大家好，我是楼仔！

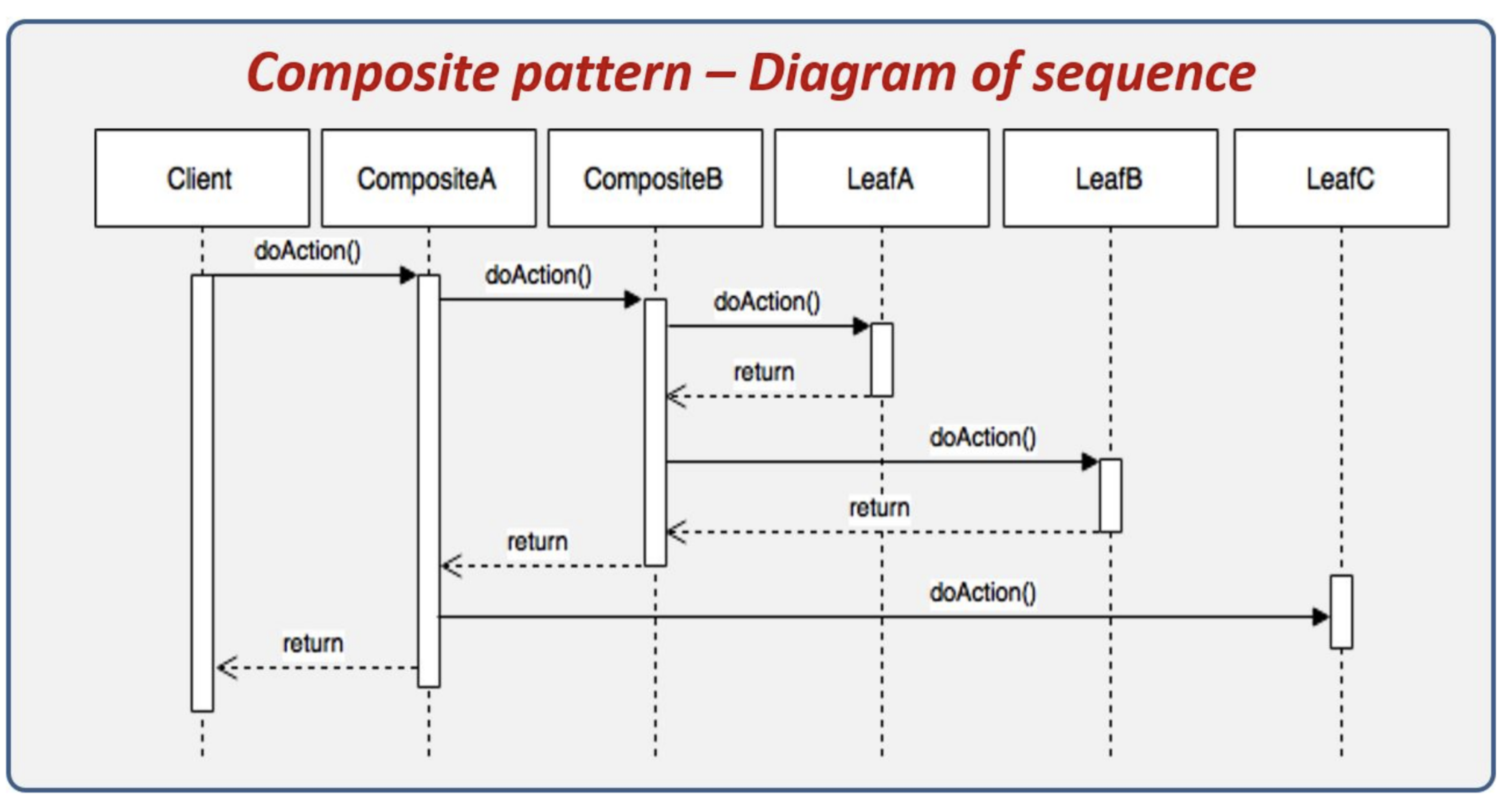
最初接触组合模式，是来源于我们小组同学的一个分享，他当时给我们讲购物车界面的重构，把之前的购物车大单体拆成了树状结构，就用到了组合模式。

我们看一下组合模式的定义：

> 组合模式（Composite Pattern），又叫部分整体模式，是用于把一组相似的对象当作一个单一的对象。组合模式依据树形结构来组合对象，用来表示部分以及整体层次。这种类型的设计模式属于结构型模式，它创建了对象组的树形结构。

组合模式其实比较简单，层次很分明，主要包括一个抽象接口、组合对象节点和叶子节点：

- Component抽象组件：为组合中的所有对象提供一个接口，不管是叶子对象还是组合对象。
- Component组合节点对象：实现了接口的所有操作，并且持有子节点对象。
- Leaf叶子节点对象：叶子节点没有任何子节点，实现了接口中的某些操作。



