清華大學
Tsinghua University

《面向对象程序设计基础》项目阅读报告

选题1:TinyXML

经42-计48 陈奕涵

学号：2024011483

一、项目整体介绍

- 项目功能： ①解析XML文档，生成DOM（文档对象模型）结构，可以通过DOM访问、修改 XML文档；②通过DOM从头构建XML文档到内存中。
- 项目目标：TinyXML2 的目标是提供一个轻量、快速、可嵌入的 C++ XML 解析与构建库，适用于对性能与集成要求较高的项目场景。

轻量：不依赖stl，自己创建DryArray动态数组类；采用定制化的内存池 MemPool 管理各类节点内存，有效减少内存占用和碎片。

快速：内存访问优化，由于内存池模式，节点内存连续分布，实现了紧凑的数据结构。

可嵌入：不依赖任何第三方库或平台特性，跨平台兼容性极高。不使用异常（而是错误码），避免引入额外的异常处理机制。核心文件数量少，方便嵌入其他项目。

一、项目整体介绍

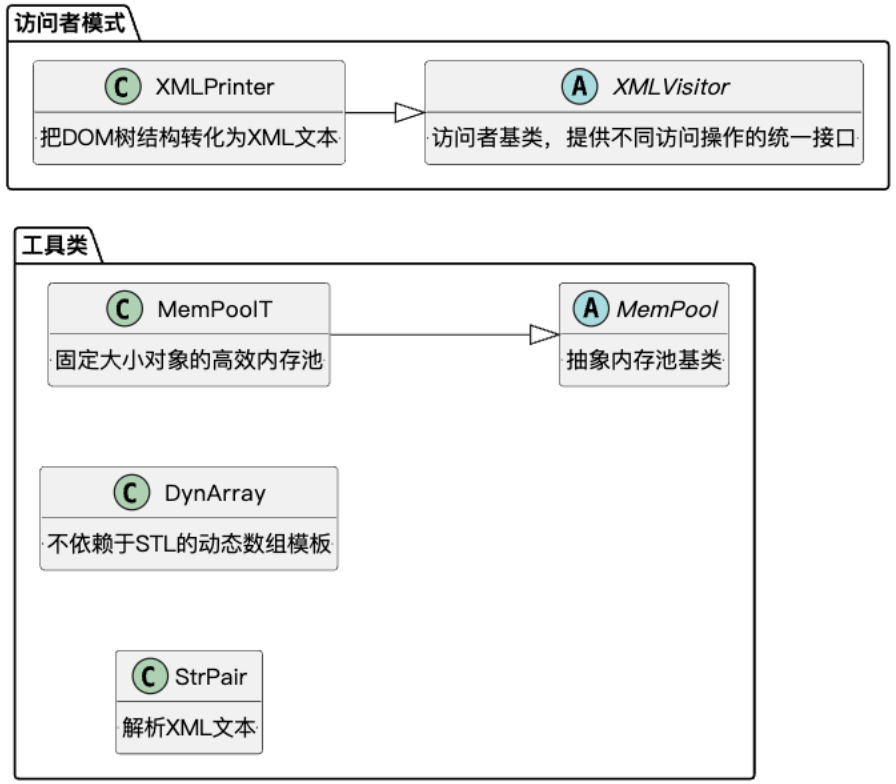
- 使用方法与实现效果：本人编写了Bash 脚本，实现从下载源码到编译、测试（可选安装）tinyxml2库的完整自动化流程。

