

设计模式之代理模式（Proxy模式）



六尺帐篷 (/u/f8e9b1c246f1)
2017.07.22 16:28 字数 955 阅读 306 评论 0 喜欢 14
(/u/f8e9b1c246f1)

编辑文章 (/writer#/notebooks/5229761/notes/14877868)

- 代理模式的引入
- 代理模式的实例程序
- 代理模式的分析

代理模式的引入

Proxy是代理人的意思，指的是代替别人进行工作的人。当不一定需要本人亲自去做的工作的时候，就可以寻找代理人去完成。

但在代理模式中，往往是相反的，通常是代理人碰到工作，就交给被代理的对象去完成，代理人只完成一些准备工作或者收尾工作。

如果读者了解过spring框架的话，就会知道aop也就是面向切面编程其实运用的就是动态代理模式，这可以让被代理的对象专注于完成自己的本职工作，而代理对象可以进行工作前的日志记录，时间计算，在工作之后进行日志记录，收尾工作等附加的功能，需要正式做工作的时候就交给被代理去做。就像插了两个刀到这个被代理的对象前后。所以形象的叫做面向切面编程。

关于动态代理模式和静态代理模式，感兴趣的读者可以参考笔者的另一篇博文：
Java动态代理与静态代理<http://www.jianshu.com/p/b5e340ec9551>
(<http://www.jianshu.com/p/b5e340ec9551>)

代理模式的实例程序

我们会实现一个打印机，向屏幕打印一串字符串，然后交给代理对象去完成这个功能。

首先看一下类图：

image.png

Printer类：



```
package Proxy;

public class Printer implements Printable {
    private String name;
    public Printer() {
        heavyJob("正在生成Printer的实例");
    }
    public Printer(String name) {           // 构造函数
        this.name = name;
        heavyJob("正在生成Printer的实例(" + name + ")");
    }
    public void setPrinterName(String name) {    // 设置名字
        this.name = name;
    }
    public String getPrinterName() {           // 获取名字
        return name;
    }
    public void print(String string) {          // 显示带打印机名字的文字
        System.out.println("=== " + name + " ===");
        System.out.println(string);
    }
    private void heavyJob(String msg) {         // 重活
        System.out.print(msg);
        for (int i = 0; i < 5; i++) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
            }
            System.out.print(".");
        }
        System.out.println("结束。");
    }
}
```

Printable接口：

```
package Proxy;

public interface Printable {
    public abstract void setPrinterName(String name);

    public abstract String getPrinterName();

    public abstract void print(String string);
}
```

PrinterProxy类,利用反射机制，动态生成被代理的对象，并且延迟初始化到需要调用它的时候再初始化



```
package Proxy;

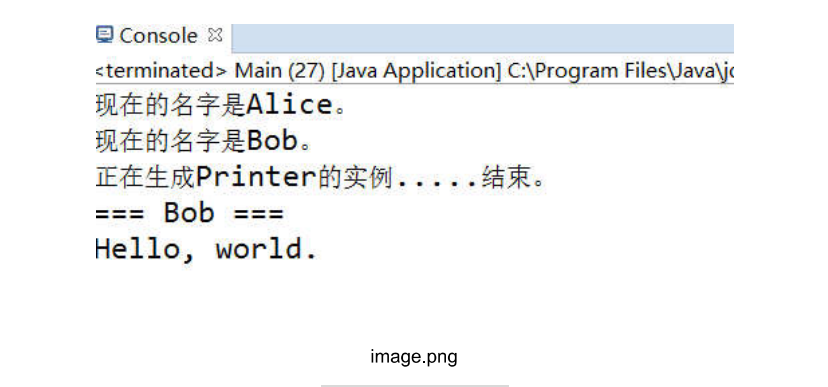
public class PrinterProxy implements Printable {
    private String name;           // 名字
    private Printable real;         // “本人”
    private String className;       // “本人”的类名
    public PrinterProxy(String name, String className) {           // 构造函数
        this.name = name;
        this.className = className;
    }
    public synchronized void setPrinterName(String name) { // 设置名字
        if (real != null) {
            real.setPrinterName(name); // 同时设置“本人”的名字
        }
        this.name = name;
    }
    public String getPrinterName() { // 获取名字
        return name;
    }
    public void print(String string) { // 显示
        realize();
        real.print(string);
    }
    private synchronized void realize() { // 生成“本人”
        if (real == null) {
            try {
                real = (Printable)Class.forName(className).newInstance();
                real.setPrinterName(name);
            } catch (ClassNotFoundException e) {
                System.err.println("没有找到 " + className + " 类。");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

Main类测试：

