第2章 原型模式

提出问题

> 问题描述：假设现在需要针对多个职位撰写求职简历，简历中求职者的基本信息（例如姓名、电话、邮箱等）保持一致，一些具体信息（例如求职意向、技能特长、自我评价等）需要针对不同岗位和企业进行调整。设计程序模拟创建简历的过程，思考如何快速创建多份简历？简历格式如下：

```
<html><body>
<h1>求职简历</h1>
<h3>姓名：小红</h3>
<h3>电话：15627272727</h3>
<h3>邮箱：xiaohong@swu.edu.cn</h3>
<h3>年龄：22</h3>
<h3>地址：天生路2号</h3>
<h3>工作经验：0</h3>
<h2>求职意向</h2>
<p>职位：开发工程师，期望月薪：20K</p>
<h2>教育背景</h2>
<p>2016-2020，西南大学，计算机专业</p>
<h2>技能特长</h2>
<p>Java语言、C#、C++、Python、主流框架</p>
<h2>自我评价</h2>
<p>酷爱编程，逻辑思维能力强</p>
</html></body>
```

根据需求分析，简历包含10个内容，其中6个简单内容，4个复杂内容，具体实现代码如下：

```
public class Resume {
    private String name;    //姓名
    private String phoneNumber; //电话
    private String email;   //邮箱
    private int age;    //年龄
    private String address; //地址
    private int workingAge; //工作经验
    private Item intention; //求职意向
    private Item education; //教育背景
    private Item skills;    //技能特长
    private Item selfEvaluation;    //自我评价

    public StringBuilder create() {
        StringBuilder data = new StringBuilder();
        data.append("<html><body>\n");
        data.append("<h1>求职简历</h1>\n");
        data.append("<h3>姓名: " + name + "</h3>\n");
        data.append("<h3>电话: " + phoneNumber + "</h3>\n");
        data.append("<h3>邮箱: " + email + "</h3>\n");
        data.append("<h3>年龄: " + age + "</h3>\n");
        data.append("<h3>地址: " + address + "</h3>\n");
        data.append("<h3>工作经验: " + workingAge + "</h3>\n");
        data.append("<h2>" + intention.getTitle() + "</h2>\n");
        data.append("<p>" + intention.getTxt() + "</p>\n");
        data.append("<h2>" + education.getTitle() + "</h2>\n");
        data.append("<p>" + education.getTxt() + "</p>\n");
        data.append("<h2>" + skills.getTitle() + "</h2>\n");
        data.append("<p>" + skills.getTxt() + "</p>\n");
        data.append("<h2>" + selfEvaluation.getTitle() + "</h2>\n");
        data.append("<p>" + selfEvaluation.getTxt() + "</p>\n");

        data.append("</html></body>\n");
        return data;
    }
}
// Item类用于封装一个较为复杂的简历要素
public class Item {
    private String title;
    private String txt;
    public Item(String title, String txt) {
        this.title = title;
        this.txt = txt;
    }
}
```

客户端程序如下，创建一个简历对象，设置每一项的信息，最后打印简历。假设需要设计一份针

对其他职位的简历，就需要将整个流程重复一遍。

```
public class Test {
    public static void main(String[] args) {
        Resume re = new Resume();
        re.setName("小红");
        re.setAge(22);
        re.setAddress("天生路2号");
        re.setEmail("xiaohong@swu.edu.cn");
        re.setPhoneNumber("15627272727");
        re.setWorkingAge(0);
        re.setIntention(new Item("求职意向", "职位：开发工程师，期望月薪：20K"));
        re.setEducation(new Item("教育背景", "2016-2020，西南大学，计算机专业"));
        re.setSkills(new Item("技能特长", "Java语言、C#、C++、Python、主流框架"));
        re.setSelfEvaluation(new Item("自我评价", "酷爱编程，逻辑思维能力强"));
        System.out.println(re.create().toString());
    }
}
```