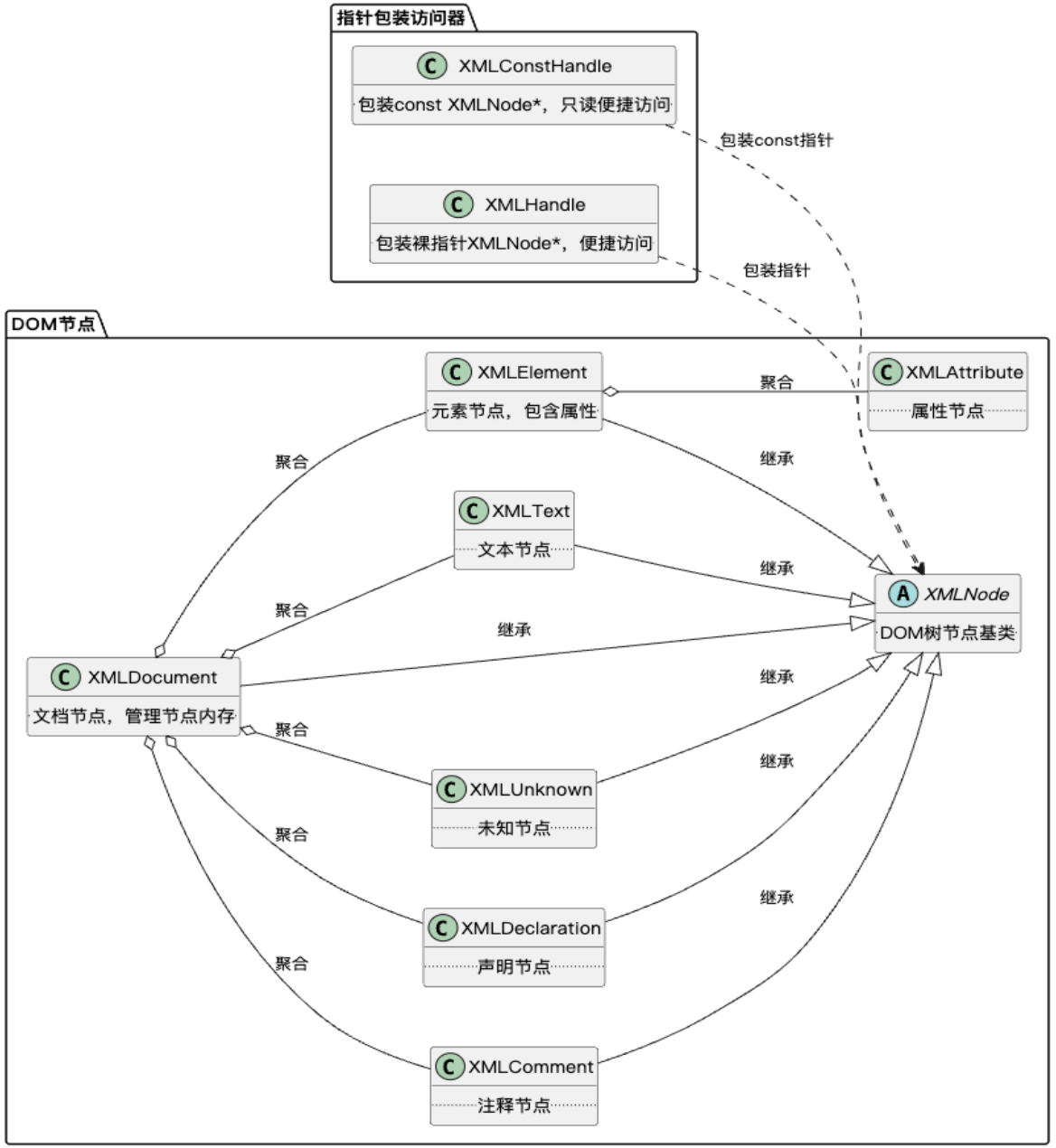
$ tinyxml2.sh

```
#!/bin/bash
# 1. 克隆 tinyxml2 源码
if [ ! -d "tinyxml2" ]; then
    git clone https://github.com/leethomason/tinyxml2.git
fi
cd tinyxml2
# 2. 创建构建目录
mkdir -p build
cd build
# 3. 编译库和测试程序
cmake ..
cmake --build .
# 4. 运行测试
ctest --output-on-failure
# 5. 安装（可选）

# sudo cmake --install .

echo "tinyxml2 build, test and install completed successfully."
```

```
drew@DrewdeMacBook-Air code % bash tinyxml2.sh
正克隆到 'tinyxml2'...
remote: Enumerating objects: 5022, done.
remote: Counting objects: 100% (1371/1371), done.
remote: Compressing objects: 100% (170/170), done.
remote: Total 5022 (delta 1272), reused 1201 (delta 1201), pack-reused 3651 (from 3)
接收对象中：100% (5022/5022), 3.54 MiB | 5.10 MiB/s, 完成.
处理 delta 中：100% (3391/3391), 完成.
-- The C compiler identification is AppleClang 17.0.0.17000013
-- The CXX compiler identification is AppleClang 17.0.0.17000013
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.7s)
-- Generating done (0.0s)
-- Build files have been written to: /Users/drew/code/tinyxml2/build
[ 25%] Building CXX object CMakeFiles/tinyxml2.dir/tinyxml2.cpp.o
[ 50%] Linking CXX static library libtinyxml2.a
[ 50%] Built target tinyxml2
[ 75%] Building CXX object CMakeFiles/xmltest.dir/xmltest.cpp.o
[100%] Linking CXX executable xmltest
[100%] Built target xmltest
Test project /Users/drew/code/tinyxml2/build
    Start 1: xmltest
1/1 Test #1: xmltest ..........................   Passed    0.46 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.47 sec
tinyxml2 build, test and install completed successfully.
```

