链表

数组各种操作的时间复杂度

插入:O(n)

删除:O(n)

修改:O(1)

查询:O(n)

线性表:

表中任意元素只有唯一的前驱,唯一的后继

头元素没有前驱,尾元素没有后继

```
5
6   class CA
7   {
8   public:
9       CA() { cout << "CA::CA()" << endl; }  已用时间 <= 6ms
10      CA(int n) { cout << "CA::CA()" << endl; }
11  };
12
13  int main()
14  {
15      CA a{};
16
17      return 0;
```

这样写才调默认构造

CList.h

```
#pragma once

#include <utility>

// 双向链表
template<typename T>
class CList
{
public:
    struct NODE
    {
        NODE() :m_pPre(nullptr), m_pNext(nullptr), m_val{} {}
        NODE(const T& val):m_pPre(nullptr), m_pNext(nullptr), m_val(val) {}
        T m_val;// 存储数据
        NODE* m_pPre;// 前驱节点的指针
        NODE* m_pNext;// 后继节点的指针
    };
    using PNODE = NODE*;
public:
```

```
    CList();
    CList(const CList& obj);
    CList(CList&& obj);
    CList(std::initializer_list<T> il);
    CList& operator=(const CList& obj);
    virtual ~CList();

    // 插入
    void PushHead(const T& val);
    void PushBack(const T& val);
    void Insert(PNODE pNode, const T& val);// 插入到指定位置的前面

    // 删除
    void PopHead();
    void PopBack();
    void Delete(PNODE pNode);

    // 修改
    void Modify(PNODE pNode, const T& val);

    // 查询
    PNODE Find(const T& val);

    // 清空
    void Clear();

    // 获取个数
    size_t GetCount();

    // 判断是否为空
    bool IsEmpty();
private:
    void Init();// 置空
private:
    PNODE m_pHead;// 头哨兵指针
    PNODE m_pTail;// 尾哨兵指针
    size_t m_nCount;// 节点的个数
};

template<typename T>
inline CList<T>::CList()
{
    Init();
}

template<typename T>
inline CList<T>::CList(const CList& obj)
{
    Init();
    *this = obj;
}

template<typename T>
inline CList<T>::CList(CList&& obj)
{
    Init();
    m_pHead = obj.m_pHead;
    m_pTail = obj.m_pTail;
```

```cpp
    m_nCount = obj.m_nCount;
    obj.m_pHead = nullptr;
    obj.m_pTail = nullptr;
    obj.m_nCount = 0;
}

template<typename T>
inline CList<T>::CList(std::initializer_list<T> il)
{
    Init();

    for (auto& val : il)
    {
        PushBack(val);
    }
}

template<typename T>
inline CList<T>& CList<T>::operator=(const CList& obj)
{
    if (*this == obj)
    {
        return *this;
    }
    // 清空自己
    Clear();

    // 循环插入
    PNODE pNode = obj.m_pHead->m_pNext;
    while (pNode != obj.m_pTail)// 从头结点开始遍历,直到尾哨兵
    {
        PushBack(pNode->obj.m_val);
        pNode = pNode->m_pNext;
    }
    return *this;
}

template<typename T>
inline CList<T>::~CList()
{
    // 元素内容清掉
    Clear();
    // 释放哨兵
    if (m_pHead != nullptr)
    {
        delete m_pHead;
        m_pHead = nullptr;
    }
    if (m_pTail != nullptr)
    {
