创建类：

Abstract Factory

Builder

Factory Method

Prototype

Singleton

结构：

Adapter

Bridge

Composite

Decorator

Facade

Flyweight

Proxy

行为：

Chain Of Responsibility

Command

Interpreter

Iterator

Mediator

Memento

Observer

State

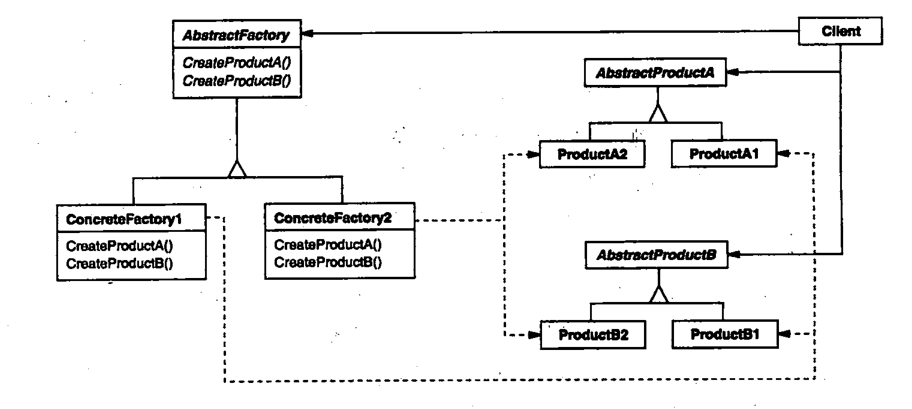
Strategy

Template Method

Visitor

### Abstract Factory：

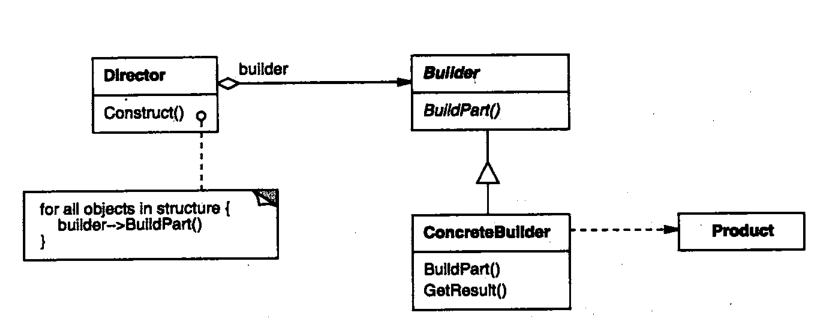
AbstractFactor类通常用工厂方法（Factory Method）实现，但他们也可以用Prototype实现



### Builder：

Abstract Factory与Builder相似，因为他也可以创建复杂对象。主要的区别是Builder模式着重于一步步构造一个复杂对象。而AbstractFacotory着重于多个锡类的产品对象（简单的或是复杂的）。Builder在最后的异步返回产品，而对于Abstract Factory来说，产品是立即返回的。

Composite通常是用Buidler生成的。

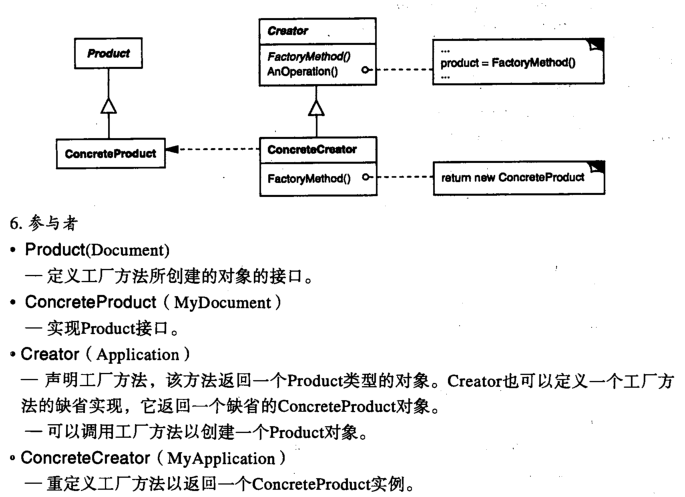


### Factory Method

Abstract Factory经常用工厂方法来实现。

工厂方法通常在Template Method中被调用。

Prototypes不需要常见Creator子类。但是，他们通常要求一个针对Product类的Initialize操作。Creator使用Initialize来初始化对象。而Factory Method不需要这样的操作。



