访问者模式

1.访问者模式介绍

访问者模式的基本想法是，软件系统中拥有一个有许多对象构成的、比较稳定的对象结构，这些对象的累都拥有一个accept方法用来接受访问者对象的访问。访问者是一个接口，它拥有一个visit方法，这个方法对方问到的对象结构中不同类型的元素作出不同的处理。在对象结构的一次访问过程中，我们遍历整个对象结构，对每一个元素都实施accept方法，在每一个元素的accept方法中会调用访问者的visit方法，从而使访问者得以处理对象结构的每一个元素，我们可以针对对象结构设计不同的访问者类来完成不同的操作，达到区别对待的效果。

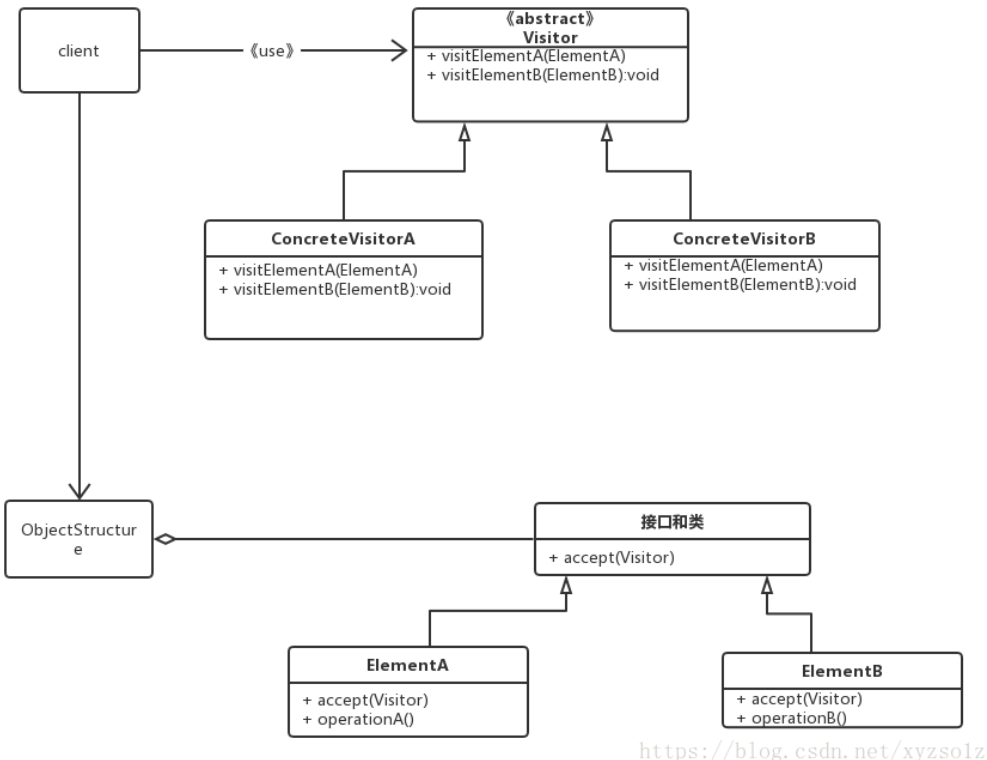
2.访问者模式的定义

封装一些作用于某种数据结构中的各元素的操作，他可以在不改变这个数据结构的前提下定义作用于这些元素的新的操作。

3.访问者模式的使用场景

1. 对象结构比较稳定，但经常需要在此对象结构上定义一些新的操作。
2. 需要对一个对象结构中的对象进行很多不同的并且不相关的操作，而需要避免这些操作“污染”这些对象的类，也不希望在增加新操作时修改这些类。

4.访问者模式的UML类图



角色介绍：

- Visitor:接口或者抽象类，他定义了对每一个元素（Element）访问的行为，他的参数就是可以访问的元素，他的方法个数理论上来讲与元素个数是一样的，因此，访问者模式要求元素的类族要稳定，如果经常添加、移除元素

类，必然会导致频繁地修改Visitor接口，如果出现这种情况，则说明不适合使用访问者模式。

- ConcreteVisitor:具体的访问者，他需要给出对每一个元素类访问时所产生的具体行为。
- Element:元素接口或者抽象类，他定义了一个接受访问者(accept)的方法，其意义是指每一个元素都要可以被访问者访问。
- ElementA、ElementB：具体的元素访问类，他提供接受访问方法的具体实现，而这个具体的实现，通常情况下是使用访问者提供的访问该元素的方法。
- ObjectStructure:定义当中所提到的对象结构，对象结构是一个抽象表述，他内部管理了元素集合，并且可以迭代这些元素访问着访问。