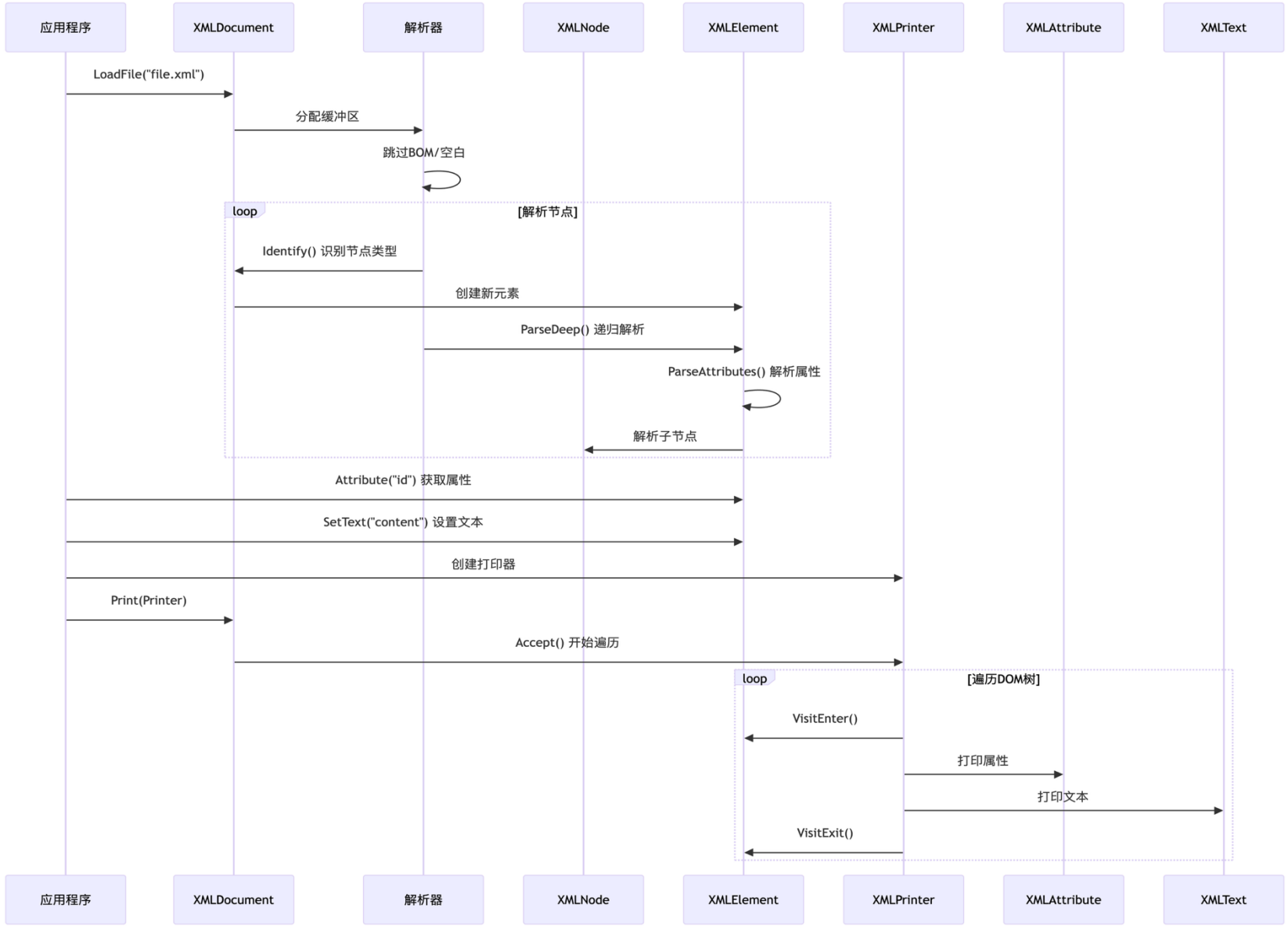
二、框架分析
（1）、UML图


访问者模式
XMLPrinter
把DOM树结构转化为XML文本
XMLVisitor
访问者基类，提供不同访问操作的统一接口
工具类
MemPoolT
固定大小对象的高效内存池
MemPool
抽象内存池基类
DynArray
不依赖于STL的动态数组模板
StrPair
解析XML文本


指针包装访问器
XMLConstHandle
包装const XMLNode*，只读便捷访问
XMLHandle
包装裸指针XMLNode*，便捷访问
包装const指针
包装指针
DOM节点
XMLElement
元素节点，包含属性
XMLAttribute
属性节点
XMLText
文本节点
XMLNode
DOM树节点基类
XMLDocument
文档节点，管理节点内存
XMLUnknown
未知节点
XMLDeclaration
声明节点
XMLComment
注释节点
聚合
继承

二、框架分析

（2）、调用关系图



三、具体功能测试与拓展
（1）、访问者模式

工作流程：

1. 调用方从文档根节点开始遍历。
2. 依次访问各类节点，调用对应的 VisitEnter、VisitExit 或 Visit 方法。
3. 用户在自定义的 XMLVisitor 子类中重写这些方法，实现对不同节点的定制处理 。

核心代码：

```
class TINYXML2_LIB XMLVisitor
{
public:
    virtual ~XMLVisitor() {}

    /// Visit a document.
    virtual bool VisitEnter( const XMLDocument& /*doc*/ )            {
        return true;
    }
    /// Visit a document.
    virtual bool VisitExit( const XMLDocument& /*doc*/ )             {
        return true;
    }

    /// Visit an element.
    virtual bool VisitEnter( const XMLElement& /*element*/, const XMLAttribute* /*firstAttribute*/ )    {
        return true;
    }
    /// Visit an element.
    virtual bool VisitExit( const XMLElement& /*element*/ )          {
        return true;
    }

    /// Visit a declaration.
    virtual bool Visit( const XMLDeclaration& /*declaration*/ )      {