### Ptototype

大量使用Composite和Decorator模式的设计通常也可以从Prototype模式处获益。

参与者：

Prototype

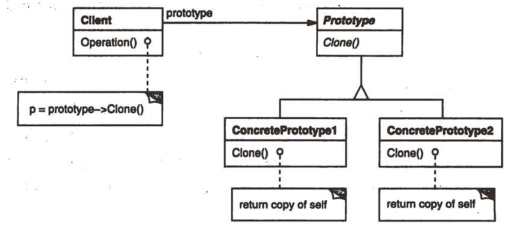
声明一个克隆自身的接口

ConcretePrototype

实现一个克隆自身的操作

Client

让一个原型克隆自身从而创建一个新的对象



### 单例模式

### Adapter

模式Bridge的结构与对象适配器类似，但是Bridge模式的出发点不同：Bridge目的是将接口部分和显示部分分离，从而对他们可以较为容易也相对独立的加以改变。而Adapter则也为这改变一个已有对象的接口。

Decorator模式增强了其他对象的功能而同时又不改变它的接口。因此decorator对应用程序的透明性比适配器要好。结果是decorator支持递归组合，而纯粹使用适配器是不可能实现这一点的。

模式Proxy在不改变它的接口的条件下，为另一个对象定义了一个代理。

参与者：

Target：

定义Client使用的与特定领域相关的接口

Client：

复合Target接口的对象协同

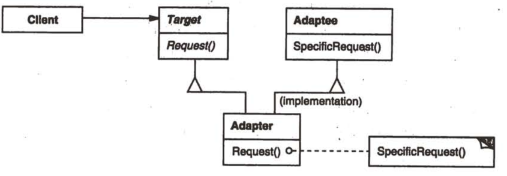
Adaptee：

定义一个已经存在的接口，这个接口需要适配

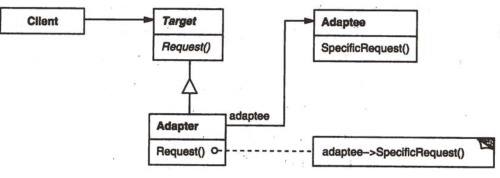
Adapter：

对Adaptee的接口与Target接口进行适配

**类适配器：**



**对象适配器**



### Bridge

AbstractFactory模式可以用来创建和配置一个特定的Bridge模式。

Adapter模式用来帮助无关的类协同工作，它通常在系统设计完成后才会被使用。然而，Bridge模式则是在系统开始时就被使用，它使得抽线接口和实现部分可以独立进行改变。

参与者：

Abstraction：

定义抽象类的接口

维护一个指向Implementor类型对象的指针

RefiendAbstraction

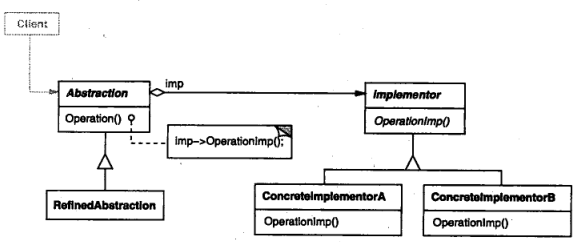
扩充由Abstraction定义的接口

Implementor

定义实现了的接口改接口不一定要与Abstraction的接口完全一致：事实上这两个 接口可以我完全不同。一般来讲,Implementor接口仅提供基本操作，而Abstraction 则定义了基于这些基本操作的较高层次的操作。

