細說 JavaScirpt 物件

向䀹田

基本觀念

- JavaScript 的物件是一堆 key-value 的集合
- 每一個 key-value 配對稱為屬性(property)
- 一個 property 可以是 function、陣列、物件或是基本資料型態(number, string)

物件範例

```
var person = {
    firstName: "Leo",
    lastName: "Shiang",
    fullName: function() {
        return this.firstName + " " + this.lastName;
    }
};

console.log(person);
```

使用物件變量（Literal）建立物件

```
var book = {
    title: 'Beauty of JavaScript',
    isbn: '978-986-347-859-1',
    price: 400,
    displayTitle: function() {
        console.log(this.title);
    }
}

book.displayTitle();
```

建立物件：使用建構式

```
var book = new Object();

book.title = 'Beauty of JavaScript';
book.isbn = '978-986-347-859-1';
book.price = 400;

book.displayTitle = function() {
    console.log(this.title);
}

book.displayTitle();
```

❶ 建立一個空物件

❷ 為物件加入屬性

❸ 為物件加入方法

❹ 呼叫物件的方法

屬性存取（property access）

```
person.firstName;
person.fullName();
```

新的屬性可以直接指定

```
var person = {};
person.age = 18;
person.getAge = function() {
  return this.age;
}

console.log(person.getAge());
```

鍵值存取（key access）

```
var person = {};
person["weight"] = 80;
person.getWeight = function() {
    return this.weight;
}

console.log(person.getWeight());
```

存取屬性：使用變數

```
var person = {};
firstNameProperty = "firstName";
lastNameProperty = "lastName";
person[firstNameProperty] = "Leo";
person[lastNameProperty] = "Shiang";

console.log(
    person[firstNameProperty] + " " +
    person[lastNameProperty]);
```

不合法的屬性名稱

```
person["生日"] = "2018-10-10";
person[12] = 12;
person.12 = 12; // 錯誤

console.log(human.12)
```

刪除屬性：delete

```
delete person.firstName;
```

列舉屬性：Object.keys

回傳一個包含給定物件內所有可列舉屬性的字串陣列

```
var book = {
    title: 'Beauty of JavaScript',
    isbn: '978-986-347-859-1',
    price: 400,
    displayTitle: function() {
        console.log(this.title);
    }
}

console.log(Object.keys(book));
```

建構函數（constructor function）

```
function Person(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.fullName = function() {
        return this.firstName + " " + this.lastName;
    }
}

var me = new Person("Leo", "Shiang");
console.log(me);
```

使用 Object.create

此 method 有兩個參數：

1. prototypeObject

   原型物件，新建立出來的物件其原型會是此參數所傳入的原型物件。

2. propertoesObject

   新建立物件的屬性。

第一個參數：原型物件

建立物件但是沒有 prototype

```
var person = Object.create(null);
person.name = "Leo";

console.log(typeof(person));
console.log(person.prototype);
```

第一個參數：原型物件

建立物件並傳入 prototype

```
var prototypeObject = {
    fullName: function() {
        return this.firstName + ' ' + this.lastName;
    }
};

var person = Object.create(prototypeObject);
person.firstName = 'Leo';
person.lastName = 'Shiang';

console.log(person.fullName());
```

列舉屬性：for… in

回傳物件內所有可列舉與 prototype 的屬性

```
var prototypeObject = {
    fullName: function() {
        return this.firstName + ' ' + this.lastName;
    }
};

var person = Object.create(prototypeObject);
person.firstName = 'Leo';
person.lastName = 'Shiang';
for (var property in person) {
    console.log(property);
}

console.log(Object.keys(person));
```

第二個參數：屬性物件

propertiesObject 是定義以下幾種屬性的描述

1. Configurable
2. Enumerable
3. Value
4. 4. writable

第二個參數：屬性物件

```
var prototypeObject = {
    fullName: function() {
        return this.firstName + " " + this.lastName;
    }
};

var person = Object.create(prototypeObject, {
    firstName: {
        value: "Leo", writable: false, enumerable: true
    },
    lastName: {
        value: "Shiang", writable: true, enumerable: false
    }
});

console.log(Object.keys(person));
```

