原型模式 肉麻情话

原型模式应用场景举例：

GG 和 MM 经常在 QQ 上聊天，但是 GG 打字的速度慢如蜗牛爬行，每次 MM 在瞬间完成恢复或者问候是，GG 都会很紧张的去尽力快速打字，尽管如此，还是让 MM 有些不高心，MM 说回复信息这么慢，显然是用心不专，不在乎她。哎，GG 也是百口难辩啊，不过也确实是没有办法。

有一天，GG 想自己的密友 K 倾诉了自己的苦衷。K 顿生大笑。说道：“傻瓜，你怎么不去网上收集一些肉麻的情话以及一些你们经常说话会涉及到主题，把这些东西拷贝下来保存在自己的电脑或者 U 盘里面，这样一来如果下次在聊天就可以借用过来了！”，“K 就是 K，我怎么没有想到呢~妙极~妙极^_^”，“不过不要太高兴，这些东西是要适当修改的，要不然你把名字都搞错的话，就等着你的 MM 把你踹死吧 O(∩_∩)O 哈哈~”K 补充道，“嗯，说的对，谢谢 K 哥解决了我的心腹之患啊”GG 乐不可支的说道。

这是 MM 由在网上和 GG 聊天，GG 专门复制那些实现准备好的肉麻情话经过稍加修改后发给 MM，MM 都快美死了…

原型模式解释：

原型模式（Prototype Pattern）是一种对象创建型模式，它采取复制原型对象的方法来创建对象的实例。使用 Prototype 模式创建的实例，具有与原型一样的初始化数据

英文定义为：Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

原型模式的 UML 图：

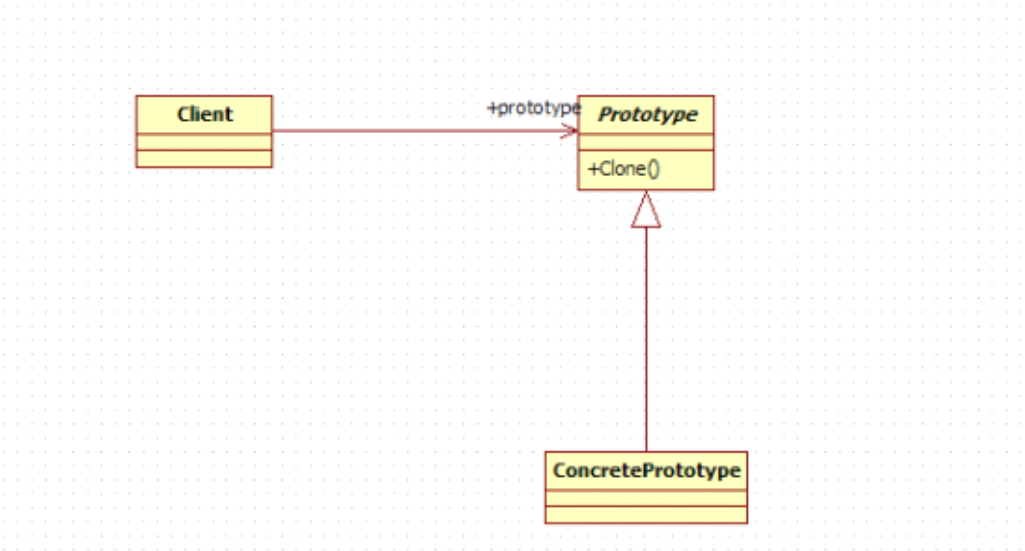
原型模式涉及以下的角色：

客户端（Client）角色: 客户端提出创建对象的请求。

抽象原型（Prototype）角色：通常由一个 Java 接口或者 Java 抽象类来实现。从而为具体原型设立好规范。

具体原型（Concrete Prototype）角色：被复制的具体对象，此具体角色实现了抽象原型角色所要求实现的方法。

原型模式的 UML 图如下所示：



原型模式深入分析：

原型模式的工作原理是:通过将一个原型对象传给那个要发动创建的对象，这个要发动创建的对象通过请求原型对象拷贝它们自己来实施创建。

Java 在语言级别是直接支持原型模式的。我们知道，在 java.lang.Object 是一切类和接口的父类，而 java.lang.Object 正好提供了一个 clone（）方法来支持原型模式。当然，一个对象如果想具有被复制的能力，还必须声明自己实现了 Cloneable 接口，如果没有声明，就会在对象被复制的时候抛出 CloneNotSupportedException.