```
package Proxy;

public class Main {
    public static void main(String[] args) {
        Printable p = new PrinterProxy("Alice", "Proxy.Printer");
        System.out.println("现在的名字是" + p.getPrinterName() + "。");
        p.setPrinterName("Bob");
        System.out.println("现在的名字是" + p.getPrinterName() + "。");
        p.print("Hello, world.");
    }
}
```

运行结果：



代理模式分析

代理模式中的角色：

- Subject（主体）
Subject角色定义了使proxy和realsubject角色之间具有一致性的接口。这个接口提供了一个使用的好处，就是client不必却分它使用的是代理对象还是真实对象。

📁

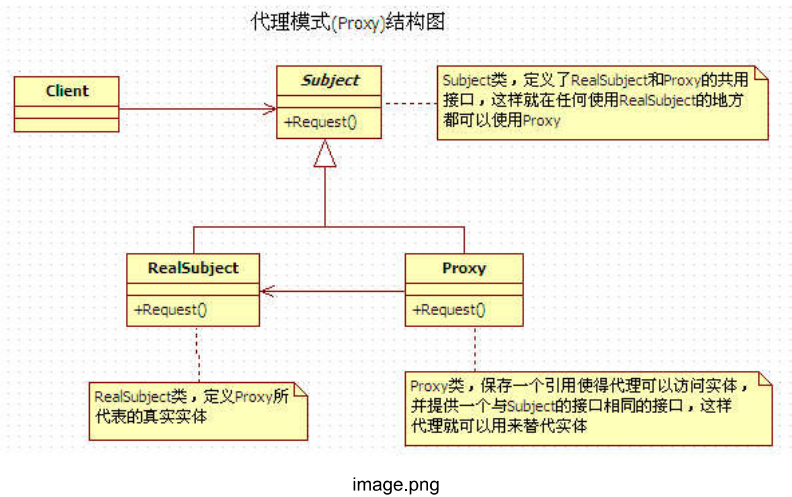
🔖

🔗

对应实例中Printable角色

- Proxy（代理人）
Proxy角色会尽量处理来自Client角色的请求。只有当自己不能处理的时候，就交给工作交给真实对象。代理对象只有在有必要时才会生成真实的对象。
实例中对应的是PrinterProxy对象。
- RealSubject(真实对象)
就是实际完成工作的对象，对应实例中的Printer对象。

代理模式的类图：




- 用代理人来提升速度
关键就在于延迟初始化。
我们可以等到需要使用到真实对象的功能才初始化。这样的好处就是可以提升性能。
从我们的实例中可能看不出这个优势，假设我们有一个大型系统，如果我们都在系统启动的时候，把所有功能初始化，所有实例初始化，那么显然系统的启动将会变得很慢。但如果我们采用代理模式，那么就会在必须的时候，在初始化对象。这样就加快了系统的启动速度。
- 代理和委托
其实我们学习了那么多设计模式，是不是感觉委托简直无处不在。几乎每个设计模式都会用到委托，代理模式也不意外，就是代理了对象委托了真实对象。
因为委托可以是对象之间发生联系，互相调用。所以委托在很多设计模式中都存在。



六尺帐篷 (/u/f8e9b1c246f1)

写了 245418 字，被 15240 人关注，获得了 1429 个喜欢 (/u/f8e9b1c246f1)

喜欢 14





更多分享

被以下专题收入，发现更多相似内容

投稿管理

+ 收入我的专题


 Android知识 (/c/3fde3b545a35?utm_source=desktop&utm_medium=notes-included-collection)


 Android... (/c/5139d555c94d?utm_source=desktop&utm_medium=notes-included-collection)

 Android开发 (/c/d1591c322c89?utm_source=desktop&utm_medium=notes-included-collection)

 程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)

 首页投稿 (/c/bDHhpK?utm_source=desktop&utm_medium=notes-included-collection)

 程序员首页投稿 (/c/89995286335f?utm_source=desktop&utm_medium=notes-included-collection)

 iOS Dev... (/c/3233d1a249ca?utm_source=desktop&utm_medium=notes-included-collection)

展开更多