取得所有屬性：GetOwnPropertyNames

```
var prototypeObject = {
    fullName: function() {
        return this.firstName + " " + this.lastName;
    }
};
var person = Object.create(prototypeObject, {
    firstName: {
        value: "Leo", writable: false, enumerable: true
    },
    lastName: {
        value: "Shiang", writable: true, enumerable: false
    }
});

console.log(Object.getOwnPropertyNames(person));
```

工廠模式

```
function createBook(title, isbn, price) {
    var book = new Object();
    book.title = 'Beauty of JavaScript';
    book.isbn = '978-986-347-859-1';
    book.price = 400;
    book.displayTitle = function() {
        console.log(this.title);
    }
    return book;
}
var book = createBook(
    'Beauty of JavaScript', '978-986-347-859-1', 400);

console.log(book);
```

建構函數

```
function Book(title, isbn, price) {
    this.title = 'Beauty of JavaScript';
    this.isbn = '978-986-347-859-1';
    this.price = 400;
    this.displayTitle = function() {
        console.log(this.title);
    }
}

var book1 = new Book('Beauty of JavaScript',
'978-986-347-859-1', 400);

var book2 = new Book('Effective JavaScript',
'978-986-276-892-1', 450);
```

```
console.log(book1.constructor === Book);
console.log(book2.constructor === Book);
console.log(book1 instanceof Book);
console.log(book2 instanceof Book);
console.log(book1 instanceof Object);
console.log(book2 instanceof Object);
```

建構函數當作一般函數使用

```
function Book(title, isbn, price) {
    this.title = 'Beauty of JavaScript';
    this.isbn = '978-986-347-859-1';
    this.price = 400;
    this.displayTitle = function() {
        console.log(this.title);
    }
}

var book1 = new Book('Beauty of JavaScript',
'978-986-347-859-1', 400);
var book2 = new Book('Effective JavaScript',
'978-986-276-892-1', 450);
```

```
console.log(book1.constructor === Book);
console.log(book2.constructor === Book);
console.log(book1 instanceof Book);
console.log(book2 instanceof Book);
console.log(book1 instanceof Object);
console.log(book2 instanceof Object);
```

建構函數的問題

```
function Book(title, isbn, price) {
    this.title = 'Beauty of JavaScript';
    this.isbn = '978-986-347-859-1';
    this.price = 400;
    this.displayTitle =
        new Function('console.log(this.title)');
}
var book1 = new Book('Beauty of JavaScript’,
    '978-986-347-859-1', 400);
var book2 = new Book('Effective JavaScript’,
    '978-986-276-892-1', 450);

console.log(book1.displayTitle == book2.displayTitle);
```

```javascript
console.log(book1.constructor === Book);
console.log(book2.constructor === Book);
console.log(book1 instanceof Book);
console.log(book2 instanceof Book);
console.log(book1 instanceof Object);
console.log(book2 instanceof Object);
```

原型模式

- 我們建立的每個函數都有一個 prototype屬性
- 這個屬性是一個指向物件的指標
- 這個物件的用途是包含可以由特定類別的所有 instance 所共享的屬性和方法

