類別 organism(生物) 具有以下成員

var name //名稱

var age //年齡

var growing //生長

var death //死亡狀態

類別 pro_animal(動物) 繼承 organism(生物) 並具有以下成員

var deg_hunger//飢餓度

ver eating//進食

var phys_strength//體力

var speed//運動速度

var run//跑

function meet//兩動物相遇時觸發 meet 事件

function meet_carnivore//遇到肉食動物事件

function meet_herbivore//遇到草食動物事件

function meet_grass//遇到草事件

function meet_tree//遇到樹事件

類別 plant (植物) 繼承 organism(生物) 並具有以下成員

var species //利用此參數在相遇事件判斷遇到為何種物種

var sta//利用此參數判斷產生何種植物

類別 carnivore (肉食動物) 繼承 pro_animal(動物) 並具有以下成員

var species = 0//利用此參數在相遇事件判斷遇到為何種物種

var sta//利用此參數判斷產生何種肉食動物

function meet_carnivore//複寫父類別遇到肉食動物事件

function meet_herbivore//複寫父類別遇到草食動物事件

類別 herbivore (草食動物) 繼承 pro_animal(動物) 並具有以下成員

var species = 1//利用此參數在相遇事件判斷遇到為何種物種

var sta//利用此參數判斷產生何種草食動物

function meet_carnivore//覆寫父類別遇到肉食動物事件

function meet_herbivore//覆寫父類別遇到草食動物事件

function meet_grass//覆寫父類別遇到草事件

類別 lion(獅子) 繼承 carnivore (肉食動物) 並設定初始條件

生命週期(growing)30~40 天

體力(phys_strength)15

速度(speed)約 60~70

類別 tiger (老虎) 繼承 carnivore (肉食動物) 並設定初始條件
生命週期(growing)25~35 天
體力(phys_strength)12
速度(speed)約 55~65

類別 leopard (獵豹) 繼承 carnivore (肉食動物) 並設定初始條件
生命週期(growing)20~30 天
體力(phys_strength)10
速度(speed)約 80~90

類別 zebra (斑馬) 繼承 herbivore (草食動物) 並設定初始條件
生命週期(growing)35~40 天
體力(phys_strength)15
速度(speed)約 50~60

類別 antelope (羚羊) 繼承 herbivore (草食動物) 並設定初始條件
生命週期(growing)25~35 天
體力(phys_strength)10
速度(speed)約 55~65

類別 ostrich (鴕鳥) 繼承 herbivore (草食動物) 並設定初始條件
生命週期(growing)15~20 天
體力(phys_strength)8
速度(speed)約 85~95

