

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  
var 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 (駝鳥) 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

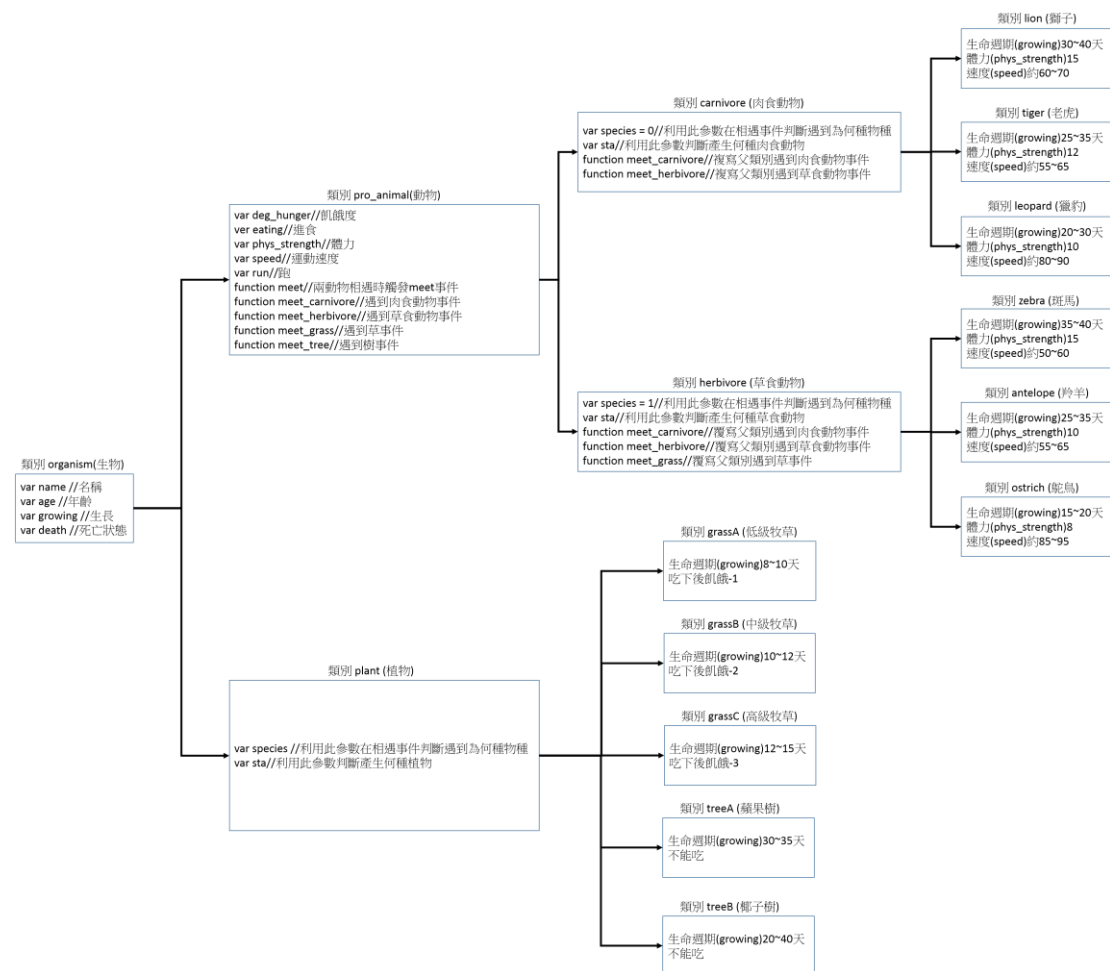
類別 grassC (高級牧草) 繼承 plant (植物) 並設定初始條件

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



## Xsproject\_V2.0 修正地方

1. 於類別 `pro_animal`(動物)新增
  - i. `Function meet()` //動物主動相遇其他物件事件
  - ii. `Function meet_carnivore ()`//動物遇到肉食動物事件
  - iii. `Function meet_herbivore ()`//動物遇到草食動物事件
  - iv. `Function meet_grass ()`//動物遇到草事件
  - v. `Function meet_tree ()`//動物遇到樹事件
2. 於類別 `carnivore` (肉食動物)新增
  - i. `Function meet_carnivore ()`//覆寫父類別(`pro_animal`)遇到肉食動物事件
  - ii. `Function meet_herbivore ()`//覆寫父類別(`pro_animal`)遇到草食動物事件
3. 於類別 `herbivore` (草食動物)新增
  - i. `Function meet_carnivore ()`//覆寫父類別(`pro_animal`)遇到肉食動物事件
  - ii. `Function meet_herbivore ()`//覆寫父類別(`pro_animal`)遇到草食動物事件
  - iii. `Function meet_grass ()`//覆寫父類別(`pro_animal`)遇到草事件
4. 建立 `lion`(獅子)類別 並繼承 `carnivore` (肉食動物)類別，並隨機產生其合理初始化條件(其餘肉食草食動物亦同，以下略)
5. 建立 `grassA` (低級牧草)類別 並繼承 `plant` (植物)類別，並隨機產生其合理初始化條件(其餘草木亦同，以下略)
6. 使用構造函數以及父子類原型實現繼承(原僅使用構造函數)，實驗結果與比較參見下節。
7. 每一日，動物體力、飢餓度、年齡上升，植物年齡上升，當物件生命週期到時，該物件自然死亡
8. 於儲存各種動植物的 `object` 上移除已死亡的 `object`(目的:降低內存)
9. 將各個動植物獨立寫成為類別，並使用 `switch case` 決定要產生何種實例(往後預新增其餘物種僅需新增 `case` 以及該物種類別)

## 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

```
1  var organism = {  
2    org:"生物"  
3  }  
4  var pro_animal = {  
5    ani:'動物'  
6  }  
7  Object.setPrototypeOf(pro_animal, organism); //指定organism為pro_animal的原型  
8  
9  var carnivore = {  
10    carn:'肉食動物'  
11  }  
12  Object.setPrototypeOf(carnivore, pro_animal); //指定pro_animal為carnivore的原型  
13  
14  var lion = {  
15    lion:true  
16  }  
17  Object.setPrototypeOf(lion, carnivore); //指定carnivore為lion的原型  
18  var tiger = {  
19    tiger:true  
20  }  
21  //Object.setPrototypeOf(tiger, carnivore);  
22  
23  console.log(lion.lion + " " + lion.carn + " " + lion.ani + " " + lion.org);  
24  //true 肉食動物 動物 生物  
25  console.log(tiger.tiger + " " + tiger.carn + " " + tiger.ani + " " + tiger.org);  
26  //true undefined undefined undefined  