        return true;
    }
    /// Visit a text node.
    virtual bool Visit( const XMLText& /*text*/ )                    {
        return true;
    }
    /// Visit a comment node.
    virtual bool Visit( const XMLComment& /*comment*/ )              {
        return true;
    }
    /// Visit an unknown node.
    virtual bool Visit( const XMLUnknown& /*unknown*/ )              {
        return true;
    }
};
```

三、具体功能测试与拓展
（1）、访问者模式

优越性：

1.实现了将操作与数据结构（XML DOM）解耦，使这些数据结构类更单一职责、职责清晰、易维护，避免受到污染。若将这些操作分散到各种节点类中会导致整个系统难以理解，难以维护和修改。

2.实现了双重分发，调用行为的具体实现依赖于节点的类型（this指针）和访问者的具体实现 （XMLVisitor指针），适合诸如XML DOM此类的复杂的树结构。

3. 提供默认虚函数，保留节点访问迭代的核心逻辑，子类只需重写覆盖拓展部分，不必实现所有虚函数，提高向下兼容性和灵活性。

```
bool XMLDocument::Accept( XMLVisitor* visitor ) const
{
    TIXMLASSERT( visitor );
    if ( visitor->VisitEnter( *this ) ) {
        for ( const XMLNode* node=FirstChild(); node; node=node->NextSibling() ) {
            if ( !node->Accept( visitor ) ) {
                break;
            }
        }
    }
    return visitor->VisitExit( *this );
}
```

三、具体功能测试与拓展
（1）、访问者模式

扩展示例：实现节点类型的统计

```cpp
#include"tinyxml2.h"
#include"tinyxml2_countnode.h"
int main() {
    XMLDocument doc;
    doc.LoadFile("example.xml");

    XMLCounter counter;
    doc.Accept(&counter);  // 通过访问者访问整棵树

    counter.PrintCount();
    return 0;
}
```