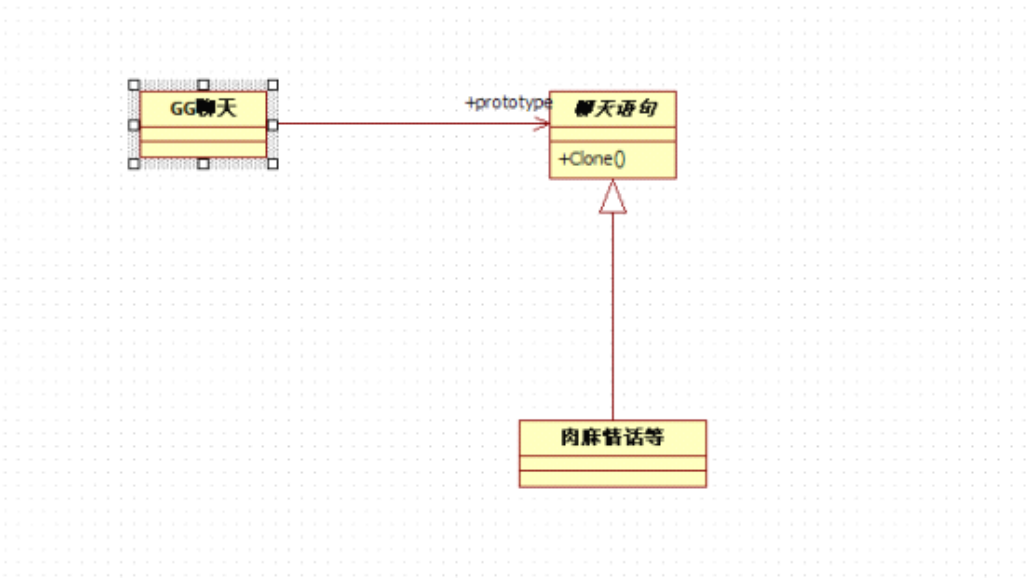
在 java.lang.Object 中提供了一个 protected Object clone()方法来支持对象的克隆，子类可以采用默认的方式进行所有字段的复制，也可以在子类中覆盖 clone()方便，根据实际需要定制自己的复制行为。

复制浅复制和深复制之分，浅复制是对基本数据类型和 String 类型而言的，深复制是对其他引用类型而言的。对于深复制，每一个应用也需要声明 Cloneable 接口。

原型模式使用场景分析及代码实现：

在上面的使用场景中，因为 GG 打字太慢经常被女朋友怪罪，所以有了拷贝网上肉麻情话的和主要聊天话题内容的办法。这样，以后 GG 每次和 MM 聊天的时候只需要把原话拷贝出来，加以适当修改就行，省时省力，而且效果绝佳^_^，这就是设计模式的原型模式的使用的好处 O(∩_∩)O~