5.访问者模式的简单示例

通过一个简单示例来学习它的典型应用。

在年终时，公司都会给员工进行业绩考核，以此来评定该员工的绩效及年终奖、晋升等，这些评定都是由公司高层来负责，但是，不同领域的管理人员对员工的评定标准是不一样的，为了简单明了地说明问题，我们把员工简单分为工程师和经理，评定员工的分别为CEO和CTO，我们假定CTO只关注工程师的代码量、经理的xinchanping数量，而CEO关注的是工程师的KPI和经理的KPI以及新产品数量，从中可以看出，CEO和CTO对于不同员工的关注点是不一样的，这就需要对不同的员工类型进行不同的处理。访问者模式此时可以派上用场了：

```
package visitorpattern;

import java.util.Random;

//员工基类
public abstract class Staff {
        public String name;
    //员工kpi
        public int kpi;

        public Staff (String aName){
                this.name=aName;
                kpi=new Random().nextInt(10);
        }

        //接受访Visitor的访问者
        public abstract void accept(Visitor visitor);
}
```

Staff类定义了员工的基本信息及一个accept方法，accept方法表示接受访问者的访问，有子类具体实现。下面看看工程师和精力的相关代码：

```
package visitorpattern;

import java.util.Random;
//工程师
public class Engineer extends Staff {

        public Engineer(String aName) {
                super(aName);
        }
```

```
        @Override
                public void accept(Visitor visitor) {
                        visitor.visit(this);
                }

                // 工程师一年
                public int getCodeLines() {
                        return new Random().nextInt(10 * 10000);
                }
}
```

```
package visitorpattern;

import java.util.Random;
//经理类型
public class Manager extends Staff {
        private int products;// 产品数量

        public Manager(String aName) {
                super(aName);
                products=new Random().nextInt(10);
        }



        @Override
        public void accept(Visitor visitor) {
                visitor.visit(this);
        }

        public int getProducts(){
                return products;
        }

}
```

在工程师类中添加了获取代码行数的函数，而在经理类型重则添加了获取新产品数量的函数，他们的职责不同，也正是由于他们的差异性是的访问者模式能够发挥他的作用，Staff、Engineer、Manager 3个类型就是对象结构，这些类型相对稳定，不会发生变化。

然后将这些员工天津爱到一个业务报表类，公司高层可以通过该报表类的showReport函数查看所有员工的业绩：

```
package visitorpattern;

import java.util.LinkedList;
import java.util.List;
//员工业务报表
public class BusinessReport {

        List<Staff> mStaffs=new LinkedList<Staff>();
        public BusinessReport() {
                mStaffs.add(new Manager("王经理"));
                mStaffs.add(new Engineer("工程师-gong1"));
```

```
                mStaffs.add(new Engineer("工程师-gong2"));
            mStaffs.add(new Engineer("工程师-gong3"));
            mStaffs.add(new Engineer("工程师-gong4"));
            mStaffs.add(new Engineer("工程师-gong5"));        }
        /*
         * 为访问者展示报表
         */
        public void showReport(Visitor visitor){
                for (Staff staff : mStaffs) {
                        staff.accept(visitor);
                }
        }
}
```

下面是VIsitor类型的定义，Visitor声明了连个visit函数，分别是对工程师和经理的访问函数：

```
package visitorpattern;

public interface   Visitor {
        //访问工程师类型
        public void visit(Engineer engineer) ;
        //访问经理类型
        public void visit(Manager leader) ;
}
```

首先定义一个Visitor接口，该接口有两个Visit函数，参数分别为Engineer、Manager，也就是说对于Engineer、Manager的访问会调用两个不同的方法，以此达成区别对待、差异化处理：

```
package visitorpattern;

//CEO访问者，只关注业绩
public class CEOVisitor implements Visitor {

        @Override
        public void visit(Engineer engineer) {
                System.out.println("工程师:  " + engineer.name + ",KPI: " + engineer.kpi);
        }

        @Override
        public void visit(Manager leader) {
                System.out.println("经理:  " + leader.name + ", KPI: " + leader.kpi
                                + ", 产品数量: " + leader.getProducts());
        }

}
```

```
package visitorpattern;

//CTO更关注员工在技术层面的贡献
public class CTOVisitor implements Visitor {
```

```
        @Override
                        public void visit(Engineer engineer) {
                System.out.println("工程师： " + engineer.name + ",代码行数： "
                                + engineer.getCodeLines());
        }

        @Override
        public void visit(Manager leader) {
                System.out.println("经理： " + leader.name + ",产品数量："
                                + leader.getProducts());
        }

}
```