27
```

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(原版)

```
1  var organism = function(x){
2      this.x = [x];
3  };
4  var pro_animal = function(y){
5      organism.call(this, 1)//pro_animal extend organism
6      this.y = y;
7  };
8  var carnivore = function(z){
9      pro_animal.call(this, 2)//carnivore extend pro_animal
10     this.z = z;
11 };
12 var lion = function(){
13     carnivore.call(this, 3)//lion extend carnivore
14 };
15
16
17 var ani1 = new lion();
18 var ani2 = new lion();
19 console.log(ani1.x);//[1]
20 console.log(ani2.x);//[1]
21 ani1.x.push(5);
22 console.log(ani1.x);//[1,5]
23 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，需大量創建動物，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(V2.0 修正版)

```
1  var organism = function(x){
2      this.x = [x];
3  };
4  var pro_animal = function(y){
5      organism.call(this, 1);
6      this.y = y;
7  };
8  proto = Object.create(organism.prototype); //將carnivore(父類)的原型取出
9  proto.constructor = pro_animal; //改變proto.constructor為子類
10 pro_animal.prototype = proto;
11
12 var carnivore = function(z){
13     pro_animal.call(this, 2);
14     this.z = z;
15 };
16 proto = Object.create(pro_animal.prototype);
17 proto.constructor = carnivore;
18 carnivore.prototype = proto;
19 var lion = function(){
20     carnivore.call(this, 3);
21 };
22 proto = Object.create(carnivore.prototype);
23 proto.constructor = lion;
24 lion.prototype = proto;
25
26 var anil = new lion();
27 var ani2 = new lion();
28
29 console.log(anil.x); // [1]
30 console.log(ani2.x); // [1]
31 anil.x.push(5);
32 console.log(anil); // [1, 5]
33 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. 使用麻煩，每一個子原型都要定義一次父原型



參考資料：

Object 對象 <http://javascript.ruanyifeng.com/stdlib/object.html>

prototype 對象 <http://javascript.ruanyifeng.com/oop/prototype.html>

Object 對象的相關方法 <http://javascript.ruanyifeng.com/oop/object.html>

重新認識 JavaScript: Day 22 深入理解 JavaScript 物件屬性

<https://ithelp.ithome.com.tw/articles/10193747>

重新認識 JavaScript: Day 24 物件與原型鏈

<https://ithelp.ithome.com.tw/articles/10194154>

prototype (原型) 屬性 <https://www.jianshu.com/p/4f87d28923ce>