類別 grassA (低級牧草) 繼承 plant (植物) 並設定初始條件
生命週期(growing)8~10 天
吃下後飢餓-1

類別 grassB (中級牧草) 繼承 plant (植物) 並設定初始條件
生命週期(growing)10~12 天
吃下後飢餓-2

類別 grassC (高級牧草) 繼承 plant (植物) 並設定初始條件
生命週期(growing)12~15 天
吃下後飢餓-3

類別 treeA (蘋果樹) 繼承 plant (植物) 並設定初始條件

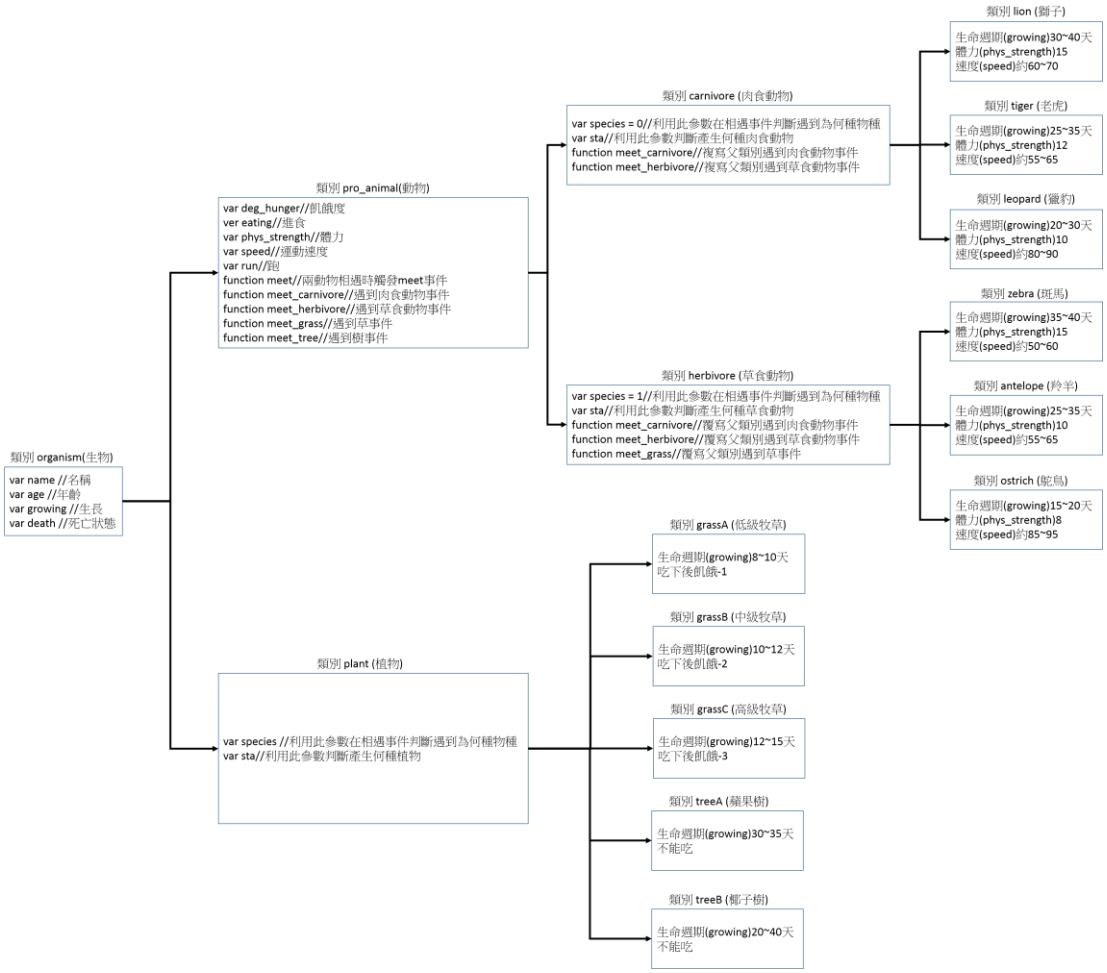
生命週期(growing)30~35 天

不能吃

類別 treeB (椰子樹) 繼承 plant (植物) 並設定初始條件

生命週期(growing)20~40 天

不能吃



Java Script 繼承自學筆記

1. 使用父類別當子類別的原型對象

```
1   var  organism = function(x){
2       this.x = [x];
3       this.aaa = function(){...}
6   };
7   var  pro_animal = function(y){
8       this.y = y;
9   };
10  var  carnivore = function(z){
11      this.z = z;
12  };
13  var  lion = function(){
14
15  };
16  pro_animal.prototype = new organism(1);
17  carnivore.prototype = new pro_animal(2);
18  lion.prototype = new carnivore(3);
19
20  var ani1 = new lion();
21  var ani2 = new lion();
22  console.log(ani1.x);//[1]
23  console.log(ani2.x);//[1]
24  ani1.x.push(5);
25  console.log(ani1.x);//[1,5]
26  console.log(ani2.x);//[1,5]
27  console.log(ani1.aaa == ani2.aaa);//true
28  console.log(ani1.aaa === ani2.aaa);//true
```

優點；易讀

缺點：

1. 原型對象的引用屬性是所有實例共享的，修改 ani1.x 後(line 24)，導致 ani2.x 跟著變動
2. 創建子類別實例時，無法向父類別傳遞參數

2. 指定父類為子類的原型

```
var organism = {
    org:"生物"
}
var  pro_animal = {
    ani:'動物'
}
Object.setPrototypeOf(pro_animal, organism);//指定organism為pro_animal的原型

var  carnivore = {
    carn:'肉食動物'
}
Object.setPrototypeOf(carnivore, pro_animal);//指定pro_animal為carnivore的原型

var  lion = {
    lion:true
}
Object.setPrototypeOf(lion, carnivore);//指定carnivore為lion的原型
var  tiger = {
    tiger:true
}
//Object.setPrototypeOf(tiger, carnivore);

console.log(lion.lion +" "+ lion.carn +" "+ lion.ani +" "+ lion.org);
//true 肉食動物 動物 生物
console.log(tiger.tiger +" "+ tiger.carn +" "+ tiger.ani +" "+ tiger.org);
//true undefined undefined undefined

```

優點；易讀

缺點：

1. 原型對象的引用屬性是所有實例共享的
2. 創建子類別實例時，無法向父類別傳遞參數

3. 使用預設構造函數來建立子類別實例

```
var  organism = function(x){
    this.x = [x];
};
var  pro_animal = function(y){
    organism.call(this, 1)//pro_animal extend organism
    this.y = y;
};
var  carnivore = function(z){
    pro_animal.call(this, 2)//carnivore extend pro_animal
    this.z = z;
};
var  lion = function(){
    carnivore.call(this, 3)//lion extend carnivore

};

var ani1 = new lion();
var ani2 = new lion();
console.log(ani1.x);//[1]
console.log(ani2.x);//[1]
ani1.x.push(5);
console.log(ani1.x);//[1,5]
console.log(ani2.x);//[1]
```

優點；

1. 解決方法 1 中，子類別實例共享父類別引用屬性問題
2. 解決方法 1 中，創建子類別實例時，可向父類別傳遞參數

缺點：

1. 每個子類別實例都會調用兩次父類(在此作業中，需大量 new 動物，因此該方法影響內存跟效能)

4. 以方法 1、2、3 為基礎實現繼承

```
var  organism = function(x){
    this.x = [x];
};
var  pro_animal = function(y){
    organism.call(this, 1)
    this.y = y;
};
proto = Object.create(organism.prototype);//將carnivore(父類)的原型取出
proto.constructor = pro_animal;//改變proto.constructor為子類
pro_animal.prototype = proto;

var  carnivore = function(z){
    pro_animal.call(this, 2)
    this.z = z;
};
proto = Object.create(pro_animal.prototype);
proto.constructor = carnivore;
carnivore.prototype = proto;
var  lion = function(){
    carnivore.call(this, 3);
};
proto = Object.create(carnivore.prototype);
proto.constructor = lion;
lion.prototype = proto;

var ani1 = new lion();
var ani2 = new lion();

console.log(ani1.x);//[1]
console.log(ani2.x);//[1]
ani1.x.push(5);
console.log(ani1);//[1, 5]
console.log(ani2);//[1]
```

優點；

1. 解決方法 1，子類別實例共享父類別引用屬性問題
2. 解決方法 1，創建子類別實例時，可向父類別傳遞參數
3. 將父類的原型取出定義給子類別的原型，減少多餘的調用父類

缺點：

1. 不易讀
2. 使用麻煩，每一個子原型都要定義一次父原型