ConcreteImplementor

实现Implementor接口并定义它的具体实现



### Composite

### Decorator

Decorator模式不同于adapter模式，因为装饰仅改变对象的侄儿而不改变它的接口，而适配器讲给对象一个全新的接口

Composite模式可以将装饰视为一个退化的、仅有一个组件的组合。然而，装饰仅给对象添加一些额外的职责——他的目的不在于对象聚集。

Strategy模式用一个装饰你可以改变对象的外边：而Strategy模式使得你可以改变对象的内核。这是改变对象的两种途径。

Compnent

定义一个对象接口

ConcreteComponent

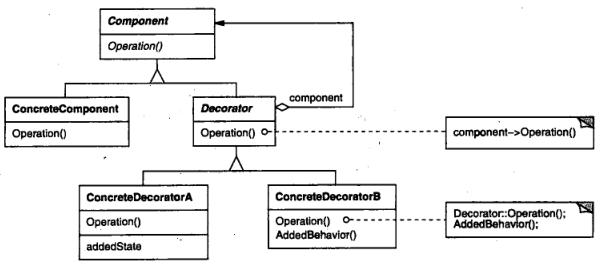
定义一个对象，可以给这个对象添加一些职责

Decorator

维持一个指向Component对象的指针，并定义一个与Component接口一致的接口

ConcreateDecorator

向组件添加职责



### Facade

Abstract Factory模式可以与facade模式一起使用以一同一个接口，这一接口可用来以一种子系统独立的方式创建子系统对象。Abstract Factory也可以代替Facade模式隐藏那些与平台相关的类。

Mediator模式与Facade模式的相似之处是，它抽象了一些已有的类的功能。然而，Mediator的目的是对同时之间的任意通讯进行抽象，通常集中不属于任何单个对象的功能。Mediator的同时对象知道终结者并与它通信，而不是直接与其他同类对象通信。相对而言，Facade模式仅对子系统对象的接口进行抽象，从而使他们更容易使用；他并不定义新功能，子系统也不知道facade的存在。

通常来讲，仅需要一个Facade对象，因此Facade对象通常属于Singleton模式。

参与者：

Facede：

知道哪些子类系统负责吹请求

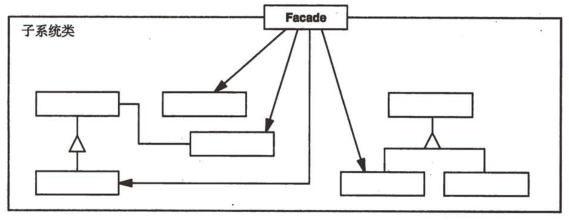
将客户的请求代理给适当的子系统对象

SubSystem Classes

实现子系统的功能

处理由facade对象指派的任务

没有facade的任何相关信息；没有指向facade的指针



### Flyweight

Flyweight模式通常和Composite模式结合起来，用共享叶节点的有向无环图实现一个逻辑上的层次结构。

通常，最好用Flyweight实现State和Strategy对象。

参与者：

Flyweight

描述一个接口，通常这个接口flyweight可以接受并作用于外部的状态

ConcreteFlyweight

事项Flyweight接口，并为内部状态增加存储空间。ConcreteFlyweight对象必须是 可共享的。它所存储的状态必须是内部的；即，它必须独立于ConcreateFlyweight 对象的场景。

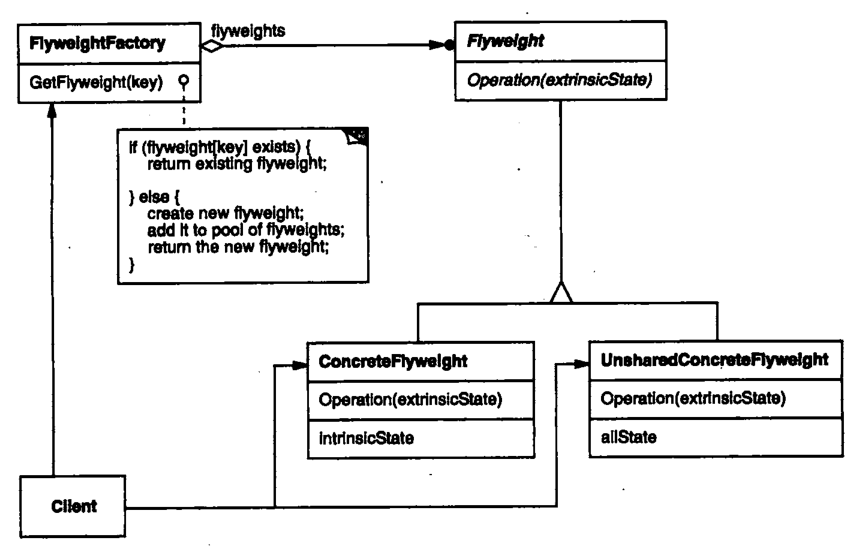
UnshareConcreateFlyweight

并非所有的Flyweight子类都需要被共享。Flyweight接口使共享称为可能，但它并 不强制共享。在Flyweight对象结构的某些层次，UnsharedConcreateFlyweight对象 通常将ConcreateFlyweight对象作为子节点。

