代理模式

概念

- 代理模式（Proxy Pattern）是一种结构型设计模式，它通过代理对象控制对原对象的访问。代理模式可以为对象的访问提供额外的功能，例如控制访问权限、延迟加载、日志记录等。
- 设计意图：代理模式的设计意图是为对象提供一个代理，以便在不改变对象接口的情况下，控制对对象的访问。代理模式可以用于以下几种情况：
  - 远程代理：为一个对象在不同地址空间提供局部代表。
  - 虚拟代理：根据需要创建开销很大的对象。
  - 保护代理：控制对原始对象的访问。
  - 智能引用代理：在访问对象时执行一些附加操作。
- 代理模式主要由以下几个部分构成：
  - 接口（Subject）：定义了代理类和真实类的公共接口。
  - 真实类（RealSubject）：实现了接口的类，是真正需要被代理的对象。
  - 代理类（Proxy）：实现了接口，并持有真实类的引用，控制对真实类的访问。
- 优点
  - 控制访问：可以控制对真实对象的访问，增加安全性。
  - 延迟加载：可以延迟加载开销大的对象，提升性能。
  - 附加功能：可以在访问对象时添加额外的功能，如日志记录、性能监控等。
- 缺点
  - 增加复杂性：引入代理对象会增加系统的复杂性。
  - 性能开销：代理模式会增加一些额外的性能开销，特别是在频繁访问的场景下。

实例

- 定义银行账户接口，代理类和真实类的公共接口

```
1 public interface IBankAccount
2 {
3     void Deposit(decimal amount);
4     void Withdraw(decimal amount);
5     decimal GetBalance();
6 }
```

- 定义银行账户，真实类

```
public class BankAccount : IBankAccount
{
    private decimal balance;

    public void Deposit(decimal amount)
    {
        balance += amount;
        Console.WriteLine($"Deposited: {amount}, New Balance: {balance}");
    }

    public void Withdraw(decimal amount)
    {
        if (amount <= balance)
        {
            balance -= amount;
            Console.WriteLine($"Withdrew: {amount}, New Balance: {balance}");
        }
        else
        {
            Console.WriteLine("Insufficient funds.");
        }
    }

    public decimal GetBalance()
    {
        return balance;
    }
}
```

- 定义代理类

```
public class BankAccountProxy : IBankAccount
{
    private BankAccount realAccount;
    private string authorizedUser;

    public BankAccountProxy(string user)
    {
        realAccount = new BankAccount();
        authorizedUser = user;
```

```
    }

    public void Deposit(decimal amount)
    {
        realAccount.Deposit(amount);
    }

    public void Withdraw(decimal amount)
    {
        if (IsAuthorized())
        {
            realAccount.Withdraw(amount);
        }
        else
        {
            Console.WriteLine("Unauthorized access attempt.");
        }
    }

    public decimal GetBalance()
    {
        if (IsAuthorized())
        {
            return realAccount.GetBalance();
        }
        else
        {
            Console.WriteLine("Unauthorized access attempt.");
            return 0;
        }
    }

    private bool IsAuthorized()
    {
        // 授权检查
        return authorizedUser == "authorizedUser";
    }
}
```

- 使用

```
static void Main(string[] args)
{
    IBankAccount account = new BankAccountProxy("authorizedUser");
    account.Deposit(100);
```

```
    account.Withdraw(50);
    Console.WriteLine($"Balance: {account.GetBalance()}");

    // 尝试未授权访问
    IBankAccount unauthorizedAccount = new
BankAccountProxy("unauthorizedUser");
    unauthorizedAccount.Withdraw(50);
    Console.WriteLine($"Balance: {unauthorizedAccount.GetBalance()}");
}
```

- 打印结果

```
Deposited: 100, New Balance: 100
Withdrew: 50, New Balance: 50
Balance: 50
Unauthorized access attempt.
Balance: 0
```