JS工具开发

为什么要开发JS工具，是为了提高代码的复用性、灵活性、可维护性，将常用的功能封装成公用的方法或方法集合。

1. 面向对象语言（Objective Oriented OO）

面向对象是指一种程序设计范型，同时也是一种程序开发的方法。对象指的是类的集合。它将对象作为程序的基本单元，将程序和数据封装其中，以提高软件的重用性、灵活性和扩展性。

举个例子：一个汽车制造厂就好比一个项目，面向对象的开发模式就好比汽车制造过程分为汽车动力系统，空调系统，电子系统，内饰布置等，其中任何一个系统又是由各种小部件组成的。每个小部件只处理自己的任务，只管好自己的功能。

2. 类（Class）

类是面向对象程序设计（OOP，Object-Oriented Programming）实现信息封装的基础。类是一种用户定义类型，也称类类型。每个类包含数据说明和一组操作数据或传递消息的函数。类的实例称为对象。

类是一个抽象的概念，完整一个功能或数据结构的设计和定义，要具体使用必须实例化，类实例化以后称为对象，会被分配内存空间，可以直接调用类里面的数据（属性）和功能（方法），类要有生命周期，使用完以后要有销毁和释放内存的实现。

类就好比汽车制造过程中的每一个小部件的设计蓝本，生产车床根据零件蓝本制造小部件就好比实例化，小部件（对象）继续用于其他大部件或系统的一个功能单元发挥作用。

3. 接口（Interface）

接口为面向对象程序设计语言提供了一组可以自定义的规范，一个类可以实现多个接口，来达到多个功能集合的实现。

一个汽车制造厂不可能所有零部件都去自己生产，它可能会使用A品牌的引擎，B品牌的空调，C品牌的汽车挡风玻璃等等，那么其他厂商提供的零部件如何保证都能给一辆汽车使用呢？这就需要汽车设计部门提供一套标准和规范，规定某个零件必须实现什么功能，插口或卡槽长什么样，要使用某个功能如何去启动等等。这就是接口在程序中的作用，程序的某个模块可能今天要这样实现，明天要那样实现，所以先制定好一个功能列表（接口），具体实现就不会影响其他模块的开发了。

4. 封装

封装是面向对象设计语言某个功能的具体实现对于使用它的其他模块来说是隐藏不可见的，为什么要隐藏？因为没必要暴露，其他模块使用我的功能就好，不用管我是怎么实现的，暴露还有一个弊端，如果调用者不小心改变了某个功能内部的一个变量或环节，就会造成这个功能出现不可预期的问题。封装结合接口通过设计模式是面向对象语言的一大特色。

汽车制造厂的某个零部件完全不需要把自己的内部结构暴露给其他零件，只要能让其他零件顺利安装并使用它就可以了。

5. 继承

面向对象语言的类可以继承，子类继承父类将获得父类的所有功能，子类通常会在父类的基础上扩展功能或者重新实现父类的原有功能。

汽车零件的升级或改良就可以看做是一个子类继承父类，不但保留了原有零件的功能，而且功能可以得到扩展。

6. 多态

多态指同一个实体同时具有多种形式。它是面向对象程序设计（OOP）的一个重要特征。如果一个语言只支持类而不支持多态，只能说明它是基于对象的，而不是面向对象的。

比如子类可以重新父类的方法，同样的方法运行的效果却不同，这是一种多态，再比如一个方法可以有不同的参数列表以及不同的实现，这也是一种多态。

7. 设计模式

为什么要有设计模式？一个复杂的项目，多人同时参与开发，按模块分配任务，如果张三这样写，李四那样写，在整合以及日后的代码维护上就会出现各种问题，接口不统一，无法扩展，数据结构不一致等等。那么引入合适的设计模式就能让多人协同的项目开发变得事半功倍。

设计开发对于汽车制造厂这个案例来说就好比自动化的工业流水线，引导整个制造过程有条不紊的高效进行。

总体来说设计模式分为三大类：

- 创建型模式，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。
- 结构型模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。
- 行为型模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

设计模式的六大原则：

- 总原则：开闭原则（Open Close Principle）

开闭原则就是说对扩展开放，对修改关闭。在程序需要进行拓展的时候，不能去修改原有的代码，而是要扩展原有代码，实现一个热插拔的效果。所以一句话概括就是：为了使程序的扩展性好，易于维护和升级。想要达到这样的效果，我们需要使用接口和抽象类等