原型模式範例

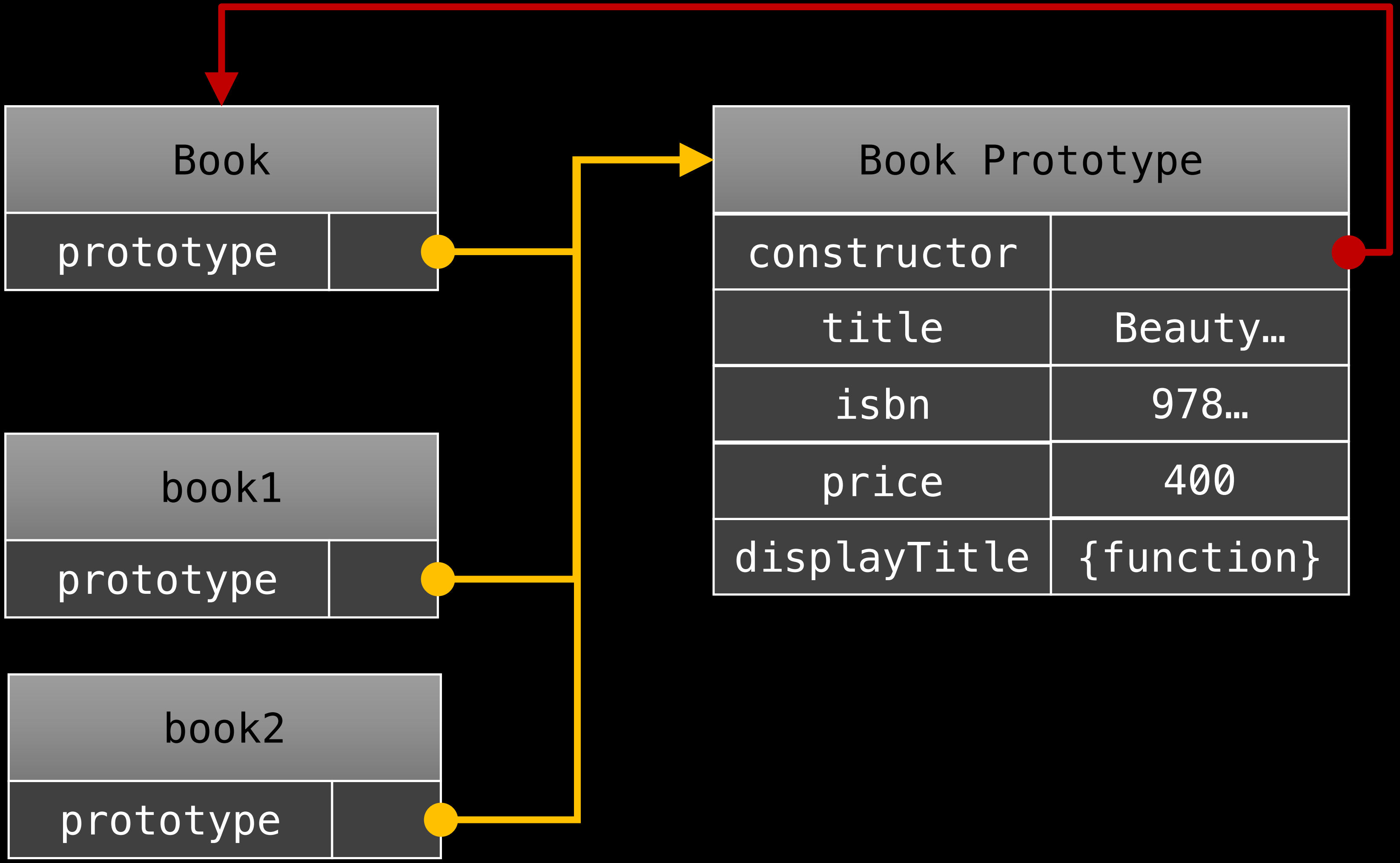
```
function Book() {}

Book.prototype.title = 'Beauty of JavaScript';
Book.prototype.isbn = '978-986-347-859-1';
Book.prototype.price = 400;
Book.prototype.displayTitle = function () {
    console.log(this.title);
};
```

```
var book1 = new Book();
book1.displayTitle();

var book2 = new Book();
book2.displayTitle();

console.log(book1.displayTitle === book2.displayTitle);
```

