**正课:**

**1. OOP**

**2. ES5**

**1. OOP：**

**自定义继承:**

**3. 两种类型间的继承:**

**问题: 在子类型构造函数中直接调用父类型构造函数，无法将父类型的属性加入到新对象中**

**原因: 子类型构造函数中直接调用的父类型构造函数，导致父类型构造函数中的this默认指向window，导致父类型的属性都泄漏到全局。**

**解决: 当一个函数中的this不是想要的时，可用call()，将函数中不想要的this换成想要的对象。**

**如何: 函数名.call(替换this的对象, 参数值... ...)**

**1. 可立刻调用函数**

**2. 将函数中的this，临时替换为想要的"对象"**

**3. 将参数值传给函数**

**多态:**

**什么是: 同一个函数，不同情况下表现出不同的状态**

**包括: 2种:**

**1. 重载:**

**2. 重写(override): 如果子对象觉得从父对象继承来的成员不好用，可在子对象本地定义同名成员，来覆盖父对象的成员。**

**鄙视: 判断一个对象是不是数组，共有几种方法:**

**typeof仅能区分原始类型和function，无法进一步区分每种对象的类型**

**1. 判断原型对象:**

**obj.\_\_proto\_\_==Array.prototype**

**Array.prototype.isPrototypeOf(obj)**

**2. 判断构造函数:**

**obj.constructor==Array**

**obj instanceof Array**

**obj(是)Array的一个实例吗？**

**问题: 以上两种方式，检查不严格，可能被篡改**

**3. 判断隐藏的class属性:**

**Object.prototype.toString.call(obj1)**

**this->obj1.class**

**4. Array.isArray(obj)**

**直接返回bool值**

**原理同方法3，也是严格的检查**

**静态方法:**

**什么是静态方法: 不需要创建子对象实例，用构造函数就可直接调用的方法**

**vs 实例方法: 必须创建该类型的子对象，用子对象才能调用的方法**

**为什么用静态方法: 有时在调用函数时，无法确定调用函数的实例是什么。**

**何时: 只要希望不确定实例对象，也能调用方法时**

**如何:**

**静态方法都要定义在构造函数对象上，而不是添加在原型对象中**

**鄙视: 何时使用静态方法，何时使用实例方法:**

**1. 如果规定必须某个类型的子对象才能调用的方法，就用实例方法**

**实例方法保存在该类型的原型对象中**

**2. 如果不确定调用该方法的数据类型时，就用静态方法**

**静态方法保存在构造函数对象上**

**2. ES5:**

**1. 严格模式:**

**什么是: 比普通js运行机制要求更严格的模式**

**为什么: 解决js中部分广受诟病的缺陷**

**比如:**

**1. 未声明的变量也可赋值，并自动创建在全局——全局污染**

**2. 静默失败: 执行不成功，也不报错**

**3. 普通函数调用中的this默认指window**

**4. 允许递归——效率极低**

**递归重复计算量巨大！**

**何时: 今后所有js程序都应该在严格模式下运行**

**包括:**

**1. 禁止给未声明的变量赋值**

**2. 将静默失败升级为错误**

**3. 普通函数调用中的this不再指向window，而是undefined**

**4. 强烈不建议使用递归**

**多数递归都可用循环代替**

**如何: 2种:**

**1. 可对整个js文件或<script>标签启用严格模式**

**在顶部添加: "use strict";**

**2. 仅对一个function启用严格模式**

**在function内的顶部加:"use strict"**

**2. 保护对象:**

**什么是: 让使用者按规则有限度的使用对象**

**为什么: js中的对象，默认没有任何限制**

**何时: 如果想控制使用者对对象的使用时**

**如何: 2个层面**

**1. 保护对象的属性:**

**ES5将对象的属性分为:**

**命名属性: 凡是用.可直接访问到的属性**

**如何保护命名属性: 又分为2类:**

**数据属性: 实际存储属性值的属性**

**访问器属性: 不实际存储属性值，仅提供对其他数据属性的保护。**

**内部属性: 不能用.直接访问的属性**

**比如: class**

**如何保护数据属性:**

**ES5将每个简单的属性变成了一个缩微的对象，每个属性对象，又包含四个特性:**

**属性:{**

**value: 属性值,**

**writable: true/false, 控制是否可修改属性值**

**enumerable: true/false, 控制是否可被for in遍历**

**但用.依然可以访问到**

**configurable: true/false, 控制是否可删除该属性**

**控制着是否可修改另外两个开关**

**一旦改为false，不可逆！**

**今后，修改前两个特性，都要带上configurable**

**}**

**查看一个属性的四个特性:**

**var attrs=**

**Object.getOwnPropertyDescriptor(obj,"属性名")**

**修改属性的四个特性:**

**Object.defineProperty(对象,"属性名",{**

**特性:值,**

**... : ...**

**})**

**问题: 一次只能修改一个属性的四个特性:**

**解决:**

**Object.defineProperties(对象,{**

**属性名:{**

**特性:值,**

**... : ...**

**},**

**属性名:{**

**... : ...**

**}**

**})**

**问题: 无法使用自定义规则保护属性**

**解决:**

**用访问器属性保护数据属性:**

**何时: 只要用自定义规则保护属性时**

**如何:**

**1. 先定义一个隐藏的要保护的数据属性**

**2. 定义访问器属性，保护数据属性:**

**强调: 访问器属性只能用defineProperty/Properties添加，不能用直接量添加。**

**Object.defineProperties(对象,{**

**其它属性...,**

**访问器属性:{**

**}**

**})**

**2. 保护对象的结构:**

**什么是: 禁止擅自删除或添加对象的属性**

**三个层次:**

**1. 禁止添加新属性:**

**Object.preventExtensions(obj)**

**原理: 每个对象都有一个内部属性: extensible:true**

**preventExtensions(obj)设置obj的extensible为false**

**2. 密封:**

**在禁止添加新属性的基础上，进一步禁止删除现有属性**

**Object.seal(obj)**

**原理: 设置obj的extensible为false**

**且自动设置obj的每个属性的configurable为false**

**3. 冻结:**

**在密封基础上禁止修改属性值**

**Object.freeze(obj)**

**原理: 设置obj的extensible为false**

**且自动设置obj的每个属性的configurable为false**

**且自动设置obj的每个属性的writable为false**