



深圳大学
Shenzhen University

第12-2讲

代理模式

软件体系结构与设计模式
Software Architecture & Design Pattern

深圳大学计算机与软件学院



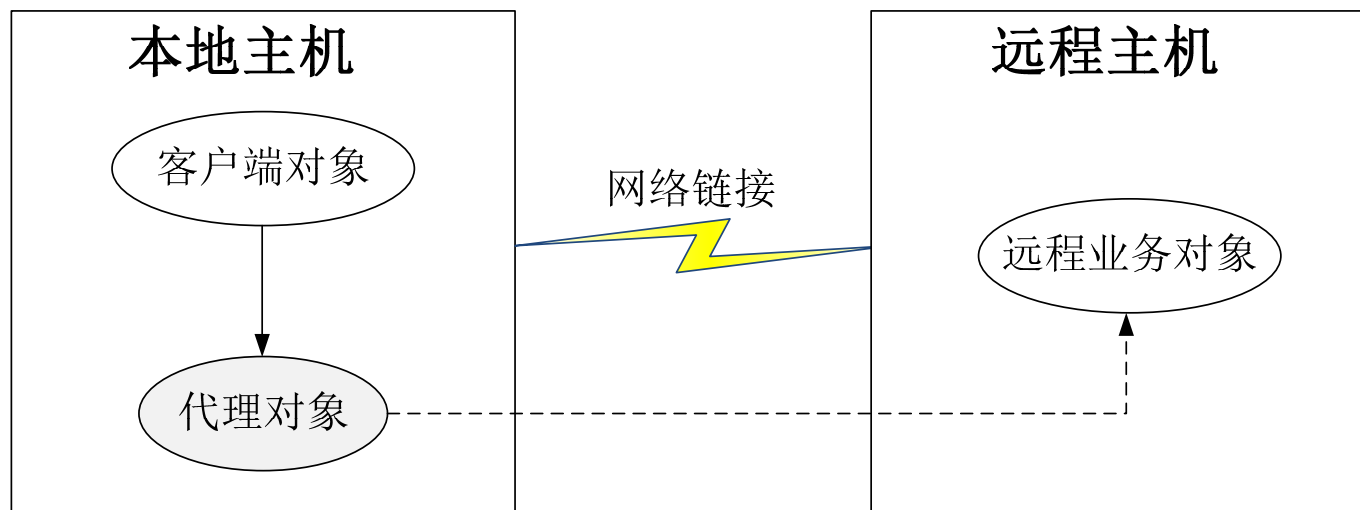
主要内容

- ◆ 代理模式动机与定义
- ◆ 代理模式结构与分析
- ◆ 代理模式实例与解析
- ◆ 代理模式效果与应用

代理模式动机

■ 代理模式动机

□ 代理小实例



小图片



大图片



代理模式动机

- 通过引入一个新的对象（如小图片和远程代理对象）来实现对真实对象的操作，或者将新的对象作为真实对象的一个替身
- 引入代理对象来间接访问一个对象 → 代理模式



代理模式定义

■ 对象结构型模式

代理模式：给某一个对象提供一个代理或占位符，并由代理对象来控制对原对象的访问。

Proxy Pattern: Provide a surrogate or placeholder for another object to control access to it.