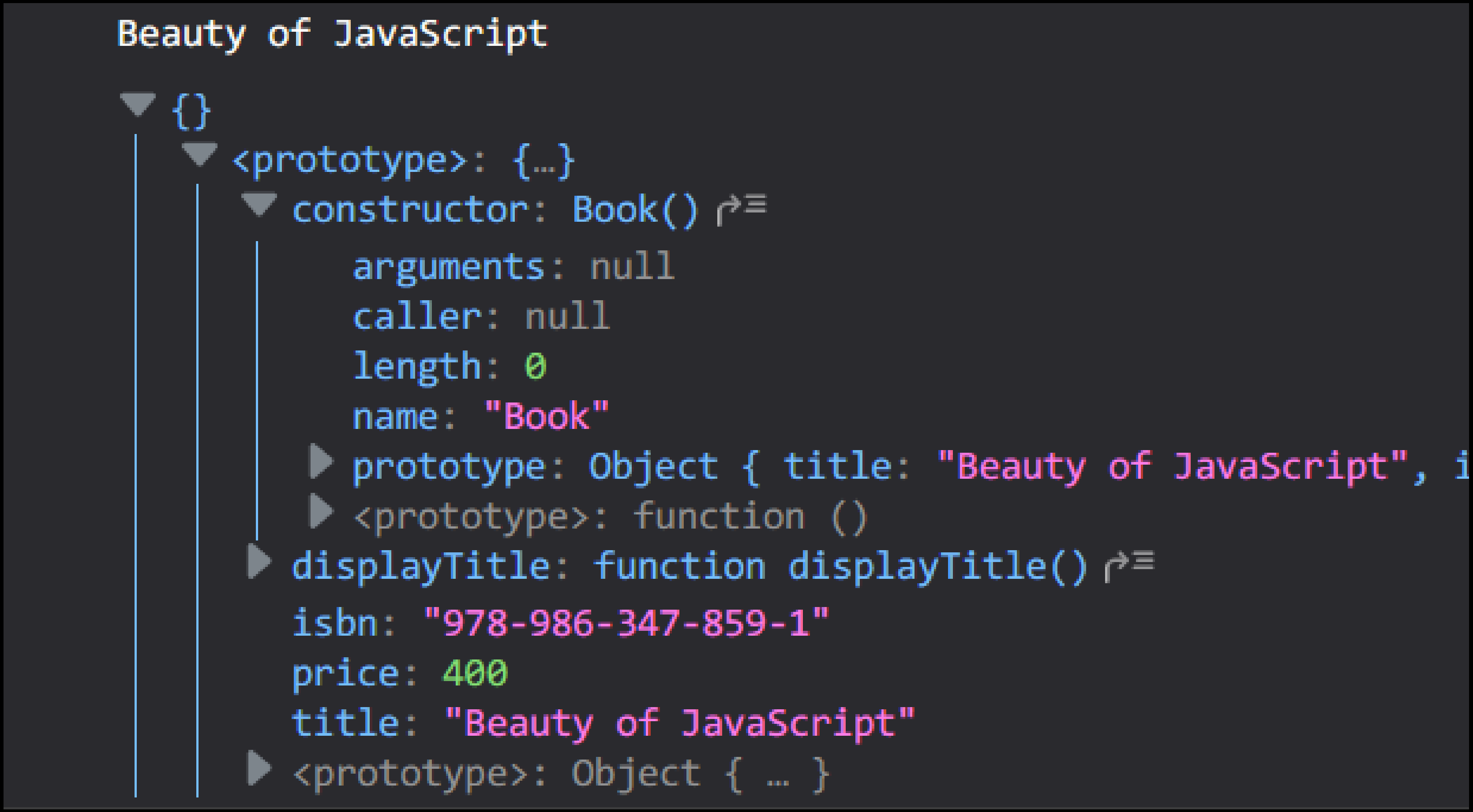
原型模式

- 只要建立一個函數，就會為該函數建立一個 prototype屬性，這個屬性指向函數的原型物件
- 所有的原型物件都會有一個 constructor 屬性，這個屬性指向 prototype 屬性所在的函數

原型模式


Book
prototype
book1
prototype
book2
prototype
Book Prototype
constructor
title
Beauty…
isbn
978…
price
400
displayTitle
{function}

原型模式



```
Beauty of JavaScript

▼ {}
  ▼ <prototype>: {…}
    ▼ constructor: Book()
        arguments: null
        caller: null
        length: 0
        name: "Book"
      ▶ prototype: Object { title: "Beauty of JavaScript", i
      ▶ <prototype>: function ()
    ▶ displayTitle: function displayTitle()
      isbn: "978-986-347-859-1"
      price: 400
      title: "Beauty of JavaScript"
    ▶ <prototype>: Object { … }
```

結合建構函數與原型模式

- 建構函數用來定義 instance 的屬性
- 原型模式用來定義方法與共享的屬性

結合建構函數與原型模式

```
function Task(name, start, end) {
    this.name = name;
    this.start = new Date(start);
    this.end = new Date(end);
    this.subTasks = [];
}

Task.prototype = {
    constructor: Task,
    getDuration: function() {
        return this.end - this.start;
    }
}
```

結合建構函數與原型模式

```
var task1 = new Task(
'需求訪談', '2018-10-23', '2018-10-30’);

var task2 = new Task(
    '系統設計', '2018-11-01', '2018-11-12');

console.log(task1.subTasks === task2.subTasks);
console.log(task1.getDuration === task2.getDuration);
```

寄生建構函數

```
function SpecialArray() {
    var instance = new Array();
    instance.push.apply(instance, arguments);
    instance.toPipedString = function() {
        return this.join("|");
    };
    return instance;
}

var colors = new SpecialArray("red", "blue", "green");

console.log(colors.toPipedString());
```