**单例模式（Singleton）**

保证一个类仅有一个实例，并提供一个访问它的全局访问点。

class Singleton {

constructor(data) {

if (Singleton.prototype.Instance === undefined) {

this.data = data

Singleton.prototype.Instance = this

}

return Singleton.prototype.Instance

}

}

const ob1 = new Singleton('one')

const ob2 = new Singleton('two')

ob1.data // 'one'

ob2.data // 'one'

ob1 === ob2 // true

**工厂模式（Factory）**

定义一个用于创建对象的接口，这个接口由子类决定实例化哪一个类。

class ProductManager {

constructor(productName) {

this.name = productName

}

\_createProductA() {}

\_createProductB() {}

factory() {

switch (this.name) {

case 'productA':

this.\_createProductA()

break

case 'productB':

this.\_createProductB()

break

}

}

}

const pm = new ProductManager()

const productA = pm.factory('productA')

**抽象工厂模式（AbstractFactory）**

抽象类是一种声明但不能使用的类，如果子类没有重写父类的抽象方法，将会报错。

// 抽象工厂类

class AbstractFactory {

constructor() {

if (new.target === AbstractFactory) {

throw new Error('不能直接使用抽象类')

}

}

createProductA(product) {

console.log('AbstractFactory createProductA')

}

createProductB(product) {

console.log('AbstractFactory createProductB')

}

}

// 子类重写父类方法

class ConcreteFactory1 extends AbstractFactory {

constructor() {

super()

}

createProductA(product) {

console.log('ConcreteFactory1 createProductA')

}

createProductB(product) {

console.log('ConcreteFactory1 createProductB')

}

}

// 子类重写父类方法

class ConcreteFactory2 extends AbstractFactory {

constructor() {

super()

}

createProductA(product) {

console.log('ConcreteFactory2 createProductA')

}

createProductB(product) {

console.log('ConcreteFactory2 createProductB')

}

}

**建造者模式（Builder）**

将一个对象复杂的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。  
四要素：

class ProductItem {

constructor(name){

}

}

class Product {

constructor() {

this.name = ''

this.Item = []

}

setName(productName) {

this.name = productName

}

}

class Builder {

constructor() {}

ProductA(){

let product = new Product();

product.setName("ProductA");

product.Item.push(new ProductItem("Item1"))

return product;

}

ProductB(){

let product = new Product();

product.setName("ProductB");

product.Item.push(new ProductItem("Item2"))

return product;

}

}