下面是客户端代码：

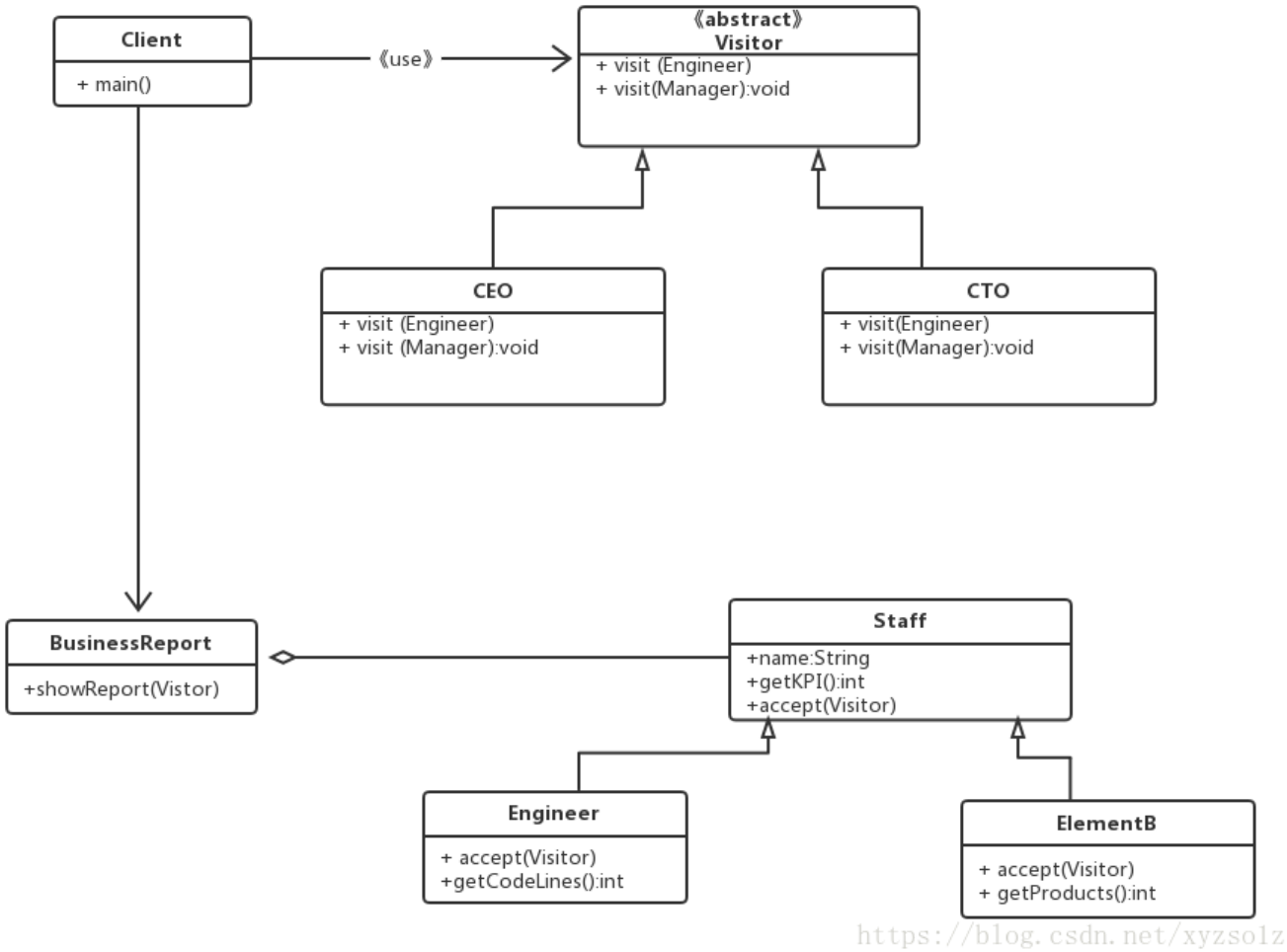
```
package visitorpattern;

public class Client {
        public static void main(String[] args) {
                // 构建报表
                BusinessReport report = new BusinessReport();
                System.out.println("============给CEO看的报表===========");
                // 设置访问者，这里是CEO
                report.showReport(new CEOVisitor());
                System.out.println("===========给CTO看的报表===========");
                // 注入另一个访问者CTO
                report.showReport(new CTOVisitor());
        }
}
```

客户端代码中，首先构造一个报表对象，该对象中维护了所有员工的集合，然后通过报表类的showReport函数为Visitor对象提供一个访问接口，在这个函数中遍历所有的员工，然后调用员工的accept函数接受访问者的访问，每个访问者对不同类型的员工调用对应的visit函数实现不同的操作。

输出如下：

```
============给CEO看的报表===========
经理： 王经理, KPI: 9, 产品数量: 5
工程师： 工程师-gong1,KPI: 4
工程师： 工程师-gong2,KPI: 8
工程师： 工程师-gong3,KPI: 0
工程师： 工程师-gong4,KPI: 4
工程师： 工程师-gong5,KPI: 0
===========给CTO看的报表===========
经理： 王经理,产品数量：5
工程师： 工程师-gong1,代码行数： 91801
工程师： 工程师-gong2,代码行数： 17428
工程师： 工程师-gong3,代码行数： 83083
工程师： 工程师-gong4,代码行数： 70246
工程师： 工程师-gong5,代码行数： 99727
```

在上述示例中，Staff扮演了Element角色，而Engineer和Manager都是ConcreteElement;CEO和CTO都是具体的Vistior对象；而BussinessReport就是ObjectStructure；Client就是客户端代码。



资源地址

《Android源码设计模式解析与实战 16章》