FlyweightFactory

创建并管理flyweight对象

确保合理地共享flyweight。当用户请求一个flyweight时，FlyweightFactory对象提 供一个已创建的实例或者创建一个（如果不存在的话）。



### Proxy

参与者：

Proxy

保存一个引用使得代理可以访问实体。若RealSubject和Subject的接口相同，Proxy 会应用Subject。

提供一个与Subjcet的接口相同的接口，这样代理就可以用来代理实体。

控制对实体的UC你区，并可能负责创建和删除它。

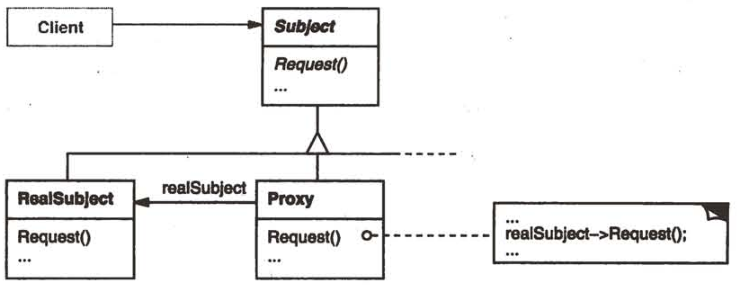
其他功能依赖于代理的类型

Subject

定义RealSubject和Proxy的公用接口，这样就在任何使用RealSubject的地方都可 以使用Proxy

RealSubject

定义Proxy所代表的实体



### Chain of Responsibility

责任链常与Composite一起使用。这种情况下，一个构件的父构件可作为它的后继。

参与者：

Handler

定义一个狐狸请求的接口

ConcreateHandler

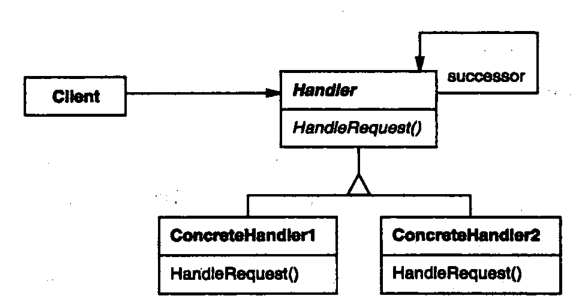
处理它所负责的请求

可访问它的后继者

入股欧克处理改请求，就处理之；否则将请求转发给他的后继者

Client

想链上的具体处理者对象提交请求



### Command

Composite模式可被用来实现宏命令

Memento模式可用来保持某个状态，命令用这一状态取消他的效果。

参与者：

Command

声明执行操作的接口

ConcreateCommand

将一个接受者对象绑定于一个动作

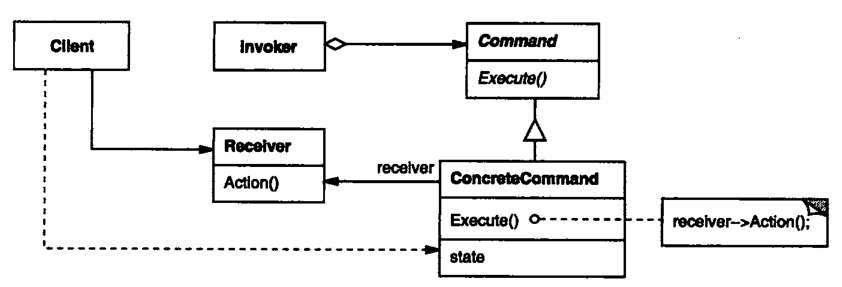
调用接受者相应的操作，以实现Execute

Client

创建一个具体命令对象并设定他的接受者

Invoker

要求改米宁执行这个请求



### Interpreter

Compsite模式：抽象语法树是一个复合模式的实例