- 单一职责原则

不要存在多于一个导致类变更的原因，也就是说每个类应该实现单一的职责，如若不然，就应该把类拆分。

- 里氏替换原则（Liskov Substitution Principle）

里氏代换原则(Liskov Substitution Principle LSP)面向对象设计的基本原则之一。里氏代换原则中说，任何基类可以出现的地方，子类一定可以出现。 LSP是继承复用的基石，只有当衍生类可以替换掉基类，软件单位的功能不受到影响时，基类才能真正被复用，而衍生类也能够在基类的基础上增加新的行为。里氏代换原则是对“开-闭”原则的补充。实现“开-闭”原则的关键步骤就是抽象化。而基类与子类的继承关系就是抽象化的具体实现，所以里氏代换原则是对实现抽象化的具体步骤的规范。

里氏替换原则中，子类对父类的方法尽量不要重写和重载。因为父类代表了定义好的结构，通过这个规范的接口与外界交互，子类不应该随便破坏它。

- 依赖倒转原则（Dependence Inversion Principle）

这个是开闭原则的基础，具体内容：面向接口编程，依赖于抽象而不依赖于具体。

写代码时用到具体类时，不与具体类交互，而与具体类的上层接口交互。

- 接口隔离原则（Interface Segregation Principle）

这个原则的意思是：每个接口中不存在子类用不到却必须实现的方法，如果不然，就要将接口拆分。使用多个隔离的接口，比使用单个接口（多个接口方法集合到一个的接口）要好。

- 迪米特法则（最少知道原则）（Demeter Principle）

就是说：一个类对自己依赖的类知道的越少越好。也就是说无论被依赖的类多么复杂，都应该将逻辑封装在方法的内部，通过public方法提供给外部。这样当被依赖的类变化时，才能最小的影响该类。

最少知道原则的另一个表达方式是：只与直接的朋友通信。类之间只要有耦合关系，就叫朋友关系。耦合分为依赖、关联、聚合、组合等。我们称出现为成员变量、方法参数、方法返回值中的类为直接朋友。局部变量、临时变量则不是直接的朋友。我们要求陌生的类不要作为局部变量出现在类中。

- 合成复用原则（Composite Reuse Principle）

原则是尽量首先使用合成/聚合的方式，而不是使用继承。

8. 单例

口述+演示

9. 工厂方法

口述+演示

10.观察者模式

其实，简单来讲就一句话：当一个对象变化时，其它依赖该对象的对象都会收到通知，并且随着变化！对象之间是一种一对多的关系。

一个Observer接口：

[java] view plaincopy

```
1.  public interface Observer {
2.      public void update();
3.  }
```

两个实现类：

[java] view plaincopy

```
1.  public class Observer1 implements Observer {
2.
3.      @Override
4.      public void update() {
5.          System.out.println("observer1 has received!");
6.      }
7.  }
```

[java] view plaincopy

```
1.  public class Observer2 implements Observer {
```

```
1. public class Observer2 implements Observer {
2.
3.     @Override
```

```
4.     public void update() {
5.         System.out.println("observer2 has received!");
6.     }
7.
8. }
```

Subject接口及实现类：

[java] view plaincopy

```
1. public interface Subject {
2.
3.     /*增加观察者*/
4.     public void add(Observer observer);
5.
6.     /*删除观察者*/
7.     public void del(Observer observer);
8.
9.     /*通知所有的观察者*/
10.    public void notifyObservers();
11.
12.    /*自身的操作*/
13.    public void operation();
14. }
```

[java] view plaincopy

```
1. public abstract class AbstractSubject implements Subject
   {
2.
3.     private Vector<Observer> vector = new Vector<Observer
   >();
4.     @Override
5.     public void add(Observer observer) {
6.         vector.add(observer);
7.     }
8.
9.     @Override
10.    public void del(Observer observer) {
11.        vector.remove(observer);
```

```java
        Enumeration<Observer> enumo = vector.elements();
        while(enumo.hasMoreElements()){
            enumo.nextElement().update();
        }
    }
}
```

[java] view plaincopy

```java
public class MySubject extends AbstractSubject {

    @Override
    public void operation() {
        System.out.println("update self!");
        notifyObservers();
    }

}
```

测试类：

[java] view plaincopy

```java
public class ObserverTest {

    public static void main(String[] args) {
        Subject sub = new MySubject();
        sub.add(new Observer1());
        sub.add(new Observer2());

        sub.operation();
    }

}
```

输出：

```
update self!
observer1 has received!
observer2 has received!
```

11.采用TS语言和设计模式

TS是JS的升级，它是面向对象的语言，有类和接口，因此在这种情况下一些多态以及使用各种设计模式去开发项目，这样会很棒的感觉。

12.实例：图片懒加载工具

口述+演示