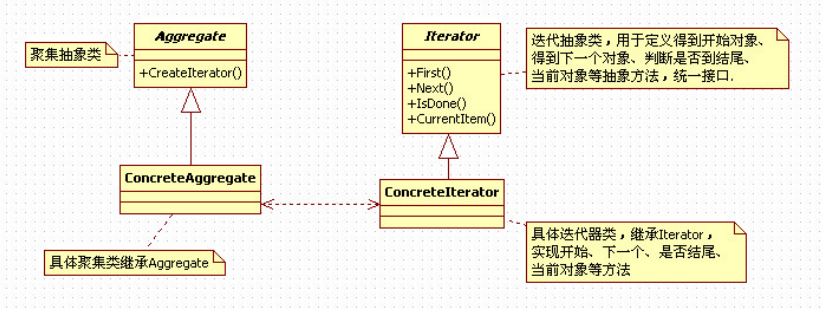
迭代器模式：提供一种方法顺序访问一个聚合对象中个各个元素，而不暴露该对像的内部表示.

模式的动机： （1）一个聚合对象，如一个列表(List)或者一个集合(Set)，应该提供一种方法来让别人可以访问它的元素，而又不需要暴露它的内部结构。 （2）针对不同的需要，可能还要以不同的方式遍历整个聚合对象，但是我们并不希望在聚合对象的抽象层接口中充斥着各种不同遍历的操作。 （3）怎样遍历一个聚合对象，又不需要了解聚合对象的内部结构，还能够提供多种不同的遍历方式， 这就是迭代器模式所要解决的问题。

为遍历不同的聚集结构提供如开始、下一个、是否结束、当前哪 一项等统一接口。迭代器模式就是分离了集合对象的遍历行为，抽象出一个迭代器类来负责，这样既可以做到不暴露集合的内部结构，又可让外部代码透明地访问集 合内部的数据。




```
#include<iostream>
#include<vector>
#include<string>
using namespace std;
class Iterator{
public:
    virtual ~Iterator(){}
    virtual string First() =0;
    virtual string Next() = 0;
     virtual string Current() = 0;
     virtual bool IsEnd() = 0;
};
class Aggregate{
public:
    virtual ~Aggregate(){}
    virtual int Count() = 0;
    virtual void Push(const string &str) = 0;
    virtual string Pop(const int index) = 0;
    virtual Iterator * CreateIterator() = 0;
};
class ConcretIterator : public Iterator{
private:
    Aggregate *m_Agre;
    int m_nCurrent;
public:
    ConcretIterator(Aggregate *a){
        m_Agre = a;
        m_nCurrent = 0;
    }
    ~ConcretIterator(){
```

```
        if(NULL != m_Agre)
            delete m_Agre;
    }
    string First(){
        return m_Agre->Pop(0);
    }
    string Next(){
        string strRet;
        m_nCurrent++;
        if(m_nCurrent < m_Agre->Count()){
            strRet =  m_Agre->Pop(m_nCurrent);
        }
        return strRet;
    }
    string Current(){
        return m_Agre->Pop(m_nCurrent);
    }
    bool IsEnd(){
        return m_nCurrent >= m_Agre->Count() ? true : false;
    }
};
class ConcretAggregate : public Aggregate{
private:
    vector<string> m_vecItems;
    Iterator *m_pIter;
public:
    ConcretAggregate(){
        m_vecItems.clear();
        m_pIter = NULL;
    }
    ~ConcretAggregate(){
        if(m_pIter != NULL)
            delete m_pIter;
    }
    Iterator* CreateIterator(){
        if(NULL == m_pIter){
            m_pIter = new ConcretIterator(this);
        }
        return m_pIter;
    }
    int Count(){
        return m_vecItems.size();
    }
    void Push(const string &str){
        m_vecItems.push_back(str);
    }
    string Pop(const int index){
        string strRet;
        if(index < Count()){
            strRet = m_vecItems[index];
        }
        return strRet;
    }
};
```

```
85 int main(){
86     ConcretAggregate *pCA = new ConcretAggregate();
87     pCA->Push("AA");
88     pCA->Push("BB");
89     pCA->Push("CC");
90     pCA->Push("DD");
91     Iterator *Iter = pCA->CreateIterator();
92     string strItem = Iter->First();
93     while (!Iter->IsEnd())
94     {
95         cout << Iter->Current() << " is OK" << endl;
96         Iter->Next();
97     }
98     return 0;
99 }
```

```
192:DesignPattnsStudy weishichun$ g++ -o Iterator.out Iterator_1.cpp
192:DesignPattnsStudy weishichun$ ./Iterator.out
AA is OK
BB is OK
CC is OK
DD is OK
192:DesignPattnsStudy weishichun$
```