模式名称

原型模式：Prototype

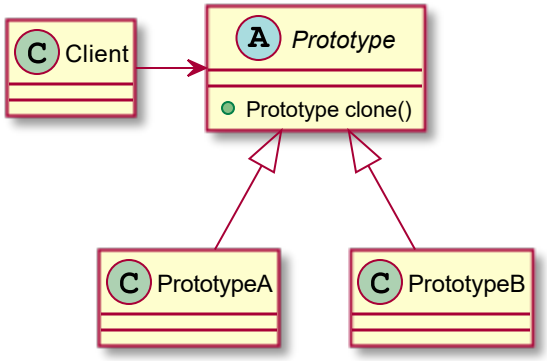
设计意图

用原型实例指定被创建对象的种类，并且通过复制原型的方式创建新对象。

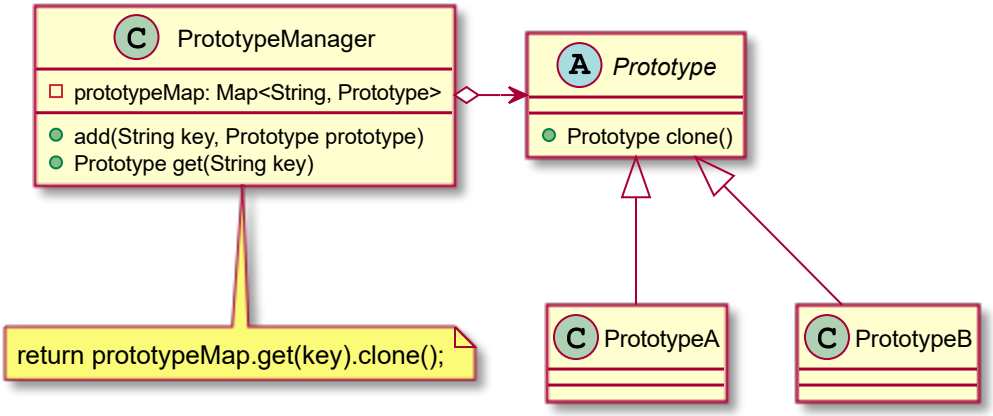
> Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

设计结构

原型模式一般形式：



原型管理器：



实现方法

Java中，Object类提供了`clone()`方法实现对象复制，采用了本地代码实现。如下所示：

```
protected native Object clone() throws CloneNotSupportedException;
```

使用该方法需要注意三点：

- 由于`clone()`方法的可见性为`protected`，需要在自定义类中重写`clone()`方法，并将可见性修改为`public`。
- `clone()`方法声明了`CloneNotSupportedException`异常，当被克隆对象对应的类没有实现`Cloneable`接口时会抛出该异常。
- `clone()`方法只实现对象自身属性的复制，对象所引用的其他对象不会复制，也就是所谓的"浅复制"，而应用中往往需要实现"深复制"，这需要额外的操作。

浅复制实现方式如下：

```
public class Prototype implements Cloneable{
    private int a;
    private int[] arr;

    public static void main(String[] args) {
        Prototype pro = new Prototype();
        pro.setA(10);
        pro.setArr(new int[]{1, 2});

        Prototype clonePro = null;
        try {
            clonePro = (Prototype) pro.clone();
            clonePro.setA(20);
            clonePro.getArr()[1] = 3;
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        System.out.println(pro.getArr()[1]); // 打印3
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        Prototype prot = (Prototype) super.clone();
        return prot;
    }

    // getter和setter方法略
}
```

深复制实现方式如下，关键在于处理引用数据类型

```
public class Prototype implements Cloneable{
    private int a;
    private int[] arr;

    public static void main(String[] args) {
        // 这里代码与上面相同
        System.out.println(pro.getArr()[1]); // 打印2
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        Prototype prot = (Prototype) super.clone();
        prot.setArr(this.getArr().clone()); //引用的数据复制
        return prot;
    }

    // getter和setter方法略
}
```