样例输出（按照提供的example.xml）：

Element nodes: 10
Text nodes: 6
Comment nodes: 2
Declaration nodes: 1
Unknown nodes: 0

tinyxml2_ > C tinyxml2_countnode.h > XMLCounter > Visit(const XMLUnknown &)

```cpp
#include <iostream>
#include "tinyxml2.h"
using namespace tinyxml2;
class XMLCounter : public XMLVisitor {
public:
    XMLCounter() :elementCount(0),textCount(0),commentCount(0),declarationCount(0),unknownCount(0) {}
    virtual bool VisitEnter(const XMLDocument& doc) override {…
    virtual bool VisitExit(const XMLDocument& doc) override {…
    virtual bool VisitEnter(const XMLElement& element, const XMLAttribute* attribute) override {
        ++elementCount;
        return true;  // 继续遍历
    }
    virtual bool VisitExit(const XMLElement& element) override {…
    virtual bool Visit(const XMLText& text) override {
        ++textCount;
        return true;
    }
    virtual bool Visit(const XMLComment& comment) override {
        ++commentCount;
        return true;
    }
    virtual bool Visit(const XMLDeclaration& declaration) override {
        ++declarationCount;
        return true;
    }
    virtual bool Visit(const XMLUnknown& unknown) override {
        ++unknownCount;
        return true;
    }
    void PrintCount() const {
        std::cout << "Element nodes:      " << elementCount << "\n";
        std::cout << "Text nodes:         " << textCount << "\n";
        std::cout << "Comment nodes:      " << commentCount << "\n";
        std::cout << "Declaration nodes:  " << declarationCount << "\n";
        std::cout << "Unknown nodes:      " << unknownCount << "\n";
    }
private:
    int elementCount;
    int textCount;
    int commentCount;
    int declarationCount;
    int unknownCount;
};
```

