1. 单例模式（Singleton）

特点：

- 确保一个类只有一个实例。
- 提供一个全局访问点。
- 控制实例化过程，防止多个实例的创建。

示例（JavaScript）：

```
class Singleton {
    constructor() {
        if (!Singleton.instance) {
            Singleton.instance = this;
        }
        return Singleton.instance;
    }

    showMessage() {
        console.log("Hello, I am a singleton instance!");
    }
}

// Usage
const instance1 = new Singleton();
const instance2 = new Singleton();

console.log(instance1 === instance2);  // true
instance1.showMessage();  // Hello, I am a singleton instance!
```

2. 工厂方法模式（Factory Method）

特点：

- 定义一个创建对象的接口，但由子类决定实例化的类是哪一个。
- 延迟到子类以便创建对象。

示例（JavaScript）：

```
class Product {
    constructor(name) {
        this.name = name;
    }

    getName() {
        return this.name;
    }
}

class ProductFactory {
```

```
    createProduct(type) {
        if (type === 'A') {
            return new Product('Product A');
        } else if (type === 'B') {
            return new Product('Product B');
        }
    }
}

// Usage
const factory = new ProductFactory();
const productA = factory.createProduct('A');
const productB = factory.createProduct('B');

console.log(productA.getName());  // Product A
console.log(productB.getName());  // Product B
```

3. 观察者模式（Observer）

特点：

- 定义对象间的一对多依赖，当一个对象状态改变时，所有依赖的对象都会自动收到通知并更新。
- 松耦合，被观察者和观察者可以独立变化。

示例（JavaScript）：

```
class Subject {
    constructor() {
        this.observers = [];
    }

    addObserver(observer) {
        this.observers.push(observer);
    }

    notifyObservers(message) {
        this.observers.forEach(observer => observer.update(message));
    }
}

class Observer {
    constructor(name) {
        this.name = name;
    }

    update(message) {
        console.log(`${this.name} received message: ${message}`);
    }
}

// Usage
```

```
const subject = new Subject();
const observer1 = new Observer('Observer 1');
const observer2 = new Observer('Observer 2');

subject.addObserver(observer1);
subject.addObserver(observer2);

subject.notifyObservers('Hello Observers!');
// Observer 1 received message: Hello Observers!
// Observer 2 received message: Hello Observers!
```

4. 策略模式（Strategy）

特点：

- 定义一系列算法，把它们一个个封装起来，并且使它们可以互换。
- 使得算法可以独立于使用它的客户而变化。

示例（JavaScript）：

```
class Strategy {
    execute() {}
}

class ConcreteStrategyA extends Strategy {
    execute() {
        console.log("Executing Strategy A");
    }
}

class ConcreteStrategyB extends Strategy {
    execute() {
        console.log("Executing Strategy B");
    }
}

class Context {
    setStrategy(strategy) {
        this.strategy = strategy;
    }

    executeStrategy() {
        this.strategy.execute();
    }
}

// Usage
const context = new Context();
const strategyA = new ConcreteStrategyA();
const strategyB = new ConcreteStrategyB();
```

```
context.setStrategy(strategyA);
context.executeStrategy();  // Executing Strategy A

context.setStrategy(strategyB);
context.executeStrategy();  // Executing Strategy B
```

总结

1. **单例模式（Singleton）**：
   - **优点**：控制实例数量、提供全局访问点。
   - **缺点**：不易扩展、可能造成资源竞争。
2. **工厂方法模式（Factory Method）**：
   - **优点**：解耦对象创建过程、增强代码灵活性。
   - **缺点**：增加类数量、代码复杂度上升。
3. **观察者模式（Observer）**：
   - **优点**：实现对象间的动态联动、增强系统的灵活性和可扩展性。
   - **缺点**：可能导致过多的通知、增加系统复杂性。
4. **策略模式（Strategy）**：
   - **优点**：算法可以自由切换、避免使用多重条件判断。
   - **缺点**：客户端必须了解所有策略以便选择适当的策略、增加对象数量。