Flyweight模式：说明了如何在抽象语法树中共享终结符

Iterator：解析器可用一个迭代器遍历改结构

Visitor：可用来在一个类上维护抽象语法树中的ge6jieidan的行为。

参与者

AbstractExpression

声明一个抽象的解析操作，这个接口为抽象语法树中所有的节点所共享

TerminalExpression

实现与文法中的终结符相关联的解释操作

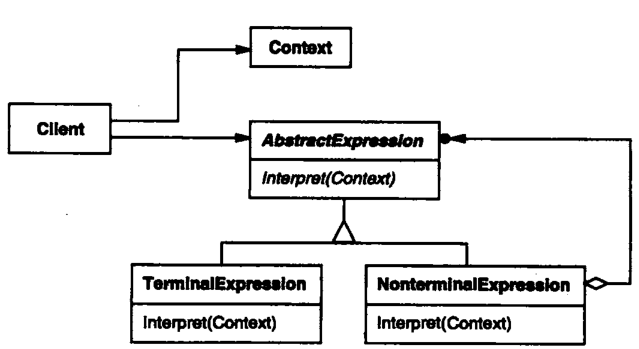
NonterminalExpression

Context

上下文

Client

客户



### Iterator

### Mediator

Facade与中介者的不同之处在于它时对应对象子系统进行抽象，从而提供了一个更为方便的接口。它的协议是单项的，即Facade对象对这个子系统类提出请求，但反之则不行。相反，Mediator提供了各Colleague对象不支持或不能支持的协作行为，而且协议是多向的。

Colleague可使用Observer模式与Mediator通信。

参与者：

Mediator

终结者定义一个皆苦用于与个同事对象通信

ConcreateMediator

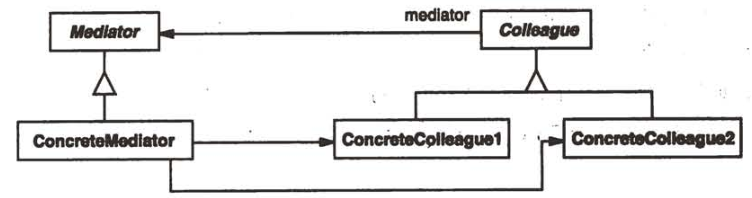
具体终结者通过协调各同时对象实现协作行为

了解并维护它的各个同事

Colleague class

每一个同时类都知道它的中介者对象

每一个同时对象在需与其他的同事通信的时候，与它的中介者通信



### Memento

Command命令可是备忘录来为可撤销的操作维护状态

参与者

Memento

备忘录存储元发起对象的内部状态。元发起根据需要决定备忘录存储元发起的哪些 内部状态

防止元发起以外的其他对象访问备忘录。备忘录实际上有两个接口，管理者只能看 到备忘录的摘接口——它只能将备忘录传递给其他对象。相反，元发起能够看到一 个宽接口，允许它访问返回到先前状态所需的所有数据。理想的情况是只允许生成 本备忘录的那个元发起访问本备忘录的内部状态