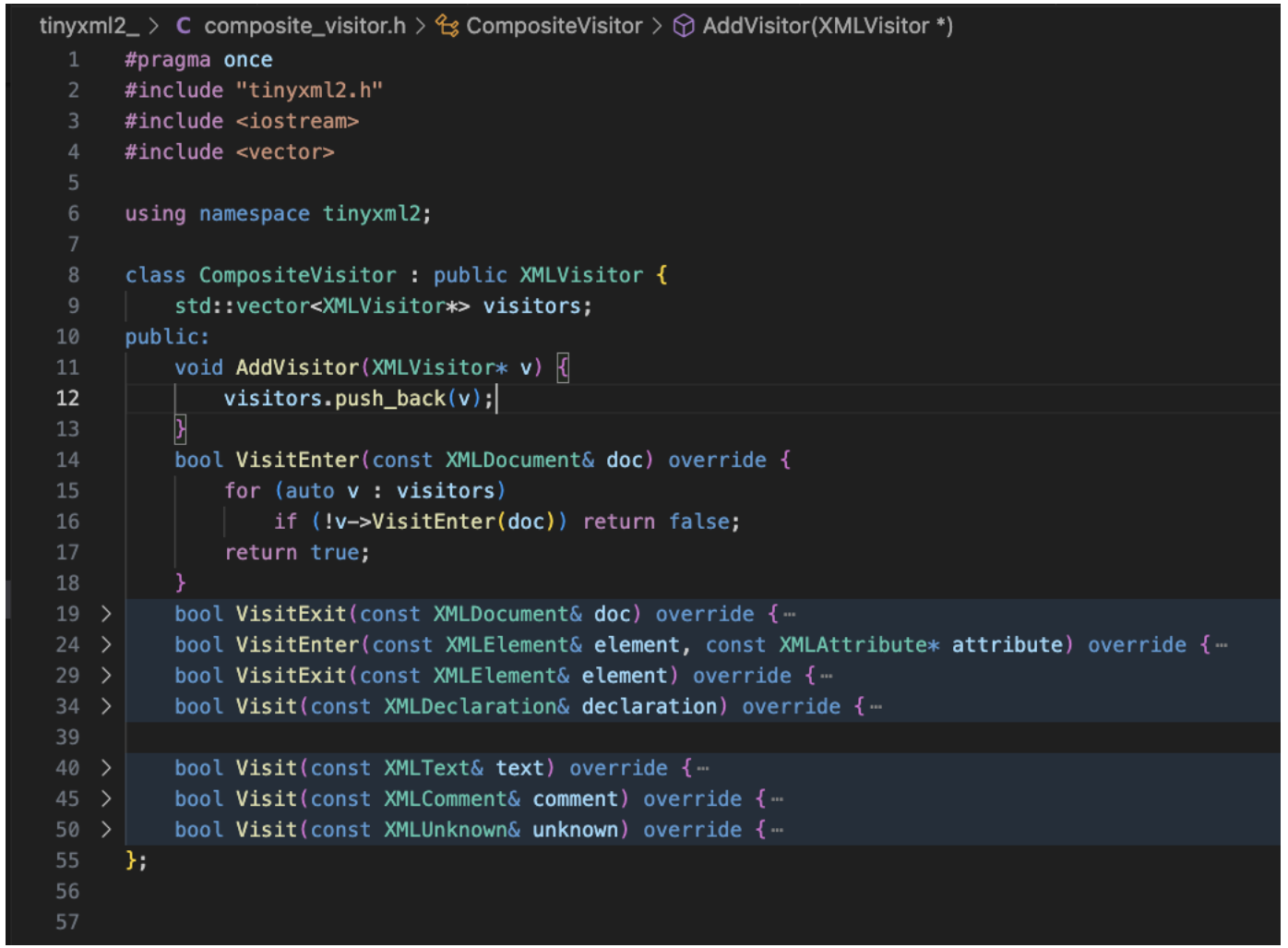
三、具体功能测试与拓展
（1）、访问者模式

访问者模式在拓展时的局限性：

1. 多态性较差：你没法通过 XMLVisitor* 去调用自定义访问者类中的额外功能（如 PrintCount()），除非你做额外类型转换（dynamic_cast<XMLCounter*>），这破坏了接口的封装性和多态性；或者在基类增加虚接口，这会导致基类接口膨胀，并且破坏程序设计的开放封闭原则。
2. 无法实现多操作的动态组合，XML结构一次只能Accept() 一个访问者。若要实现多个访问操作，只能对所有节点进行多次遍历，在处理大型XML文件时容易造成较大的性能损耗。

三、具体功能测试与拓展
（1）、访问者模式

改进——实现多操作动态组合：
1.通过构建XMLVisitor的派生类CompositeVistor，在该类中维护一个 std::vector<XMLVisitor*> visitors，可以随时添加不同访问操作，实现动态组合不同操作。
2.在访问节点时，依次调用所有添加的访问者所对应的方法，任一返回false就终止遍历，符合对节点的访问逻辑。



三、具体功能测试与拓展
（1）、访问者模式

测试——实现多操作动态组合：
利用原始项目提供的XMLPrint和自己拓展添加的XMLCounter构建一个CompositeVisitor，依次实现统计节点类型和打印XML文档的功能

样例输出：

```
XML node counts:
Element nodes:      10
Text nodes:         6
Comment nodes:      2
Declaration nodes:  1
Unknown nodes:      0

XML printed content:
<?xml version="1.0"?>
<!-- Sample XML for testing -->
<root>
    <book title="C++ Primer" year="2013">
        <author>Stanley B. Lippman</author>
        <price>39.99</price>
    </book>
    <book title="Effective C++" year="2005">
        <author>Scott Meyers</author>
        <price>29.99</price>
    </book>
    <!-- Another book entry -->
    <book title="Clean Code" year="2008">
        <author>Robert C. Martin</author>
        <price>34.95</price>
    </book>
</root>
```

tinyxml2_ > test_composite.cpp > main()

```cpp
#include"tinyxml2.h"
#include"tinyxml2_countnode.h"
#include"composite_visitor.h"
int main() {
    XMLDocument doc;
    if (doc.LoadFile("example.xml") != XML_SUCCESS) {
        std::cerr << "Failed to load XML file\n";
        return -1;
    }

    XMLCounter counter;
    XMLPrinter printer;

    CompositeVisitor composite;
    composite.AddVisitor(&counter);
    composite.AddVisitor(&printer);

    doc.Accept(&composite);

    std::cout << "XML node counts:\n";
    counter.PrintCount();

    std::cout << "\nXML printed content:\n";
    std::cout << printer.CStr() << std::endl;

    return 0;
}
```

