）：创建型模式，提供了一种创建对象的最佳方式，这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建

在应用程序运行期间，单例模式只会在全局作用域下创建一次实例对象，让所有需要调用的地方都共享这一单例对象，如下图所示：



从定义上来看，全局变量好像就是单例模式，但是一般情况我们不认为全局变量是一个单例模式，原因是：

* 全局命名污染
* 不易维护，容易被重写覆盖

## 二、实现

在javascript中，实现一个单例模式可以用一个变量来标志当前的类已经创建过对象，如果下次获取当前类的实例时，直接返回之前创建的对象即可，如下：

```JS // 定义一个类 function Singleton(name) { this.name = name; this.instance = null; } // 原型扩展类的一个方法getName() Singleton.prototype.getName = function() { console.log(this.name) }; // 获取类的实例 Singleton.getInstance = function(name) { if(!this.instance) { this.instance = new Singleton(name); } return this.instance };

// 获取对象1 const a = Singleton.getInstance('a'); // 获取对象2 const b = Singleton.getInstance('b'); // 进行比较 console.log(a === b); ```

使用闭包也能够实现，如下：

```js function Singleton(name) { this.name = name; } // 原型扩展类的一个方法getName() Singleton.prototype.getName = function() { console.log(this.name) }; // 获取类的实例 Singleton.getInstance = (function() { var instance = null; return function(name) { if(!this.instance) { this.instance = new Singleton(name); } return this.instance }  
})();

// 获取对象1 const a = Singleton.getInstance('a'); // 获取对象2 const b = Singleton.getInstance('b'); // 进行比较 console.log(a === b); ```

也可以将上述的方法稍作修改，变成构造函数的形式，如下：

```js // 单例构造函数 function CreateSingleton (name) { this.name = name; this.getName(); };

// 获取实例的名字 CreateSingleton.prototype.getName = function() { console.log(this.name) }; // 单例对象 const Singleton = (function(){ var instance; return function (name) { if(!instance) { instance = new CreateSingleton(name); } return instance; } })();

// 创建实例对象1 const a = new Singleton('a'); // 创建实例对象2 const b = new Singleton('b');

console.log(a===b); // true ```

## 三、使用场景

在前端中，很多情况都是用到单例模式，例如页面存在一个模态框的时候，只有用户点击的时候才会创建，而不是加载完成之后再创建弹窗和隐藏，并且保证弹窗全局只有一个

可以先创建一个通常的获取对象的方法，如下：

js const getSingle = function( fn ){ let result; return function(){ return result || ( result = fn .apply(this, arguments ) ); } };

创建弹窗的代码如下：

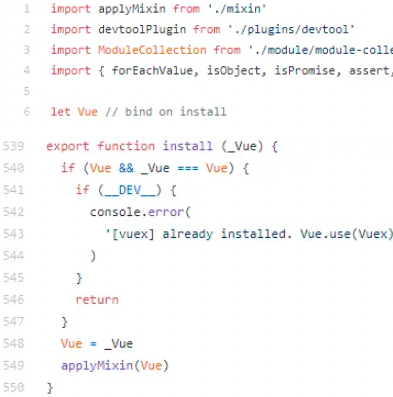
```js const createLoginLayer = function(){ var div = document.createElement( 'div' ); div.innerHTML = '我是浮窗'; div.style.display = 'none'; document.body.appendChild( div ); return div; };

const createSingleLoginLayer = getSingle( createLoginLayer );

document.getElementById( 'loginBtn' ).onclick = function(){ var loginLayer = createSingleLoginLayer(); loginLayer.style.display = 'block'; }; ```

上述这种实现称为惰性单例，意图解决需要时才创建类实例对象

并且Vuex、redux全局态管理库也应用单例模式的思想，如下图：



现在很多第三方库都是单例模式，多次引用只会使用同一个对象，如jquery、lodash、moment...

## 参考文献

* https://zh.wikipedia.org/zh-hans/%E5%8D%95%E4%BE%8B%E6%A8%A1%E5%BC%8F
* https://www.runoob.com/design-pattern/singleton-pattern.html
* https://juejin.cn/post/6844903874210299912#heading-5