# JavaScript设计模式

# 大纲

- 基础
- 封装
- 继承
- 单体模式
- 链式调用
- 接口

# 基础

# 匿名函数

```
(function () {
    /*
      do something
    */
})();
```

```
(function () {
    /*
      do something
    */
}());
```

# 闭包

```
function init() {
    var name = "JavaScript";
    function displayName() {
        alert(name);
    }
    displayName();
}
init();
```

# 闭包

```
function makeFunc() {
    var name = "JavaScript";
    function displayName() {
        alert(name);
    }
    return displayName;
}

var myFunc = makeFunc();
myFunc();
```

# 构造器

```
var Anim = function () {

};

Anim.prototype.start = function() {

};

Anim.prototype.stop = function() {

};
```

# 封装

```
var Book = function(isbn, title, author) {
    this.setIsbn(isbn);
    this.setTitle(title);
    this.setAuthor(author);
};

Book.prototype = {
    checkIsbn: function(isbn) {

    },
    getIsbn: function() {
        return this.isbn;
    },
    setIsbn: function() {
        if(!this.checkIsbn(isbn)) throw new Error('Book: Invalid ISBN.');
        this.isbn = isbn;
    },
    getTitle: function() {
        return this.title;
    },
    setTitle: function(title) {
        this.title = title || 'No title specified';
    },
    getAuthor: function() {
        return this.author;
    },
    setAuthor: function(author) {
        this.author = author || 'No author specified';
    },
    display: function() {

    }
};
```

```
var Book = function(isbn, title, author) {
    this.setIsbn(isbn);
    this.setTitle(title);
    this.setAuthor(author);
};

Book.prototype = {
    checkIsbn: function(isbn) {

    },
    getIsbn: function() {
        return this._isbn;
    },
    setIsbn: function() {
        if(!this.checkIsbn(isbn)) throw new Error('Book: Invalid ISBN.');
        this._isbn = isbn;
    },
    getTitle: function() {
        return this._title;
    },
    setTitle: function(title) {
        this._title = title || 'No title specified';
    },
    getAuthor: function() {
        return this._author;
    },
    setAuthor: function(author) {
        this._author = author || 'No author specified';
    },
    display: function() {

    }
};
```

# 还有什么写法?

```
var Book = function(newIsbn, newTitle, newAuthor) {

    var isbn, title, author; //私有属性

    function checkIsbn(isbn) {} //私有方法

    this.getIsbn = function() { return isbn; } //公开特权方法

    this.setIsbn = function(newIsbn) {
        if(!this.checkIsbn(newIsbn)) throw new Error('Book: Invalid ISBN.');
        isbn = newIsbn;
    }

    this.getTitle = function() { return title; }

    this.setTitle = function(newTitle) {
        title = newTitle || 'No title specified';
    }

    this.getAuthor = function(Author) { return author; }

    this.setAuthor = function(newAuthor) {
        author = newAuthor || 'No author specified';
    }

    this.setIsbn(newIsbn);
    this.setTitle(newTitle);
    this.setAuthor(newAuthor);
};

Book.prototype = {
    display: function() {} //公开非特权方法
};
```

# 优点是显而易见的，缺点是什么?

- 耗费更多内存
- 私有属性和方法无法继承

```
var Book = (function() {

    var numOfBooks = 0; //静态私有属性

    function checkIsbn(isbn) {} //静态私有方法

    return function(newIsbn, newTitle, newAuthor) {
        var isbn, title, author; //私有属性

        this.getIsbn = function() { return isbn; }; //公开特权方法
        this.setIsbn = function(newIsbn) {
            if(!checkIsbn(newIsbn)) throw new Error('Book: Invalid ISBN.');
            isbn = newIsbn;
        };
        this.getTitle = function() { return title; };
        this.setTitle = function(newTitle) {
            title = newTitle || 'No title specified';
        };
        this.getAuthor = function() { return author;};
        this.setAuthor = function(newAuthor) {
            author = newAuthor || 'No author specified';
        };

        numOfBooks++;
        if(numOfBooks > 50) throw new Error('Book: Only 50 instances of Book can be created.');

        this.setIsbn(newIsbn);
        this.setTitle(newTitle);
        this.setAuthor(newAuthor);
    };

}());

Book.convertToTitleCase = function(inputString) {}; //静态公开方法

Book.prototype = {
    display: function() {} //公开非特权方法
};
```

