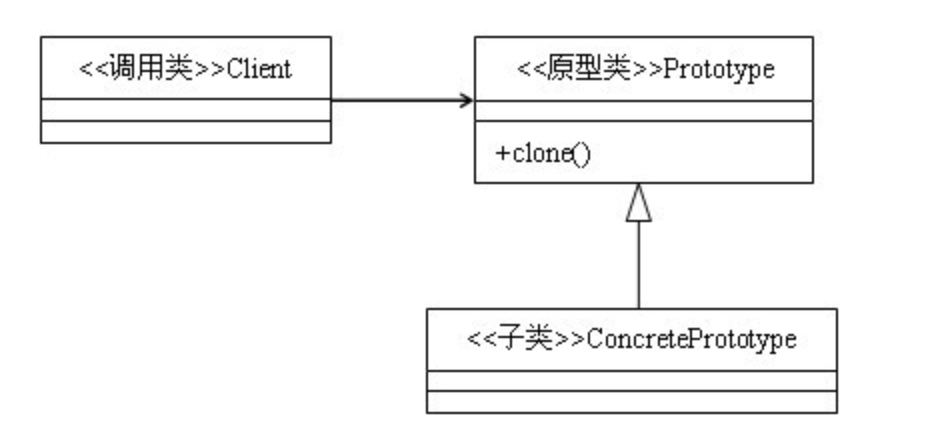
1. 原型模式：用原型实例指定创建对象的种类，并通过拷贝这些原型创建新的对象。
2. 类图



* 1. Prototype：原型类，声明一个克隆自己的接口。
  2. ConcretePrototype：具体的原型类，实现一个克隆自己的操作
  3. Client: 让一个原型对象克隆自己，从而创建一个新的对象(属性一样)

1. 原型模式主要用于对象的复制，它的核心是就是类图中的原型类Prototype。Prototype类需要具备以下两个条件
   1. 实现Cloneable接口。在java语言有一个Cloneable接口，它的作用只有一个，就是在运行时通知虚拟机可以安全地在实现了此接口的类上使用clone方法。在java虚拟机中，只有实现了这个接口的类才可以被拷贝，否则在运行时会抛出CloneNotSupportedException异常
   2. 重写Object类中的clone方法。Java中，所有类的父类都是Object类，Object类中有一个clone方法，作用是返回对象的一个拷贝，但是其作用域protected类型的，一般的类无法调用，因此，Prototype类需要将clone方法的作用域修改为public类型
2. **原型模式的优点及适用场景**
   1. 使用原型模式创建对象比直接new一个对象在性能上要好的多，因为Object类的clone方法是一个本地方法，它直接操作内存中的二进制流，特别是复制大对象时，性能的差别非常明显。
   2. 使用原型模式的另一个好处是简化对象的创建，使得创建对象就像我们在编辑文档时的复制粘贴一样简单。

因为以上优点，所以在需要重复地创建相似对象时可以考虑使用原型模式。比如需要在一个循环体内创建对象，假如对象创建过程比较复杂或者循环次数很多的话，使用原型模式不但可以简化创建过程，而且可以使系统的整体性能提高很多

1. 创建新的对象比较复杂时，可以利用原型模式简化对象的创建过程，同时也能够提高效率
2. 不用重新初始化对象，而是动态地获得对象运行时的状态
3. 如果原始对象发生变化(增加或者减少属性)，其它克隆对象的也会发生相应的变化，无需修改代码
4. 在实现深克隆的时候可能需要比较复杂的代码
5. 缺点:需要为每一个类配备一个克隆方法，这对全新的类来说不是很难，但对已有的类进行改造时，需要修改其源代码，违背了 ocp 原则
6. **原型模式的注意事项**
   1. 使用原型模式复制对象不会调用类的构造方法。因为对象的复制是通过调用Object类的clone方法来完成的，它直接在内存中复制数据，因此不会调用到类的构造方法。不但构造方法中的代码不会执行，甚至连访问权限都对原型模式无效。还记得单例模式吗？单例模式中，只要将构造方法的访问权限设置为private型，就可以实现单例。但是clone方法直接无视构造方法的权限，所以，单例模式与原型模式是冲突的，在使用时要特别注意。
   2. **深拷贝与浅拷贝**。Object类的clone方法只会拷贝对象中的基本的数据类型，对于数组、容器对象、引用对象等都不会拷贝，这就是**浅拷贝**。如果要实现深拷贝，必须将原型模式中的数组、容器对象、引用对象等另行拷贝。由于ArrayList不是基本类型，所以成员变量list，不会被拷贝，需要我们自己实现深拷贝，幸运的是java提供的大部分的容器类都实现了Cloneable接口。所以实现深拷贝并不是特别困难(深拷贝与浅拷贝问题中，会发生浅拷贝的有java中的8中基本类型以及他们的封装类型，另外还有String类型。其余的都是深拷贝)

|  |
| --- |
| public class Prototype implements Cloneable {  private ArrayList list = new ArrayList();  // 方式 1 使用clone 方法  public Prototype clone() {  Prototype prototype = null;  try{  // 这里完成对基本数据类型(属性)和 String 的克隆  prototype = (Prototype)super.clone();  // 对引用类型的属性，进行单独处理  prototype.list = (ArrayList) this.list.clone();  }catch(CloneNotSupportedException e){  e.printStackTrace();  }  return prototype;  }  // 方式2通过对象的序列化实现 (推荐)  public Prototype clone() {  ByteArrayOutputStream bos = null;  ObjectOutputStream oos = null;  ByteArrayInputStream bis = null;  ObjectInputStream ois = null;  //序列化  bos = new ByteArrayOutputStream();  oos = new ObjectOutputStream(bos);  oos.writeObject(this); //当前这个对象以对象流的方式输出  // 反序列化  bis = new ByteArrayInputStream(bos.toByteArray());  ois = new ObjectInputStream(bis);  Prototype copyObj = (Prototype)ois.readObject();  return copyObj;  }  } |

1. Spring中原型bean的创建，就是原型模式的使用