- 引入一个新的代理对象
- 代理对象在客户端对象和目标对象之间起到中介的作用
- 去掉客户不能看到的内容和服务或者增添客户需要的额外的新服务

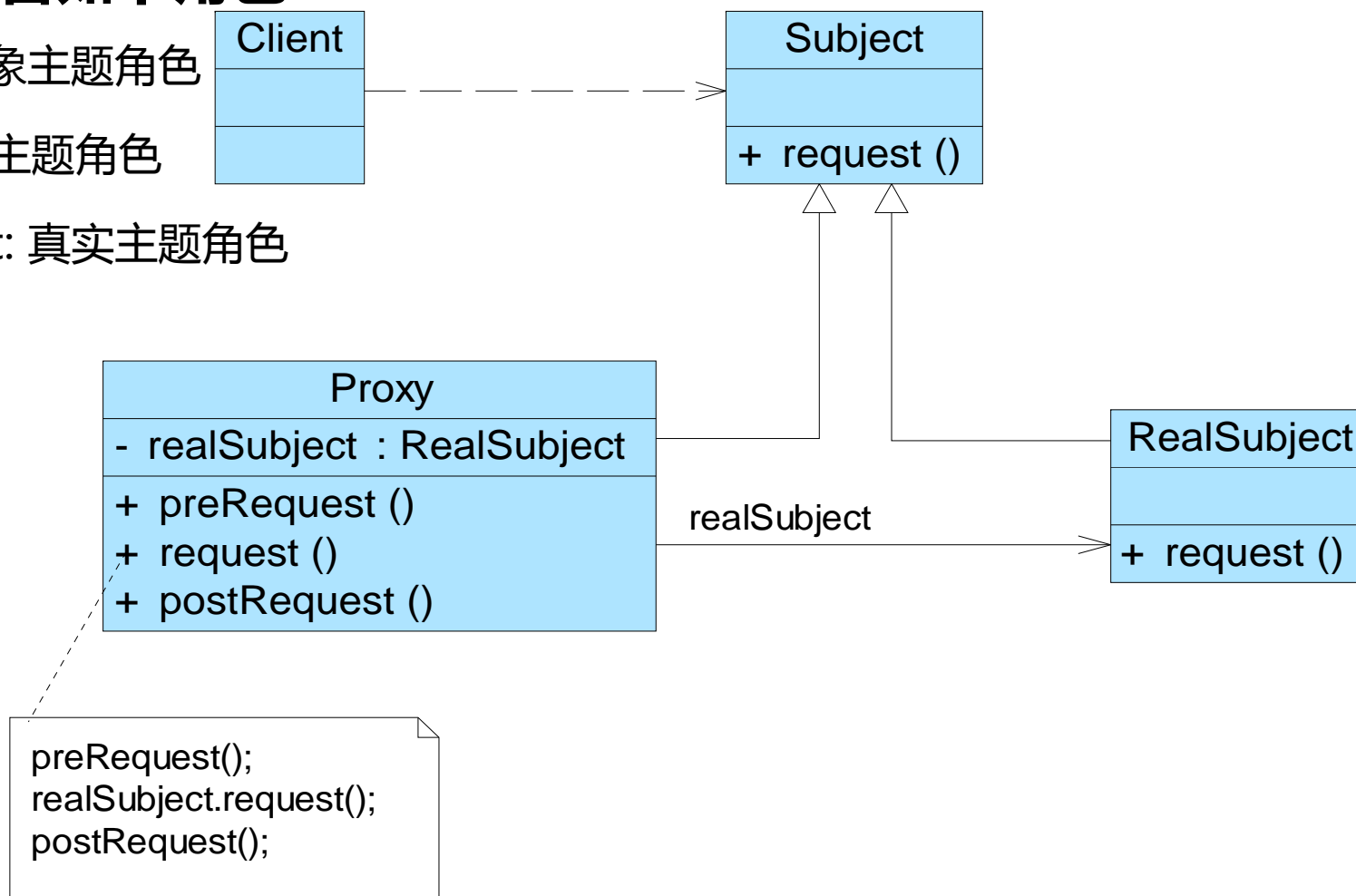
代理模式结构

■ 代理模式包含如下角色：

□ Subject: 抽象主题角色

□ Proxy: 代理主题角色

□ RealSubject: 真实主题角色





代理模式分析

■ 代理类示例代码：

```
public class Proxy extends Subject {  
    private RealSubject realSubject = new RealSubject(); //维持一个对真实主题  
    对象的引用  
    public void preRequest() {  
        .....  
    }  
  
    public void request() {  
        preRequest();  
        realSubject.request(); //调用真实主题对象的方法  
        postRequest();  
    }  
  
    public void postRequest() {  
        .....  
    }  
}
```