# 继承

# 类式继承

```
function Person(name) {
    this.name = name;
}

Person.prototype.getName = function() {
    return this.name;
};
```

```
function Author(name, books) {
    Person.call(this, name);
    this.books = books;
}

Author.prototype = new Person();
Author.prototype.constructor = Author;
Author.prototype.getBooks = function() {
    return this.books;
}
```

```
function extend(subClass, superClass) {
    var F = function() {};
    F.prototype = superClass.prototype;
    subClass.prototype = new F();
    subClass.prototype.constructor = subClass;
}

function Person(name) {
    this.name = name;
}

Person.prototype.getName = function() {
    return this.name;
};

function Author(name, books) {
    Person.call(this, name);
    this.books = books;
}

extend(Author, Person);

Author.prototype.getBooks = function() {
    return this.books;
}
```

```
function extend(subClass, superClass) {
    var F = function() {};
    F.prototype = superClass.prototype;
    subClass.prototype = new F();
    subClass.prototype.constructor = subClass;

    subClass.superclass = superClass.prototype; //增加superClass属性
    if(superClass.prototype.constructor === Object) {
        superClass.prototype.constructor = superClass;
    }
}

function Person(name) {
    this.name = name;
}

Person.prototype.getName = function() {
    return this.name;
}

function Author(name, books) {
    Author.superclass.constructor.call(this, name);
    this.books = books;
}

extend(Author, Person);
Author.prototype.getBooks = function() {
    return this.books;
};

Author.prototype.getName = function() {
    var name = Author.superclass.getName.call(this);
    return name + ', Author of ' + this.getBooks().join(', ');
}
```

# 原型式继承

```
function clone(object) {
    function F() {}
    F.prototype = object;
    return new F;
}

var Person = {
    name: 'default name',
    getName: function() {
        return this.name;
    }
};

var Author = clone(Person);
Author.books = [];
Author.getBooks = function() {
    return this.books;
}
```

# 思考：有什么问题?

```
var authors = [];

authors[0] = clone(Author);
authors[0].name = 'Dustin Diaz';
authors[0].books[0] = 'JavaScript Design Patterns';

authors[1] = clone(Author);
authors[1].name = 'Ross Harmes';
authors[1].books[0] = 'JavaScript Language';
```

# 单体模式

# 基本形式

```
var Singleton = {
    attribute1: true,
    attribute2: 10,

    method1: function() {

    },

    method2: function() {

    }
};

Singleton.attribute1 = false;
var total = Singleton.attribute2 + 5;
var result = Singleton.method1();
```

# 封装

```
var DataParser = {
    //私有方法
    _stripWhitespace: function(str) {
        return str.replace(/\s+/, '');
    },
    _stringSplit: function(str, delimiter) {
        return str.split(delimiter);
    },
    //公开方法
    stringToArray: function(str, delimiter, stripWS) {
        if(stripWS) {
            str = this._stripWhitespace(str);
        }
        var outputArray = this._stringSplit(str, delimiter);
        return outputArray;
    }

};
```

# 封装

```
var DataParser = (function(){
    //私有变量
    var whitespaceRegex = /\s+/;

    //私有方法
    function stripWhitespace(str) {
        return str.replace(whitespaceRegex, '');
    }
    function stringSplit(str, delimiter) {
        return str.split(delimiter);
    }

    return {
        //公开方法
        stringToArray: function(str, delimiter, stripWS) {
            if(stripWS) {
                str = stripWhitespace(str);
            }
            var outputArray = stringSplit(str, delimiter);
            return outputArray;
        }
    };
}());
```

# 分支

```
var Singleton = (function() {
    var objectA = {
        method1: function() {},
        method2: function() {}
    };
    var objectB = {
        method1: function() {},
        method2: function() {}
    };
    return (someCondition) ? objectA : objectB;
}());
```

```
var simpleXhrFactory = (function(){
    var standard = {
        createXhrObject: function() {
            return new XMLHttpRequest();
        }
    };
    var activeXNew = {
        createXhrObject: function() {
            return new ActiveXObject('Msxml2.XMLHTTP');
        }
    };
    var activeXOld = {
        createXhrObject: function() {
            return new ActiveXObject('Microsoft.XMLHTTP');
        }
    };
    var testObject;
    try {
        testObject = standard.createXhrObject();
        return standard;
    } catch(e) {
        try {
            testObject = activeXNew.createXhrObject();
            return activeXNew;
        } catch(e) {
            try {
                testObject = activeXOld.createXhrObject();
                return activeXOld;
            } catch(e) {
                throw new Error('No XHR object found.');
            }
        }
    }
}());
```