UML 模型图如下所示：



建立一个肉麻情话类，类中有非常详细的注释，这里就不在解释了：

```
package com.diermeng.designPattern.Prototype.impl;
import java.util.ArrayList;
import java.util.List;
/*
 * 肉麻情话类
 */
public class SweetWord implements Cloneable{
    //肉麻情话句子
    private String content;
    //肉麻情话句子集合
    private List<String> contents;

    /*
     * 获取肉麻情话集合
     */
    public List<String> getContents() {
        return contents;
    }
    /*
     * 设置肉麻情话集合
     */
    public void setContents(List<String> contents) {
        this.contents = contents;
    }


    /*
```

```
     * 获取肉麻情话
     */
    public String getContent() {
        return content;
    }
    /*
     * 设置肉麻情话
     */
    public void setContent(String content) {
        this.content = content;
    }

    /*
     * 肉麻情话覆盖了Object类的clone()方法，因为这里有List引用进行深度复制
     * @see java.lang.Object#clone()
     */
    public SweetWord clone() {
        try {
            //新建一个肉麻情话对象，同时复制基本的属性
            SweetWord sweetWord  = (SweetWord)super.clone();
            //新建一个肉麻情话集合
            List<String> newContents = new ArrayList<String>();
            //把原对象的肉麻情话集合中的肉麻情话集合通过forEach循环加入新建的
newContents中
            for(String friend : this.getContents()) {
                newContents.add(friend);
            }
            //把新的肉麻情话集合设置进新的对象
            sweetWord.setContents(newContents);
            //返回新的的肉麻情话对象
            return  sweetWord;
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
            return null;
        }
    }

}
```

最后我们建立测试客户端：

```
package com.diermeng.designPattern.Prototype.client;
```

```
import java.util.ArrayList;
import java.util.List;

import com.diermeng.designPattern.Prototype.impl.SweetWord;

/*
 * 肉麻情话测试客户端
 */
public class PrototypeClient {
    public static void main(String[] args) {

        //新建一个肉麻情话对象并设置相应的属性
        SweetWord content1 = new SweetWord();
        List<String> contents = new ArrayList<String>();
        contents.add("宝贝儿，我爱你");
        contents.add("你是我的唯一");

        content1.setContents(contents);
        //复制content1
        SweetWord content2 = content1.clone();
        //分别输入两个对象的内容
        System.out.println(content1.getContents());
        System.out.println(content2.getContents());

        //在原来的肉麻情话对象中加入新的内容并把新的内容设置进去
        contents.add("你是我真命天女");
        content1.setContents(contents);

        //分别输出新的修改后的两个肉麻情话对象
        System.out.println(content1.getContents());
        System.out.println(content2.getContents());
    }
}
```

输出的结果如下：

```
[宝贝儿，我爱你，你是我的唯一]
[宝贝儿，我爱你，你是我的唯一]
[宝贝儿，我爱你，你是我的唯一，你是我真命天女]
[宝贝儿，我爱你，你是我的唯一]
```

原型模式的优缺点分析：

优点：

1.允许动态地增加或减少产品类。由于创建产品类实例的方法是产品类内部具有的，因此增加新产品对整个结构没有影响。

2.提供简化的创建结构。

3.具有给一个应用软件动态加载新功能的能力。

4.产品类不需要非得有任何事先确定的等级结构，因为原型模式适用于任何的等级结构。

缺点：

每一个类都必须配备一个克隆方法，这对于全新的类来说不是很难，而对已有的类来说实现 clone()方法不一定很容易，而且在进行比较深层次的复制的时候也需要编写一定工作量的代码

原型模式的实际应用简介：

原型对象一般在适用于一下场景：

在创建对象的时候，我们不仅希望被创建的对象继承其类的基本机构，而且还希望继承原型对象的数据。

希望对目标对象的修改不影响既有的原型对象（深度克隆的时候可以完全互不影响）。

隐藏克隆操作的细节。很多时候，对对象本身的克隆需要涉及到类本身的数据细节。

温馨提示：

因为使用原型模式的时候每个类都要具备克隆方法。如果在类的设计之初没有很好的规划，等使用很久了才想到克隆，就可能非常的麻烦，尤其是在设计到深层次复制的时候，因为此时牵扯到很多因素，而且工作量非常大。

在给女朋友复制肉麻情话的之前必须充分检查，做适当的修改，别搞的发过去的情话中有参见某某具体网址的情况出现，否则的话，你就死定了 O(∩_∩)O 哈！