Originator

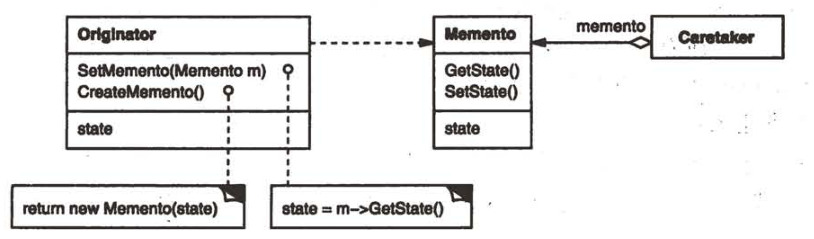
元发起创建一个备忘录，用以记录当前时刻它的内部状态。

使用备忘录恢复内部状态。

Caretaker

负责保存好备忘录

不能对备忘录的内容进行操作或检查



### Observer

Mediator通过封装福安的更新予以，ChangeManager充当目标和观察者之间的中介者。

Singleton：ChangeManager可使用Singleton模式来保证他是唯一的并且是可全局访问的。

参与者：

Subject

目标知道它的观察者。可以哟亿多个观察者观察同一目标

提供注册和删除观察者对象的接口

Observer

为那些在目标发生改变时需获得通知的对象定义一个更新接口。

ConcreateSubject

将有关状态存入各ConcreateObserver对象

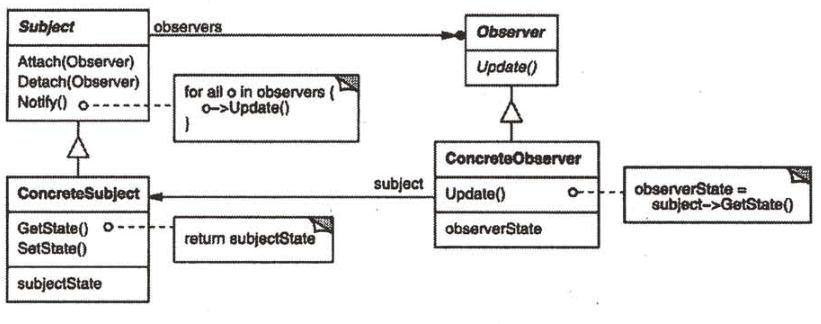
当它的状态发生改变时，向他的各个观察者发出通知。

ConcreateObserver

维护一个指向ConcreateSubject对象的引用

存储有关状态，这些状态应与目标的状态保持一致

实现Observer的更新接口以使自身状态与目标的状态保持一致



### State

Flyweight模式解释了合适以及怎样共享状态对象

状态对象通常是Singleton

参与者：

Context

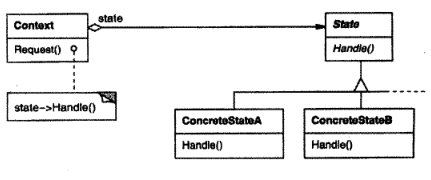
定义客户感兴趣的接口

Status

定义一个接口 以封装与Context的一个特定状态相关的行为

ConcreateState

每一子类实现一个与Context的一个特定状态相关的行为



### Strategy

Flyweight：Strategy对象经常是很好的轻量级对象。

参与者

Strategy

定义所有支持的算法的公共接口。Context使用这个接口来调用某ConcreateStrategy定 义的算法

ConcreateStrategy

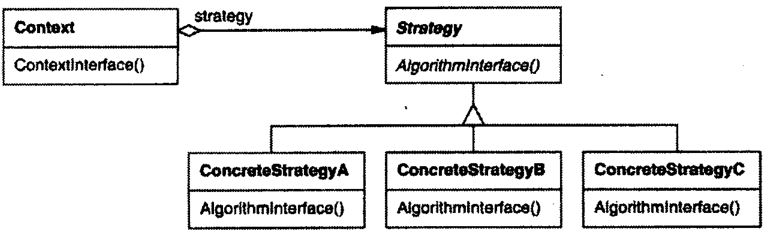
以Strategy接口实现某具体算法

Context

用一个ConcreateStrategy对象来配置

维护一个对Strategy对象的引用

可定义一个接口来让Strategy访问它的数据



### Template Method

FactoryMethod模式长被模板方法调用。

Strategy：模板方法使用集成来改变算法的一部分。Strategy使用委托来改变整个算法。

参与者

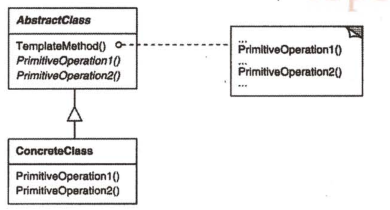
AbstractClass

定义抽象的源于操作，具体的子类重定义他们以实现一个算法的各步骤

实现一个模板防范，定义一个算法的股价，改模板方法不仅调用原语操作，也调用定义 在AbstractClass或其他对象中的操作

ConcreateClass

实现原语操作以完成算法中与特定子类相关的步骤



### Visitor

Composite：访问者可以用于对一个由Composite模式定义的对象结构进行操作

Interpreter：访问者可以英语解释

参与者

Visitor

为改对象结构中ConcreateElement的每一个类声明一个Visit操作。改操作的名字和特 征标识了发送Visit请求给该访问者的那个类。这使得访问者可以确定整被访问元素的 具体的类。这样访问者就可以通过该元素的特定接口直接访问它。

ConcreteVisitor

实现每个由Visitor声明的操作。每个操作实现本算法的一部分，而改算法片段乃是对 应于结构中对象的类。ConcreteVisitor为改算法提供了上下文并储存它的局部状态。这 一状态常常在遍历改结构的过程中累计结果

Element

定义一个Accept操作，它以一个访问者为参数

ConcreateElement

实现Accept操作，改操作以一个访问者为参数。

ObjectStructure

能枚举它的元素

可以提供一个高层的接口以允许改访问者访问它的元素

可以是一个符合或是一个集合，如一个列表或一个无序集合