代理模式分析

■ 几种常用的代理模式

- 远程代理(Remote Proxy)：为一个位于不同的地址空间的对象提供一个本地的代理对象，这个不同的地址空间可以在同一台主机中，也可以在另一台主机中，远程代理又称为大使(Ambassador)
- 虚拟代理(Virtual Proxy)：如果需要创建一个资源消耗较大的对象，先创建一个消耗相对较小的对象来表示，真实对象只在需要时才会被真正创建



代理模式分析

- 保护代理(Protect Proxy)：控制对一个对象的访问，可以给不同的用户提供不同级别的使用权限
- 缓冲代理(Cache Proxy)：为某一个目标操作的结果提供临时的存储空间，以便多个客户端可以共享这些结果
- 智能引用代理(Smart Reference Proxy)：当一个对象被引用时，提供一些额外的操作，例如将对象被调用的次数记录下来等

■ 代理模式实例

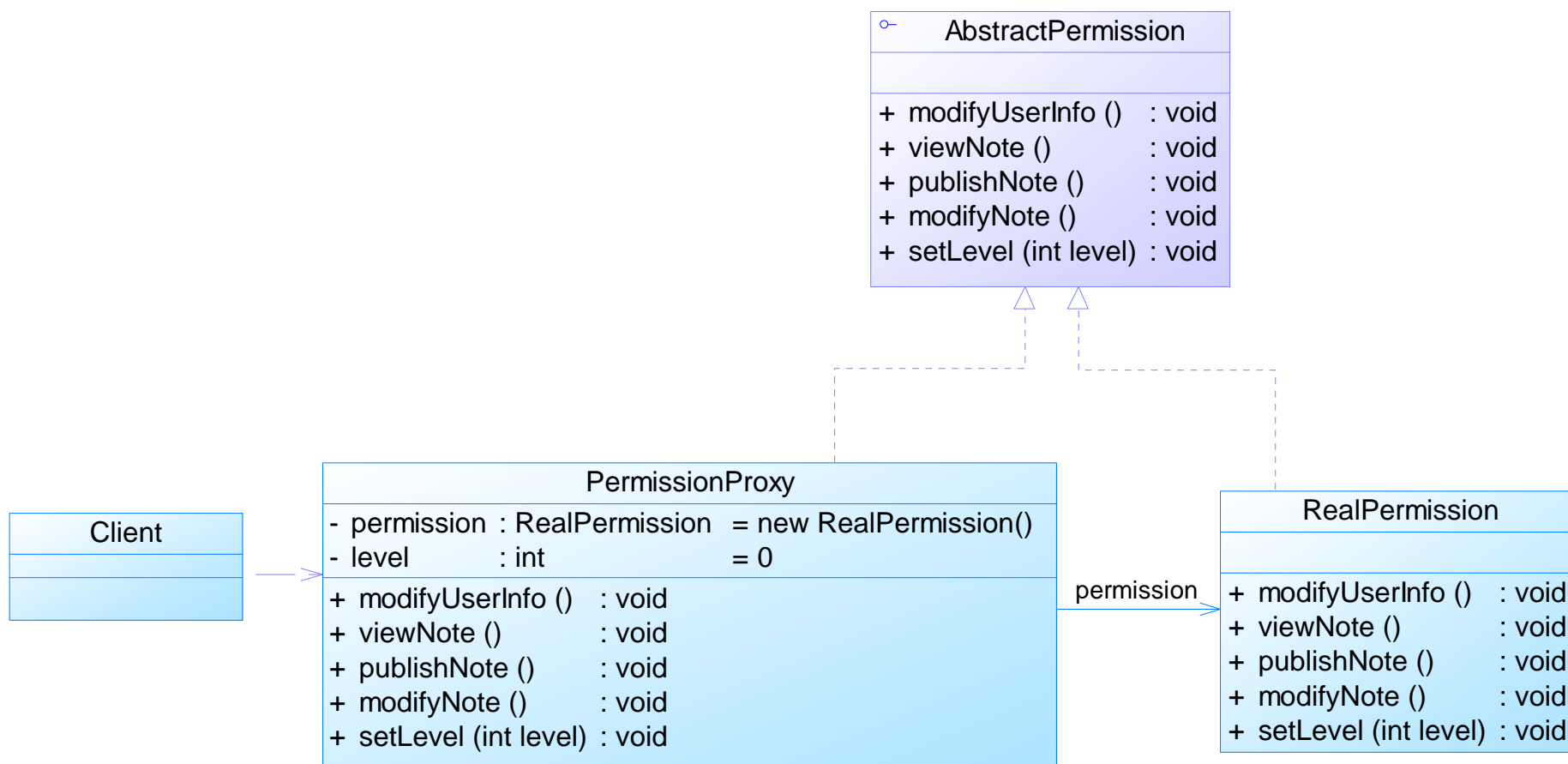
□ 论坛权限控制代理：实例说明

- 在一个论坛中已注册用户和游客的权限不同，已注册的用户拥有发帖、修改自己的注册信息、修改自己的帖子等功能；而游客只能看到别人发的帖子，没有其他权限。使用代理模式来设计该权限管理模块。
- 在本实例中我们使用代理模式中的保护代理，该代理用于控制对一个对象的访问，可以给不同的用户提供不同级别的使用权限。

代理模式实例与解析

■ 代理模式实例

□ 论坛权限控制代理：参考类图



代理模式实例与解析

■ 代理模式实例

□ 论坛权限控制代理：参考代码

■ DesignPatterns之proxy包

```
AbstractPermission.java ✖  
1 package proxy;  
2  
3 public interface AbstractPermission  
4 {  
5     public void modifyUserInfo();  
6     public void viewNote();  
7     public void publishNote();  
8     public void modifyNote();  
9     public void setLevel(int level);  
10 }
```

```
2
3 public class RealPermission implements AbstractPermission
4 {
5     public void modifyUserInfo()
6     {
7         System.out.println("修改用户信息!");
8     }
9
10    public void viewNote()
11    {
12        System.out.println("查看帖子!");
13    }
14
15    public void publishNote()
16    {
17        System.out.println("发布新帖!");
18    }
19
20    public void modifyNote()
21    {
22        System.out.println("修改发帖内容!");
23    }
24
25    public void setLevel(int level)
26    {
27    }
28 }
```

```
3 public class PermissionProxy implements AbstractPermission
4 {
5     private RealPermission permission=new RealPermission();
6     private int level=0;
7
8     public void modifyUserInfo()
9     {
10         if(0==level)
11         {
12             System.out.println("对不起，你没有该权限!");
13         }
14         else if(1==level)
15         {
16             permission.modifyUserInfo();
17         }
18     }
19
20     public void viewNote()
21     {
22         permission.viewNote();
23     }
24
25     public void publishNote()
26     {
27         if(0==level)
28         {
```

PermissionProxy.java

```
29         System.out.println("对不起，你没有该权限!");
30     }
31     else if(1==level)
32     {
33         permission.publishNote();
34     }
35 }
36
37 public void modifyNote()
38 {
39     if(0==level)
40     {
41         System.out.println("对不起，你没有该权限!");
42     }
43     else if(1==level)
44     {
45         permission.modifyNote();
46     }
47 }
48
49 public void setLevel(int level)
50 {
51     this.level=level;
52 }
53 }
```