# 链式调用

```
var Kitten = function() {
    this.name = 'Garfield';
    this.color = 'brown';
    this.gender = 'male';
};

Kitten.prototype.setName = function(name) {
    this.name = name;
    return this;
};

Kitten.prototype.setColor = function(color) {
    this.color = color;
    return this;
};

Kitten.prototype.setGender = function(gender) {
    this.gender = gender;
    return this;
};

Kitten.prototype.save = function(){
    console.log('saving ' + this.name + ', the ' + this.color + ' ' + this.gender + '
kitten..')
    return this;
};

new Kitten()
    .setName('Bob')
    .setColor('black')
    .setGender('male')
    .save();
```

# jQuery的链式调用

```
$('ul.first')
    .find('.foo')
    .css('background-color', 'red')
    .end()
    find('.bar')
    .css('background-color', 'green');
```

# Promise的链式调用

```
getJSON('/post/1.json')
    .then(function(post) {
        return getJSON(post.commentURL);
    })
    .then(function(comments) {
        console.log('Resolved: ', comments);
    }, function(error) {
        console.log('Rejected: ', error);
    })
```

# 接口

# Java中的接口

```
public interface Dataoutput {
    void writeBoolean(boolean value) throws IOException;
    void writeByte(int value) throws IOException;
    void writeChar(int value) throws IOException;
    void writeShort(int value) throws IOException;
    void writeInt(int value) throws IOException;
}
```

```
public class DataOutputStream extends FilterOutputStream implements
DataOutput {
    public final void writeBoolean (boolean value) throws IOException
{
        write(value ? 1 : 0);
    }
}
```

# PHP中的接口

```php
interface MyInterface {
    public function interfaceMethod($argumentOne, $argumentTwo);
}

class Myclass implements MyInterface {
    public function interfaceMethod($argumentOne, $argumentTwo) {
        return $argumentOne . $argumentTwo;
    }
}
```

# C#中的接口

```
interface MyInterface {
    string interfaceMethod(string argumentOne, String argumentTwo);
}

class Myclass : MyInterface {
    public string interfaceMethod(string argumentOne, string
argumentTwo) {
        return argumentOne + argumentTwo;
    }
}
```

```
var Composite = new Interface('Composite', ['add', 'remove',
'getChild']);
var FormItem = new Interface('FormItem', ['save']);

var CompositeForm = function(id, method, action) {
};

CompositeForm.prototype.add = function(child) {};

CompositeForm.prototype.remove = function(child) {};

CompositeForm.prototype.getChild = function(child) {};

CompositeForm.prototype.save = function(child) {};

function addForm(formInstance) {
    Interface.ensureImplements(formInstance, Composite, FormItem);
    /* do something */
}
```

```
var Interface = function(name, methods) {
    if(arguments.length !== 2) {
        throw new Error('Interface constructor called with ' + arguments.length + ' arguments, but
expected exactly 2.');
    }
    this.name = name;
    this.methods = [];
    for(var i = 0, len = methods.length; i < len; i++) {
        if(typeof methods[i] !== 'string') {
            throw new Error('Interface constructor expects method names to be passed in as a string.');
        }
        this.methods.push(methods[i]);
    }
}

Interface.ensureImplements = function(object) {
    if(arguments.length < 2) {
        throw new Error('Function Interface.ensureImplements called with ' + arguments.length + '
arguments, but expected exactly 2.');
    }

    for(var i = 1, len = arguments.length; i < len; i++) {
        var interface = arguments[i];
        if(interface.constructor !== Interface) {
            throw new Error('Function Interface.ensureImplements expects arguments two and above to be
instances of Interface.');
        }
        for(var j = 0, methodsLen = interface.methods.length; j < methodsLen; j++) {
            var method = interface.methods[j];
            if(!object[method] || typeof object[method] !== 'function') {
                throw new Error('Function Interface.ensureImplements: object does not implement the ' +
interface.name + ' interface. Method ' + method + ' was not found.');
            }
        }
    }
};
```

# 扩展：设计模式

- 工厂模式
- 桥接模式
- 组合模式
- 门面模式
- 适配器模式
- 装饰者模式
- 享元模式
- 代理模式
- 观察者模式
- 命令模式
- 职责链模式

# 参考资料

- 《Pro JavaScript Design Patterns》
- MDN https://developer.mozilla.org/zh-CN/
- http://es6.ruanyifeng.com/