**JavaScript设计模式--单例模式**

一、定义

保证一个类仅有一个实例，并提供一个访问它的全局访问点。   
当单击登陆按钮，页面中出现一个登陆浮窗，这个登陆浮窗是唯一的，无论单击多少次登陆按钮，这个浮窗都只会被创建一次，那么这个登陆浮窗就适合用单例模式来创建。

二、实现原理

要实现单例并不复杂，使用一个变量来标志当前是否已经为某个类创建过对象，如果是，则在下一次获取该类的实例时，直接返回之前创建的对象。

三、假单例

全局变量不是单例模式，但在JavaScript开发中，我们经常会把全局变量当成单例来使用。

**var *a*** = {};

降低全局变量带来的命名污染   
（1）使用命名空间

<**script**>  
 **var *namespace1*** = {  
 a: **function**(){},  
 **b**: 2  
 }  
</**script**>

（2）使用闭包封装私有变量

<**script**>  
 **var *user*** = (**function**() {  
 **var** \_name = **'lee'**,  
 \_age = **'25'**;  
 **return** {  
 getUserInfo: **function**() {  
 **return** \_name + **":"** + \_age;  
 }  
 };  
 })();  
</**script**>

四、惰性单例：在需要的时候才能创建对象实例。

<**script**>  
 **var** *getSingle* = **function**(fn) {  
 **var** result;  
 **return function**() {  
 **return** result || (result = fn.apply(**this**, arguments));  
 };  
 };  
  
 *// 测试* **function** *testSingle*(){}  
 *getSingle*(*testSingle*)() === *getSingle*(*testSingle*)(); *// true*</**script**>

五、补充：

（1）懒加载

<**script**>  
 **var** *lazyload* = **function**() {  
 **console**.log(1);  
 *lazyload* = **function**() {  
 **console**.log(2);  
 }  
 **return** *lazyload*();  
 }  
  
 *lazyload*();  
</**script**>

（2）预加载

<**script**>  
 **var *preload*** = (**function**() {  
 **console**.log(1);  
 ***preload*** = **function**() {  
 **console**.log(2);  
 };  
 **return *preload***;  
 })();  
  
 ***preload***();  
</**script**>

最基本的单例模式——对象字面量

<**script**>  
 **var *Singleton*** = {  
 **attribute1**: **true**,  
 **attribute2**: 10,  
  
 method1: **function**() {  
  
 },  
 method2: **function**(arg) {  
  
 }  
 };  
  
 ***Singleton***.**attribute1** = **false**;  
 **var *total*** = ***Singleton***.**attribute2** + 5;  
 **var *result*** = ***Singleton***.method1();  
</**script**>

使用闭包

如果以后要扩展该对象，你可以添加自己的私有成员和方法，使用闭包将其封装这些变量和函数声明，只暴露你想暴露的public成员和方法。

<**script**>  
 **var *Singleton*** = (**function**() {  
 *// Private members.* **var** privateAttribute1 = **false**;  
 **var** privateAttribute2 = [1, 2, 3];  
 *// Private function* **function** *privateMethod1*() {  
 ...  
 }  
 **function** *privateMethod2*(args) {  
 ...  
 }  
  
  
 **return** { *// Public members.* **publicAttribute1**: **true**,  
 **publicAttribute2**: 10,  
  
 publicMethod1: **function**() {  
 ...  
 },  
 publicMethod2: **function**(args) {  
 ...  
 }  
 };  
 })();  
 **var *singleton*** = ***Singleton***();  
 ***singleton***.publicMethod1();  
</**script**>

# 惰性实例化

前面所讲的单例模式都有一个共同特点，单例对象都是在脚本加载时被创建出来，对于资源密集型的或配置开销甚大的单体，也许更合理的做法是将其实例化推迟到需要使用时才实例化。它最常用于那些必须加载大量数据的单体。惰性加载单体的特别之处在于，对他们的访问必须借助于一个静态方法。应该这样调用其方法：Singleton.getInstance().methodName()。getInstance()方法会检查单体是否已经被实例化，如果还没有，那么创建并返回其实例，如果单体已经实例过，那么将返回现有实例。

<**script**>  
 **var *Singleton*** = (**function**() {  
  
 **var** instanced;  
  
 **function** *constructor* () {  
 **var** privateAttribute1 = **false**;  
 **var** privateAttribute2 = [1, 2, 3];  
  
 **function** *privateMethod1*() {  
 ...  
 }  
 **function** *privateMethod2*(args) {  
 ...  
 }  
  
 **return** {  
 **publicAttribute1**: **true**,  
 **publicAttribute2**: 10,  
  
 publicMethod1: **function**() {  
 ...  
 },  
 publicMethod2: **function**(args) {  
 ...  
 }  
 };  
 }  
  
 **return** {  
 getInstance: **function** () {  
 **if** (!instanced) {  
 instanced = *constructor*();  
 }  
 **return** instanced;  
 }  
 };  
 })();  
 ***Singleton***.getInstance().publicMethod1();  
</**script**>