        delete m_pTail;
        m_pTail = nullptr;
    }
}

template<typename T>
inline void CList<T>::PushHead(const T& val)
{
```

```cpp
    // 头哨兵后面是头结点,val插到头节点前面,val就是新的头结点
    Insert(m_pHead->m_pNext, val);
}

template<typename T>
inline void CList<T>::PushBack(const T& val)
{
    Insert(m_pTail, val);// 插到尾哨兵前面
}

template<typename T>
inline void CList<T>::Insert(PNODE pNode, const T& val)
{
    // 检查参数
    if (pNode == nullptr)
    {
        return;
    }
    // 构造新结点
    PNODE pNewNode = new NODE(val);
    if (pNewNode == nullptr)
    {
        return;
    }

    // 插入新结点
    PNODE pPre = pNode->m_pPre;
    pPre->m_pNext = pNewNode;
    pNewNode->m_pPre = pPre;
    pNewNode->m_pNext = pNode;
    pNode->m_pPre = pNewNode;

    // 个数增加
    ++m_nCount;
}

template<typename T>
inline void CList<T>::PopHead()
{
    Delete(m_pHead->m_pNext);
}

template<typename T>
inline void CList<T>::PopBack()
{
    Delete(m_pTail->m_pPre);
}

template<typename T>
inline void CList<T>::Delete(PNODE pNode)
{
    // 检查参数
    if (pNode == nullptr || pNode == m_pHead || pNode == m_pTail)
    {
        return;
    }
    // 拿到前驱和后继
    PNODE pPre = pNode->m_pPre;
```

```
194     PNODE pNext = pNode->m_pNext;
195
196     pPre->m_pNext = pNext;
197     pNext->m_pPre = pPre;
198
199     // 删除
200     delete pNode;
201
202     // 个数减少
203     --m_nCount;
204 }
205
206 template<typename T>
207 inline void CList<T>::Modify(PNODE pNode, const T& val)
208 {
209     // 检查参数
210     if (pNode == nullptr || pNode == m_pHead || pNode == m_pTail)
211     {
212         return;
213     }
214     pNode->m_val = val;
215 }
216
217 template<typename T>
218 inline typename CList<T>::PNODE CList<T>::Find(const T& val)
219 {
220     // 保存头结点
221     PNODE pNode = m_pHead->m_pNext;
222     while (pNode != m_pTail)// 从头结点开始遍历,直到尾哨兵
223     {
224         if (pNode->m_val == val)
225         {
226             return pNode;
227         }
228         pNode = pNode->m_pNext;
229     }
230     return nullptr;
231 }
232
233 template<typename T>
234 inline void CList<T>::Clear()
235 {
236     while (!IsEmpty())
237     {
238         PopBack();
239     }
240 }
241
242 template<typename T>
243 inline size_t CList<T>::GetCount()
244 {
245     return m_nCount;
246 }
247
248 template<typename T>
249 inline bool CList<T>::IsEmpty()
250 {
251     return m_nCount == 0;
```

```
}

template<typename T>
inline void CList<T>::Init()
{
    m_pHead = new NODE;
    m_pTail = new NODE;
    m_pHead->m_pNext = m_pTail;
    m_pTail->m_pPre = m_pHead;
    m_nCount = 0;
}
```