Client.java

```
1 package proxy;
2
3 public class Client
4 {
5     public static void main(String args[])
6     {
7         AbstractPermission permission;
8         permission=(AbstractPermission)XMLUtil.getBean();
9
10        permission.modifyUserInfo();
11        permission.viewNote();
12        permission.publishNote();
13        permission.modifyNote();
14        System.out.println("-----");
15        permission.setLevel(1);
16        permission.modifyUserInfo();
17        permission.viewNote();
18        permission.publishNote();
19        permission.modifyNote();
20    }
21 }
```

Proxyconfig.xml

```
1 <?xml version="1.0"?>
2 <config>
3     <className>proxy.PermissionProxy</className>
4 </config>
```


XMLUtil.java

```
7 public class XMLUtil
8 {
9     //该方法用于从XML配置文件中提取具体类名，并返回一个实例对象
10    public static Object getBean()
11    {
12        try
13        {
14            //创建文档对象
15            DocumentBuilderFactory dFactory = DocumentBuilderFactory.newInstance();
16            DocumentBuilder builder = dFactory.newDocumentBuilder();
17            Document doc;
18            doc = builder.parse(new File("Proxyconfig.xml"));
19
20            //获取包含类名的文本节点
21            NodeList nl = doc.getElementsByTagName("className");
22            Node classNode=nl.item(0).getFirstChild();
23            String cName=classNode.getNodeValue();
24
25            //通过类名生成实例对象并将其返回
26            Class c=Class.forName(cName);
27            Object obj=c.newInstance();
28            return obj;
29        }
```

XMLUtil.java

```
30        catch(Exception e)
31        {
32            e.printStackTrace();
33            return null;
34        }
35    }
36 }
```



代理模式效果与应用

■ 代理模式优点：

- 能够协调调用者和被调用者，在一定程度上降低了系统的耦合度
- 客户端可以针对抽象主题角色进行编程，增加和更换代理类无须修改源代码，符合开闭原则，系统具有较好的灵活性和可扩展性
- 远程代理：可以将一些消耗资源较多的对象和操作移至性能更好的计算机上，提高了系统的整体运行效率



代理模式效果与应用

■ 代理模式优点：

- **虚拟代理**：通过一个消耗资源较少的对象来代表一个消耗资源较多的对象，可以在一定程度上节省系统的运行开销
- **缓冲代理**：为某一个操作的结果提供临时的缓存存储空间，以便在后续使用中能够共享这些结果，优化系统性能，缩短执行时间
- **保护代理**：可以控制对一个对象的访问权限，为不同用户提供不同级别的使用权限



代理模式效果与应用

■ 代理模式缺点：

- 由于在客户端和真实主题之间增加了代理对象，因此有些类型的代理模式可能会造成请求的处理速度变慢（例如保护代理）
- 实现代理模式需要额外的工作，而且有些代理模式的实现过程较为复杂（例如远程代理）