三、具体功能测试与拓展

（2）、内存管理模式

核心代码：1.内存管理框架

```
template< size_t ITEM_SIZE >
class MemPoolT : public MemPool
{
public:
    MemPoolT() : _blockPtrs(), _root(0), _currentAllocs(0), _nAllocs(0), _maxAllocs(0), _nUntracked(0)  {}
    ~MemPoolT() {…

    void Clear() {…

    virtual size_t ItemSize() const override {…
    size_t CurrentAllocs() const {…

    virtual void* Alloc() override{…

    virtual void Free( void* mem ) override {…
    void Trace( const char* name ) {…

    void SetTracked() override {…

    size_t Untracked() const {…

    // This number is perf sensitive. 4k seems like a good tradeoff on my machine.…
    enum { ITEMS_PER_BLOCK = (4 * 1024) / ITEM_SIZE };

private:
    MemPoolT( const MemPoolT& ); // not supported
    void operator=( const MemPoolT& ); // not supported

    union Item {
        Item*   next;
        char    itemData[static_cast<size_t>(ITEM_SIZE)];
    };
    struct Block {
        Item items[ITEMS_PER_BLOCK];
    };
    DynArray< Block*, 10 > _blockPtrs;
    Item* _root;

    size_t _currentAllocs;
    size_t _nAllocs;
    size_t _maxAllocs;
    size_t _nUntracked;
};
```

2.内存管理关键环节——分配、释放、清空

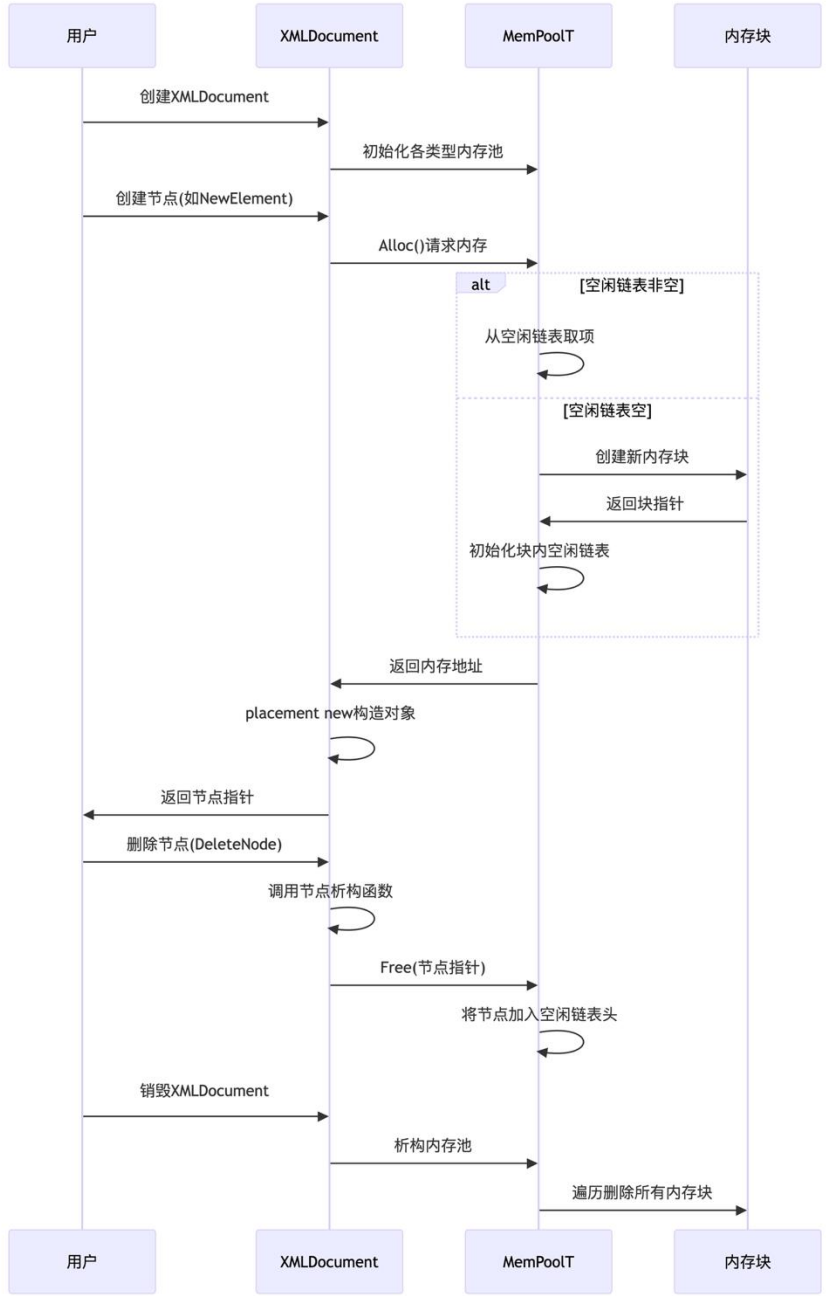
```
virtual void* Alloc() override{
    if ( !_root ) {
        // Need a new block.
        Block* block = new Block;
        _blockPtrs.Push( block );

        Item* blockItems = block->items;
        for( size_t i = 0; i < ITEMS_PER_BLOCK - 1; ++i ) {
            blockItems[i].next = &(blockItems[i + 1]);
        }
        blockItems[ITEMS_PER_BLOCK - 1].next = 0;
        _root = blockItems;
    }
    Item* const result = _root;
    TIXMLASSERT( result != 0 );
    _root = _root->next;

    ++_currentAllocs;
    if ( _currentAllocs > _maxAllocs ) {
        _maxAllocs = _currentAllocs;
    }
    ++_nAllocs;
    ++_nUntracked;
    return result;
}
```

```
void Clear() {
    // Delete the blocks.
    while( !_blockPtrs.Empty()) {
        Block* lastBlock = _blockPtrs.Pop();
        delete lastBlock;
    }
    _root = 0;
    _currentAllocs = 0;
    _nAllocs = 0;
    _maxAllocs = 0;
    _nUntracked = 0;
}
```

```
virtual void Free( void* mem ) override {
    if ( !mem ) {
        return;
    }
    --_currentAllocs;
    Item* item = static_cast<Item*>( mem );
```

