软件体系结构 作业11

22920212204392 黄勖

1 请举例说明克隆模式的其他应用。

应用：

1. 缓存管理： 在缓存管理中，克隆模式可以用于创建缓存对象的副本。当需要从缓存中获取对象时，可以使用克隆模式克隆缓存中的对象，而无需重新从数据库或其他数据源中获取数据。通过克隆模式，可以提高缓存数据的读取效率，减少对外部资源的依赖。
2. 大型对象创建： 在某些场景下，创建一个大型对象可能需要耗费大量的时间和资源。使用克隆模式，可以通过复制一个现有的大型对象来创建新对象，从而避免重新构建整个对象。这在需要频繁创建相似对象的情况下特别有用，可以大大提高对象的创建效率。
3. 一个对象有多个修改者时：一个对象需要提供给其他对象访问，而且各个调用者可能都需要修改其值时，可以考虑用克隆模式拷贝多个对象供调用者使用。

2 试描述浅克隆和深克隆。

浅克隆：被Clone的对象的所有变量都含有原来对象相同的值，而引用变量还是原来对用的引用。拷贝对象时仅仅拷贝对象本身（包括对象中的基本变量），而不拷贝对象包含的引用指向的对象。

深克隆：被克隆对象的所有变量都含有原来的对象相同的值，引用变量也重新复制了一份。不仅拷贝对象本身，而且拷贝对象包含的引用指向的所有对象。

2.1 浅克隆示例

在Java中实现浅克隆通常涉及实现`Cloneable`接口，并重写`clone()`方法。

```
public class Student implements Cloneable {
    String name;
    String university;
    StudentInfo info;

    public Student(String name, String university, StudentInfo info) {
        this.name = name;
        this.university = university;
        this.info = info;
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    public static void main(String[] args) {
```

```
        StudentInfo info = new StudentInfo("22920212204392");
        Student original = new Student("黄勖", "厦门大学", info);
        Student cloned = null;
        try {
            cloned = (Student) original.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }

        System.out.println("Original: " + original.name + ", " +
original.university + ", " + original.info.id);
        System.out.println("Cloned: " + cloned.name + ", " + cloned.university
+ ", " + cloned.info.id);

        cloned.info.id = "Changed ID";
        System.out.println("After change:");
        System.out.println("Original: " + original.info.id);
        System.out.println("Cloned: " + cloned.info.id);
    }
}

class StudentInfo {
    String id;

    public StudentInfo(String id) {
        this.id = id;
    }
}
```

输出结果:

2.2 深克隆示例

实现深克隆通常需要手动复制所有对象和子对象。

```
import java.io.*;

public class DeepCopy implements Serializable {
    String name;
    String university;
    StudentInfo info;
```

```
    public DeepCopy(String name, String university, StudentInfo info) {
        this.name = name;
        this.university = university;
        this.info = new StudentInfo(info.id);
    }

    public static DeepCopy deepClone(DeepCopy object) {
        try {
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(bos);
            oos.writeObject(object);
            oos.flush();
            ByteArrayInputStream bin = new
ByteArrayInputStream(bos.toByteArray());
            ObjectInputStream ois = new ObjectInputStream(bin);
            return (DeepCopy) ois.readObject();
        } catch (Exception e) {
            return null;
        }
    }

    public static void main(String[] args) {
        StudentInfo info = new StudentInfo("22920212204392");
        DeepCopy original = new DeepCopy("黄勖", "厦门大学", info);
        DeepCopy cloned = deepClone(original);

        System.out.println("Original: " + original.name + ", " +
original.university + ", " + original.info.id);
        System.out.println("Cloned: " + cloned.name + ", " + cloned.university
+ ", " + cloned.info.id);

        cloned.info.id = "Changed ID";
        System.out.println("After change:");
        System.out.println("Original: " + original.info.id);
        System.out.println("Cloned: " + cloned.info.id);
    }
}

class StudentInfo implements Serializable {
    String id;

    public StudentInfo(String id) {
        this.id = id;
    }
}
```

输出结果:

在深克隆示例中，即使修改了克隆对象的信息，原始对象的信息也保持不变。这展示了深克隆与浅克隆的根本区别。