## 一句话概括：

**创建重复的对象，同时又能保证性能。**

## 补充介绍：

原型模式（Prototype Pattern）实现了一个原型接口，该接口用于创建当前对象的克隆。当直接创建对象代价比较大时则采用这种模式。

例如，一个对象需要在一个高代价的数据库操作之后被创建，我们则可以缓存该对象，在下次调用的时候直接返回它的克隆，在需要的时候来更新数据库，以减少对数据库的调用。

利用已有的一个原型对象，快速地生成和原型对象一样的实例。

## 参与角色:

1）原型对象的基类

2）原型对象实现类

3）缓存原型对象的类

## 优点：

1）性能提高；

2）逃避构造函数的约束

## 缺点：

1）配备克隆方法需要对类的功能进行通盘考虑，这对于全新的类不是很难，但对于已有的类不一定很容易，特别当一个类引用不支持串行化的间接对象，或者引用含有循环结构的时候；

2）必须实现 Cloneable 接口

## 使用案例或场景：

**使用场景：**

资源优化场景。

对象构建需要消耗很多资源，包括数据库资源硬件资源等。

性能和安全都有要求。

一个对象多个修改者。

一个对象需要供给多个对象调用，而各个调用者都有可能需要修改其值时，可以考虑使用原型模式拷贝多个对象，以供给多个调用者使用。

在实际项目中，原型模式很少单独出现，通常是和工厂方法模式一起出现，通过clone方法创建一个对象，然后通过工厂方法提供给调用者，原型模式已经与java融为一体，可以随时通过clone方法来使用。

**案例：**

Java里面的clone方法；

Spring 里面可以设置bean的类型为prototype，这样可以创建多个一样的bean。

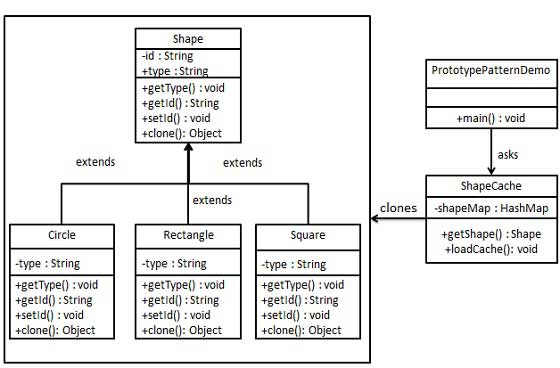
## 示例程序

需要源码的朋友可以前往github下载：

<https://github.com/aharddreamer/chendong/tree/master/design-patterns/demo-code/design-patterns>

**程序简介**：

我们将创建一个抽象类 *Shape* 和扩展了 *Shape* 类的实体类。下一步是定义类 *ShapeCache*，该类把 shape 对象存储在一个 *Hashtable* 中，并在请求的时候返回它们的克隆。



类清单：

Shape 原型对象的基类

ShapeCache 缓存Shape对象的类

Circle 原型对象的实现类1

Rectangle 原型对象的实现类2

Square 原型对象的实现类3

PrototypePatternTest 测试类

**代码：**

**public abstract class** Shape **implements** Cloneable {  
 **private** String **id**;  
 **protected** String **type**;  
  
 **abstract void** draw();  
  
 **public** String getId() {  
 **return id**;  
 }  
  
 **public void** setId(String id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getType() {  
 **return type**;  
 }  
  
 **public void** setType(String type) {  
 **this**.**type** = type;  
 }  
  
 **public** Object clone() {  
 Object clone = **null**;  
 **try** {  
 clone = **super**.clone();  
 } **catch** (CloneNotSupportedException e) {  
 e.printStackTrace();  
 }  
 **return** clone;  
 }  
}

**public class** Circle **extends** Shape {  
 **public** Circle() {  
 **type** = **"Circle"**;  
 }  
 @Override  
 **void** draw() {  
 System.***out***.println(**"Circle draw..."**);  
 }  
}

**public class** Rectangle **extends** Shape {  
 **public** Rectangle() {  
 **type** = **"Rectangle"**;  
 }  
 @Override  
 **void** draw() {  
 System.***out***.println(**"Rectangle draw..."**);  
 }  
}

**public class** Square **extends** Shape {  
 **public** Square() {  
 **type** = **"Square"**;  
 }  
 @Override  
 **void** draw() {  
 System.***out***.println(**"Square draw..."**);  
 }  
}

**public class** ShapeCache {  
 **private static** Hashtable<String, Shape> *shapeMap* = **new** Hashtable<>();  
  
 **public static** Shape getShape(String shapeId) {  
 Shape shape = *shapeMap*.get(shapeId);  
 **return** (Shape) shape.clone();  
 }  
 *// 对每种形状都运行数据库查询，并创建该形状  
 // shapeMap.put(shapeKey, shape);  
 // 例如，我们要添加三种形状* **public static void** loadCache() {  
 Circle circle = **new** Circle();  
 circle.setId(**"1"**);  
 *shapeMap*.put(circle.getId(),circle);  
  
 Square square = **new** Square();  
 square.setId(**"2"**);  
 *shapeMap*.put(square.getId(),square);  
  
 Rectangle rectangle = **new** Rectangle();  
 rectangle.setId(**"3"**);  
 *shapeMap*.put(rectangle.getId(),rectangle);  
 }  
}

**public class** PrototypePatternTest {  
 **public static void** main(String[] args) {  
 ShapeCache.*loadCache*();  
 Shape cloneShape1 = ShapeCache.*getShape*(**"1"**);  
 System.***out***.println(**"Shape: "** + cloneShape1);  
 cloneShape1.draw();  
  
 Shape cloneShape2 = ShapeCache.*getShape*(**"2"**);  
 System.***out***.println(**"Shape: "** + cloneShape2);  
 cloneShape2.draw();  
  
 Shape cloneShape3 = ShapeCache.*getShape*(**"3"**);  
 System.***out***.println(**"Shape: "** + cloneShape3);  
 cloneShape3.draw();  
 }  
}

运行结果：

Shape: org.cd.designpatterns.prototype.Circle@279f2327

Circle draw...

Shape: org.cd.designpatterns.prototype.Square@2ff4acd0

Square draw...

Shape: org.cd.designpatterns.prototype.Rectangle@54bedef2

Rectangle draw...

参考：

《建造者模式》菜鸟教程网站

《图解设计模式》【日】结城浩著