三、具体功能测试与拓展
（2）、内存管理模式

工作流程：
1.初始化内存池：构造内存池对象，不立即分配任何内存。
2.首次分配：分配一个新的 Block，内部包含多个固定大小的内存单元（Item）；用每个 Item 的 next 指针把它们串成一个空闲链表（Free List）；空闲链表头 _root 指向第一个可用 Item。从空闲链表中取出一个 Item，将其从链表头移除；返回该 Item 的指针作为可用内存块；更新分配计数器。
3.释放过程：将释放的 Item 加回空闲链表的头部（即实现快速复用）。
4.再次分配：若空闲链表还有可用 Item，直接复用，无需分配；若 链表为空，则再次分配新的 Block，并重复上述步骤。
5.清空内存池：释放所有分配过的 Block；清空链表及状态信息。



三、具体功能测试与拓展
（2）、内存管理模式

优越性：

1.分配/释放对象的复杂度为o(1)，只需要更新一个链表头指针即可完成。

2.重构动态分配内存的模式，避免使用操作系统的new/delete，降低了分配内存的开销。

3.内存池可以精准控制内存使用量，通过一系列统计数据（如当前分配数、历史最大分配数、总分配次数等），能够跟踪内存的分配、释放。

4.内存池提供clear函数，集中清理内存，防止内存泄漏。

5.由于所有节点的生命周期和文档节点的生命周期一致，所以将内存池的分配绑定到XMLDocument节点的构造，通过XMLDocument节点实现对所有节点的内存管理

```
XMLDocument::XMLDocument( bool processEntities, Whitespace whitespaceMode ) :
    XMLNode( 0 ),
    _writeBOM( false ),
    _processEntities( processEntities ),
    _errorID(XML_SUCCESS),
    _whitespaceMode( whitespaceMode ),
    _errorStr(),
    _errorLineNum( 0 ),
    _charBuffer( 0 ),
    _parseCurLineNum( 0 ),
    _parsingDepth(0),
    _unlinked(),
    _elementPool(),
    _attributePool(),
    _textPool(),
    _commentPool()
{
    // avoid VC++ C4355 warning about 'this' in initializer list (C4355 is off by default in VS2012+)
    _document = this;
}
```

```
MemPoolT< sizeof(XMLElement) >    _elementPool;
MemPoolT< sizeof(XMLAttribute) > _attributePool;
MemPoolT< sizeof(XMLText) >      _textPool;
MemPoolT< sizeof(XMLComment) >   _commentPool;
```

清華大學
Tsinghua University

谢谢！