双向链表.cpp

```
// 链表.cpp

#include <iostream>
#include "CList.h"
using namespace std;

class CA
{
public:
    CA() :m_p(nullptr) {}
    CA(int n) :m_p(new int(n)) {}
    CA(const CA& obj) :m_p(new int(*obj.m_p)) {}
    CA& operator=(const CA& obj)
    {
        if (m_p != nullptr)
        {
            delete m_p;
        }
        m_p = new int(*obj.m_p);
        return *this;
    }
    ~CA()
    {
        if (m_p != nullptr)
        {
            delete m_p;
        }
    }
private:
    int* m_p;
};

CList<CA> Foo()
{
    CList<CA> lst({2, 3, 4, 5, 6, 7, 8, 9});
    return lst;
}

int main()
{
    auto lst1 = Foo();

    // 14 7 3 5
```

```
    CList<int> lst;
    lst.PushBack(3);
    lst.PushBack(5);
    lst.PushHead(7);
    lst.PushHead(14);

    auto pNode = lst.Find(7);
    lst.Insert(pNode, 13);

    pNode = lst.Find(66);

    lst.Delete(lst.Find(5));
    lst.Delete(lst.Find(99));
    lst.PopHead();
    lst.PopBack();

    return 0;
}
```

迭代器

Clist.hpp

```
#pragma once
#include <utility>
#include <assert.h>

// 双向链表
template<typename T>
class CList
{
public:
    struct NODE
    {
        NODE() :m_pPre(nullptr), m_pNext(nullptr), m_val{} {}
        NODE(const T& val) :m_pPre(nullptr), m_pNext(nullptr), m_val(val) {}
        T m_val;// 存储数据
        NODE* m_pPre;// 前驱节点的指针
        NODE* m_pNext;// 后继节点的指针
    };
    using PNODE = NODE*;

    class CIterator
    {
    public:
        CIterator(PNODE pCur, PNODE pHead, PNODE pTail) :
            m_pCur(pCur), m_pHead(pHead), m_pTail(pTail) {}
        CIterator& operator++()// 前++
        {
            // 检查,不能移动到尾哨兵后面
            assert(m_pCur != m_pTail);
            // 位置后移
            m_pCur = m_pCur->m_pNext;

            return *this;
        }
        bool operator!=(const CIterator& obj)
```

```
        {
            return m_pCur != obj.m_pCur;
        }
        T& operator*()
        {
            // 尾哨兵不能取内容
            assert(m_pCur != m_pTail);
            return m_pCur->m_val;
        }

    private:
        PNODE m_pCur;
        PNODE m_pHead;// 头哨兵,负责边界检查
        PNODE m_pTail;// 尾哨兵,负责边界检查
    };
public:
    CList();
    CList(const CList& obj);
    CList(CList&& obj);
    CList(std::initializer_list<T> il);
    CList& operator=(const CList& obj);
    virtual ~CList();

    // 获取迭代器
    CIterator begin();
    CIterator end();

    // 插入
    void PushHead(const T& val);
    void PushBack(const T& val);
    void Insert(PNODE pNode, const T& val);// 插入到指定位置的前面

    // 删除
    void PopHead();
    void PopBack();
    void Delete(PNODE pNode);

    // 修改
    void Modify(PNODE pNode, const T& val);

    // 查询
    PNODE Find(const T& val);

    // 清空
    void Clear();

    // 获取个数
    size_t GetCount();

    // 判断是否为空
    bool IsEmpty();
private:
    void Init();// 置空
private:
    PNODE m_pHead;// 头哨兵指针
    PNODE m_pTail;// 尾哨兵指针
    size_t m_nCount;// 节点的个数
};
```

```cpp

template<typename T>
inline CList<T>::CList()
{
    Init();
}

template<typename T>
inline CList<T>::CList(const CList& obj)
{
    Init();
    *this = obj;
}

template<typename T>
inline CList<T>::CList(CList&& obj)
{
    Init();
    m_pHead = obj.m_pHead;
    m_pTail = obj.m_pTail;
    m_nCount = obj.m_nCount;
    obj.m_pHead = nullptr;
    obj.m_pTail = nullptr;
    obj.m_nCount = 0;
}

template<typename T>
inline CList<T>::CList(std::initializer_list<T> il)
{
    Init();

    for (auto& val : il)
    {
        PushBack(val);
    }
}

template<typename T>
inline CList<T>& CList<T>::operator=(const CList& obj)
{
    if (*this == obj)
    {
        return *this;
    }
    // 清空自己
    Clear();

    // 循环插入
    PNODE pNode = obj.m_pHead->m_pNext;
    while (pNode != obj.m_pTail)// 从头结点开始遍历,直到尾哨兵
    {