代理模式效果与应用

■ 在以下情况下可以使用代理模式：

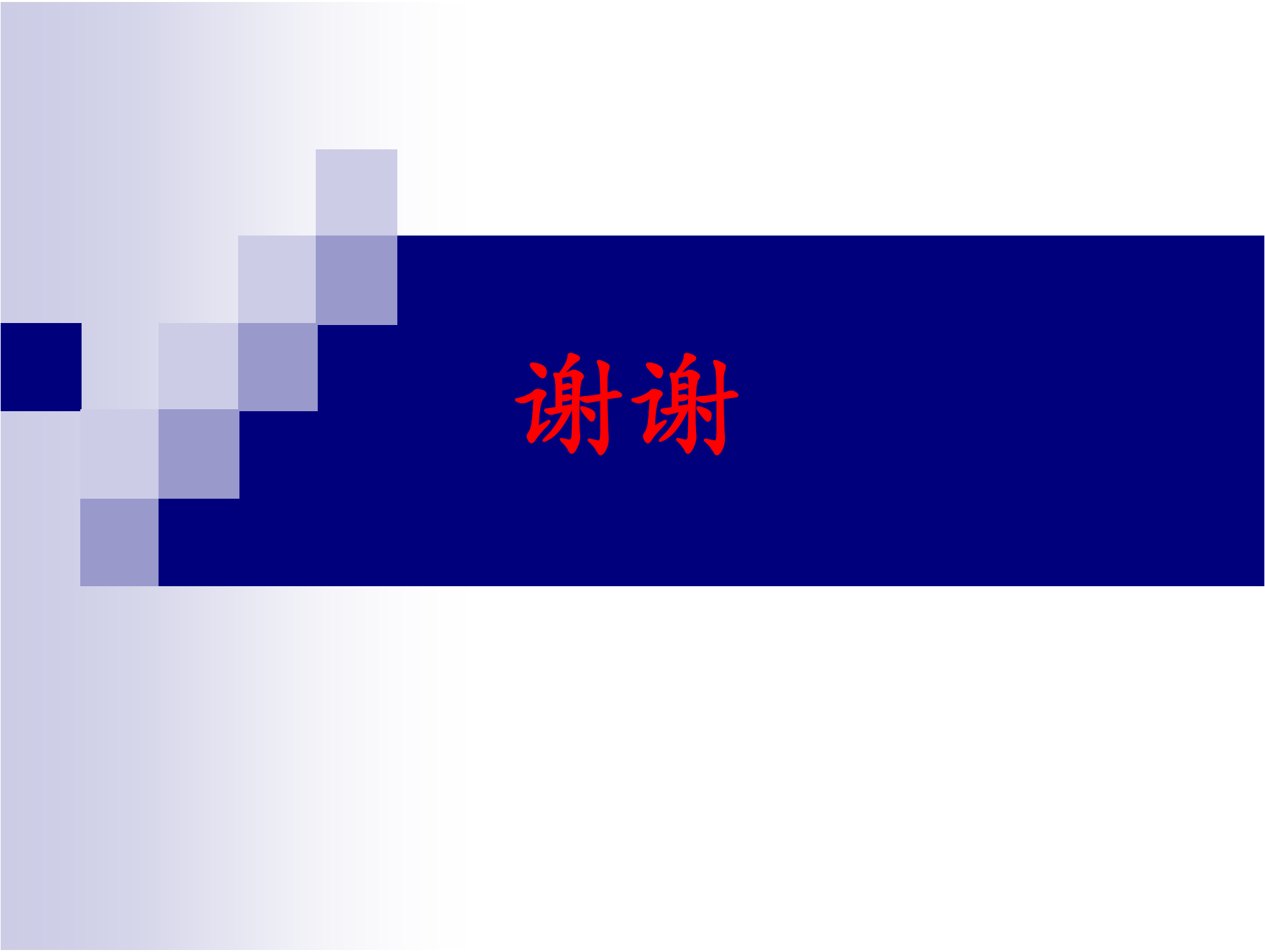
- 当客户端对象需要访问远程主机中的对象时可以使用**远程代理**
- 当需要用一個消耗资源较少的对象来代表一个消耗资源较多的对象，从而降低系统开销、缩短运行时间时可以使用**虚拟代理**
- 当需要为某一个被频繁访问的操作结果提供一个临时存储空间，以供多个客户端共享访问这些结果时可以使用**缓冲代理**



代理模式效果与应用

■ 在以下情况下可以使用代理模式：

- 当需要控制对一个对象的访问，为不同用户提供不同级别的访问权限时可以使用**保护代理**
- 当需要为一个对象的访问（引用）提供一些额外的操作时可以使用**智能引用代理**

The image features a decorative graphic on the left side, consisting of a series of squares in various shades of blue and purple arranged in a staircase pattern. A solid dark blue horizontal bar extends from the right side of this pattern across the middle of the image. The Chinese characters "谢谢" (Thank you) are written in red on this bar.

谢谢