前面讲的有点啰嗦，理解一个设计模式，最快就是看示例，如果大家不想看文字，也可以直接跳到代码示例部分。

组合模式

对组合模式的理解，其实就是一个“总-分”的关系，直接先看定义的抽象类：

```
public abstract class penguin {
    protected String name;

    public penguin(String name) {
        this.name = name;
    }

    public abstract void beating();

    public void add(penguin p) {
        throw new UnsupportedOperationException();
    }
    public void remove(penguin p) {
        throw new UnsupportedOperationException();
    }
    public penguin getChild(int i) {
        throw new UnsupportedOperationException();
    }
    public List<penguin> getChilds() {
        throw new UnsupportedOperationException();
    }
}
```

这个抽象类其实就是定义了一个公共的行为beating，然后增加了一些方法，这些方法在“Component组合节点对象”都需要实现，但是在“Leaf叶子节点对象”可以不用实现。我这里其实想先丢弃叶子节点，讲一个简洁版的组合模式。下面看“Component组合节点对象”的代码：

```
public class batchPenguin extends penguin {
    private List<penguin> m_penguins = new ArrayList<>();
    public batchPenguin(String name) {
        super(name);
    }
    @Override
    public void beating() {
        System.out.println(this.name + "打豆豆");
        for (penguin p : m_penguins) {
            p.beating();
        }
    }
    @Override
    public void add(penguin p) {
        m_penguins.add(p);
    }
    @Override
    public void remove(penguin p) {
        m_penguins.remove(p);
    }
    @Override
    public penguin getChild(int i) {
        return m_penguins.get(i);
    }
    @Override
    public List<penguin> getChilds() {
        return m_penguins;
    }
}
```

所有的对象，都有一个打豆豆的行为，然后每个对象都有增加、删除、获取子节点的方法，这个其实就是组合模式的核心，最后是使用姿势：

```
public static void main(String[] args) {
    batchPenguin grandFatherPenguin = new batchPenguin("grandFatherPenguin");
    batchPenguin fatherPenguin = new batchPenguin("fatherPenguin");
    batchPenguin motherPenguin = new batchPenguin("motherPenguin");
    batchPenguin childPenguin1 = new batchPenguin("childPenguin1");
    batchPenguin childPenguin2 = new batchPenguin("childPenguin2");
    batchPenguin childPenguin3 = new batchPenguin("childPenguin3");
    batchPenguin childPenguin4 = new batchPenguin("childPenguin4");
    fatherPenguin.add(childPenguin1);
    fatherPenguin.add(childPenguin2);
    motherPenguin.add(childPenguin3);
    motherPenguin.add(childPenguin4);
    grandFatherPenguin.add(fatherPenguin);
    grandFatherPenguin.add(motherPenguin);
    grandFatherPenguin.beating();
}
```

输出结果：

```
grandFatherPenguin打豆豆
fatherPenguin打豆豆
childPenguin1打豆豆
childPenguin2打豆豆
motherPenguin打豆豆
childPenguin3打豆豆
childPenguin4打豆豆
```

我们可以看到组合模式的好处，就是我们不用关系每个对象里面的子成员，只要我们把子对象add()进去后，调用父节点的beating()操作后，会执行子成员，以及子成员包括下面所有子节点的beating()操作，形成一个递归操作。

加上叶子节点

我上面的示例，没有加上叶子节点，其实我在实际的应用场景中，没有使用叶子节点，直接就是这个简版的组合模式，也能达到我想要的效果。不过既然组合模式有叶子节点，我也就加上，仅作为了解使用，下面是叶子节点代码：

```
public class leaf extends penguin{
    public leaf(String name) {
        super(name);
    }
    @Override
    public void beating() {
        System.out.println(name + "打豆豆");
    }
}
```

然后在main中新增leaf节点：

```
leaf leaf1 = new leaf("leaf1");
leaf leaf2 = new leaf("leaf2");
leaf leaf3 = new leaf("leaf3");
leaf leaf4 = new leaf("leaf4");
childPenguin1.add(leaf1);
childPenguin2.add(leaf2);
childPenguin3.add(leaf3);
childPenguin4.add(leaf4);
```

最后输出：

```
grandFatherPenguin打豆豆
fatherPenguin打豆豆
childPenguin1打豆豆
leaf1打豆豆
childPenguin2打豆豆
leaf2打豆豆
motherPenguin打豆豆
childPenguin3打豆豆
leaf3打豆豆
childPenguin4打豆豆
leaf4打豆豆
```

说实话，我没有看到这个叶子节点加进去有啥意义，仅仅是标记一个结尾符么？或者说有其他具体的适用场景？这里我就不深究了。所以还是那句话，我们只需要理解每种设计模式的思想即可，实际的应用场景，完全不用照搬。