        PushBack(pNode->obj.m_val);
        pNode = pNode->m_pNext;
    }
    return *this;
}

template<typename T>
```

```cpp
inline CList<T>::~CList()
{
    // 元素内容清掉
    Clear();
    // 释放哨兵
    if (m_pHead ≠ nullptr)
    {
        delete m_pHead;
        m_pHead = nullptr;
    }
    if (m_pTail ≠ nullptr)
    {
        delete m_pTail;
        m_pTail = nullptr;
    }
}

template<typename T>
inline typename CList<T>::CIterator CList<T>::begin()
{
    return CIterator(m_pHead→m_pNext, m_pHead, m_pTail);
}

template<typename T>
inline typename CList<T>::CIterator CList<T>::end()
{
    return CIterator(m_pTail, m_pHead, m_pTail);
}

template<typename T>
inline void CList<T>::PushHead(const T& val)
{
    // 头哨兵后面是头结点,val插到头节点前面,val就是新的头结点
    Insert(m_pHead→m_pNext, val);
}

template<typename T>
inline void CList<T>::PushBack(const T& val)
{
    Insert(m_pTail, val);// 插到尾哨兵前面
}

template<typename T>
inline void CList<T>::Insert(PNODE pNode, const T& val)
{
    // 检查参数
    if (pNode == nullptr)
    {
        return;
    }
    // 构造新结点
    PNODE pNewNode = new NODE(val);
    if (pNewNode == nullptr)
    {
        return;
    }

    // 插入新结点
```

```
209     PNODE pPre = pNode->m_pPre;
210     pPre->m_pNext = pNewNode;
211     pNewNode->m_pPre = pPre;
212     pNewNode->m_pNext = pNode;
213     pNode->m_pPre = pNewNode;
214 
215     // 个数增加
216     ++m_nCount;
217 }
218 
219 template<typename T>
220 inline void CList<T>::PopHead()
221 {
222     Delete(m_pHead->m_pNext);
223 }
224 
225 template<typename T>
226 inline void CList<T>::PopBack()
227 {
228     Delete(m_pTail->m_pPre);
229 }
230 
231 template<typename T>
232 inline void CList<T>::Delete(PNODE pNode)
233 {
234     // 检查参数
235     if (pNode == nullptr || pNode == m_pHead || pNode == m_pTail)
236     {
237         return;
238     }
239     // 拿到前驱和后继
240     PNODE pPre = pNode->m_pPre;
241     PNODE pNext = pNode->m_pNext;
242 
243     pPre->m_pNext = pNext;
244     pNext->m_pPre = pPre;
245 
246     // 删除
247     delete pNode;
248 
249     // 个数减少
250     --m_nCount;
251 }
252 
253 template<typename T>
254 inline void CList<T>::Modify(PNODE pNode, const T& val)
255 {
256     // 检查参数
257     if (pNode == nullptr || pNode == m_pHead || pNode == m_pTail)
258     {
259         return;
260     }
261     pNode->m_val = val;
262 }
263 
264 template<typename T>
265 inline typename CList<T>::PNODE CList<T>::Find(const T& val)
266 {
```

```
    // 保存头结点
    PNODE pNode = m_pHead->m_pNext;
    while (pNode != m_pTail)// 从头结点开始遍历,直到尾哨兵
    {
        if (pNode->m_val == val)
        {
            return pNode;
        }
        pNode = pNode->m_pNext;
    }
    return nullptr;
}

template<typename T>
inline void CList<T>::Clear()
{
    while (!IsEmpty())
    {
        PopBack();
    }
}

template<typename T>
inline size_t CList<T>::GetCount()
{
    return m_nCount;
}

template<typename T>
inline bool CList<T>::IsEmpty()
{
    return m_nCount == 0;
}

template<typename T>
inline void CList<T>::Init()
{
    m_pHead = new NODE;
    m_pTail = new NODE;
    m_pHead->m_pNext = m_pTail;
    m_pTail->m_pPre = m_pHead;
    m_nCount = 0;
}
```

迭代器.cpp

```
// 迭代器.cpp
#include <iostream>
#include <list>
#include "CList.hpp"
using namespace std;

int main()
{
#if 0
    list<int> lst({ 1, 4, 5, 6, 7, 8, 9 });
    list<int>::iterator itr = lst.begin();
```

```
    itr++;
    --itr;
    // --itr;// 有检查,会报错.是否是头结点
    auto itr0 = lst.end();// 尾哨兵
    itr0--;
    itr0++;
    // itr0++;// 有检查,会报错.是否是尾结点

#endif // list库的迭代器

    CList<int> lst({ 1, 3, 5 ,6, 7, 8, 9, 11 });
    CList<int>::CIterator itr = lst.begin();
    ++itr;

    for (auto val : lst)// 必须要小写的begin,end才能这样用
    {
        cout << val << endl;
    }

    return 0;
}
```