解决问题

掌握了`Object.clone()`方法的使用后，采用原型模式对之前程序进行修改。
`Item`类主要修改如下：

```
public class Item implements Cloneable{
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

`Resume`类主要修改如下：

```
public class Resume implements Cloneable{
    public Object clone() throws CloneNotSupportedException {
        Resume re = (Resume) super.clone();
        re.setIntention((Item) this.getIntention().clone());
        re.setEducation((Item) this.getEducation().clone());
        re.setSkills((Item) this.getSkills().clone());
        re.setSelfEvaluation((Item) this.getSelfEvaluation().clone());
        return re;
    }
}
```

客户端测试代码如下：

```
public class Test {
    public static void main(String[] args) {
        Resume prototype = new Resume();
        prototype.setName("小红");
        prototype.setAge(22);
        prototype.setAddress("天生路2号");
        prototype.setEmail("xiaohong@swu.edu.cn");
        prototype.setPhoneNumber("15627272727");
        prototype.setWorkingAge(0);
        prototype.setIntention(new Item("求职意向", ""));
        prototype.setEducation(new Item("教育背景", "2016-2020，西南大学，计算机专业"));
        prototype.setSkills(new Item("技能特长", ""));
        prototype.setSelfEvaluation(new Item("自我评价", ""));

        Resume r1 = null;
        Resume r2 = null;
        try {
            r1 = (Resume) prototype.clone();
            r2 = (Resume) prototype.clone();
            r1.getIntention().setTxt("职位：后端工程师，期望月薪：20K");
            r1.getSkills().setTxt("Java语言、C#、C++、Python、主流框架");
            r1.getSelfEvaluation().setTxt("酷爱编程，逻辑思维能力强");

            r2.getIntention().setTxt("职位：前端工程师，期望月薪：20K");
            r2.getSkills().setTxt("Javascript、HTML、CSS、主流前端框架");
            r2.getSelfEvaluation().setTxt("界面设计能力强");

        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        System.out.println(r1.create().toString());
        System.out.println(r2.create().toString());
    }
}
```

测试效果如下：

```
<html><body>
<h1>求职简历</h1>
<h3>姓名：小红</h3>
<h3>电话：15627272727</h3>
<h3>邮箱：xiaohong@swu.edu.cn</h3>
<h3>年龄：22</h3>
<h3>地址：天生路2号</h3>
<h3>工作经验：0</h3>
<h2>求职意向</h2>
<p>职位：后端工程师，期望月薪：20K</p>
<h2>教育背景</h2>
<p>2016-2020，西南大学，计算机专业</p>
<h2>技能特长</h2>
<p>Java语言、C#、C++、Python、主流框架</p>
<h2>自我评价</h2>
<p>酷爱编程，逻辑思维能力强</p>
</html></body>

<html><body>
<h1>求职简历</h1>
<h3>姓名：小红</h3>
<h3>电话：15627272727</h3>
<h3>邮箱：xiaohong@swu.edu.cn</h3>
<h3>年龄：22</h3>
<h3>地址：天生路2号</h3>
<h3>工作经验：0</h3>
<h2>求职意向</h2>
<p>职位：前端工程师，期望月薪：20K</p>
<h2>教育背景</h2>
<p>2016-2020，西南大学，计算机专业</p>
<h2>技能特长</h2>
<p>Javascript、HTML、CSS、主流前端框架</p>
<h2>自我评价</h2>
<p>界面设计能力强</p>
</html></body>
```

效果与适用性

效果：

- 复制实例提高复杂对象创建效率。
- 相对工厂模式能有效降低子类数量。
- 实现系统动态配置。加载时创建实例并注册为原型，通过原型管理器统一管理，根据需求

动态创建对象。

适用性

- 当一个系统应该独立于它的产品创建、构成和表示时；
- 当类的实例化在系统运行时进行指定时；
- 避免像工厂模式那样定义对应产品的工厂类时；
- 当一个类的实例只有几个不同状态的组合时，预先建立所有状态实例作为原型用于克隆比手工实例化该类要更方便。