实际场景

查了网上的资料，大家都说组合模式在菜单、文件、文件夹的管理上用的非常多，因为他们是树形结构模式，不过实际的场景中，我只接触过购物车重构这块，所以我就简单说一下。

下面是小米商城购物车界面，可以看到里面有很多功能模块，你可以直接采用堆砌的方式实现每一个模块，然后依次调用每个模块具体的执行逻辑：

当然，我们也可以用组合模式，将购物车抽象成下面的树状结构（还有很多模块，仅列举一部分）：

代码我就不贴了，核心实现就是通过组合模式将购物车的对象按照“总-分”关系组合在一起，最后执行购物车的Process()方法，就可以调用所有对象的Process()操作，从而完成每个模块对自身业务的逻辑处理。

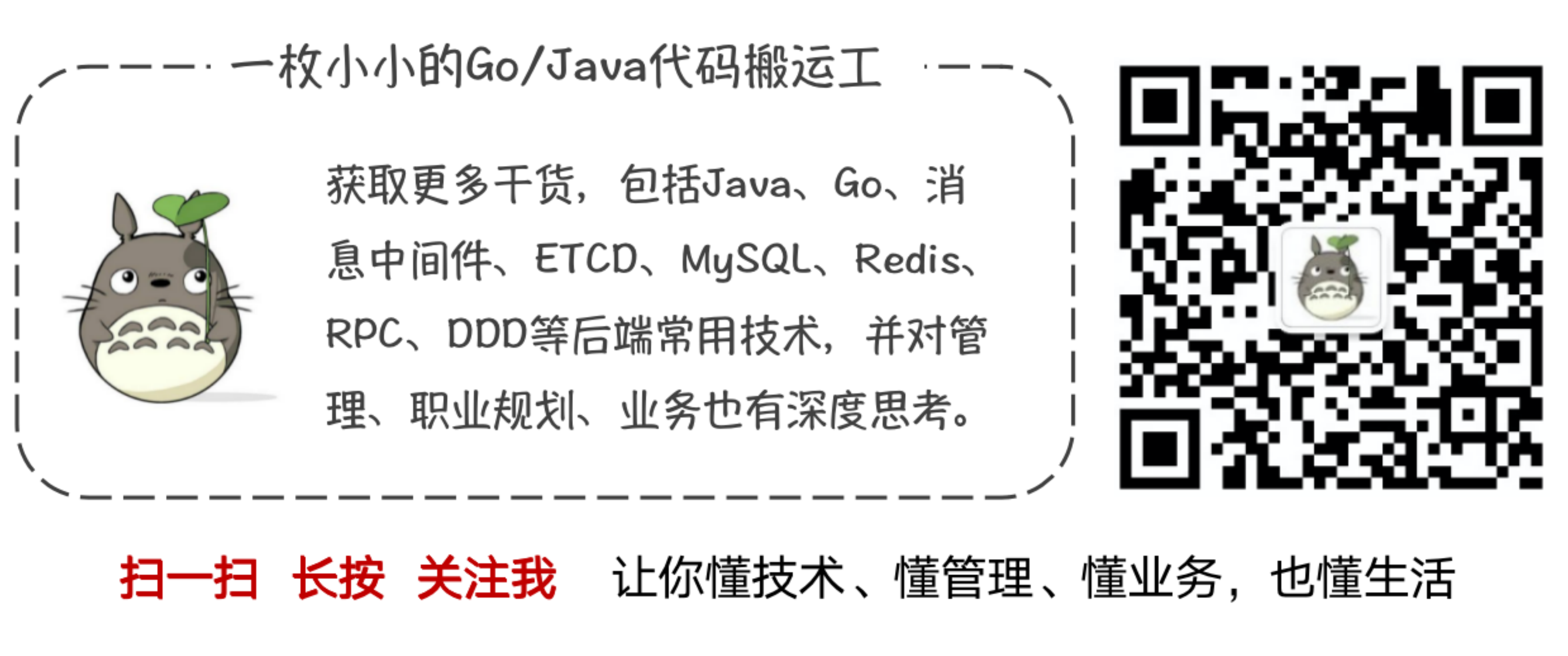
结语

总结一下，后续如果你的代码需要处理成“总-分”关系，或者说是树形结构关系，最后通过一次调用完成所有对象的操作行为，那么就可以选择组合模式。

其实在写组合模式前，我没有找到项目中具体应用的场景，只看到网上的文章，说目录关系可以用组合模式，但是总感觉对这个模式的理解一直停留在表面，当我找到项目中购物车的实际运用场景时，我才感觉对这个模式有了更深入的理解。那大家也可以想想，自己做过的项目中，有哪些场景用到了组合模式呢?

学习交流

可以扫下面二维码，关注「楼仔」公众号。



长按二维码，回复 **「加群」**，欢迎一起学习交流哈~~ 👏🏻👏🏻👏🏻

> 尽信书则不如无书，因个人能力有限，难免有疏漏和错误之处，如发现 bug 或者有更好的建议，欢迎批评指正，不吝感激。