### ParserDom： 继承 ParserSax

成员变量

tree<Node>mHtmlTree;                   //数据成员，用于存放解析好的树tree<Node>::iteratormCurrentState;  //用于遍历树的迭代器，具体声明可见tree.h文件

成员函数

 public:

                   ParserDom(){}                     //构造方法

                   ~ParserDom(){}                 //析构方法

                    consttree<Node> &parseTree(const std::string &html);       //通过String字符串来解析树

                   consttree<Node> &getTree()                   //返回(解析好的)数据成员mHtmlTree

                   {return mHtmlTree; }

         protected:

                   //声明了一系列虚函数，用于之后的重载

                   virtualvoid beginParsing();      //开始解析

                   virtualvoid foundTag(Node node, bool isEnd);         //寻找指定标签

                   virtualvoid foundText(Node node); //寻找文本

                   virtualvoid foundComment(Node node);         //寻找注释文本

                   virtualvoid endParsing();         //结束解析

## 操作符重载

std::ostream&operator<<(std::ostream &stream, const tree<HTML::Node>&tr);

//重写了<<操作符，使得可以该操作符可以直接输出tree<HTML::Node>类型的变量

ParserSax：流文件输入，解析html，形成dom树，每个节点都是node结构

成员变量

unsignedlong mCurrentOffset;      //当前正在解析的偏移位置

constchar \*mpLiteral;    //具体作用需要到ParseSax.tcc中才显现出来

 boolmCdata;  //同上

成员函数

## 构造函数

ParserSax(): mpLiteral(0),mCdata(false) {}//构造方法，初始化两个成员变量的值

virtual ~ParserSax() {}

## 在ParseDom.cc中进行了实现的虚函数

foundTag(Node node, bool isEnd) {} foundText(Node node)

foundComment(Node node) {} endParsing() {}

## 函数模板

template<typename \_Iterator> void parse(\_Iterator &begin, \_Iterator &end,std::forward\_iterator\_tag);

template<typename \_Iterator> void parseHtmlTag(\_Iterator b, \_Iterator c);

template<typename \_Iterator> void parseContent(\_Iterator b, \_Iterator c);

template<typename \_Iterator> void parseComment(\_Iterator b, \_Iterator c);

template<typename \_Iterator>\_Iterator skipHtmlTag(\_Iterator ptr, \_Iterator end);

template<typename \_Iterator> \_Iterator skipHtmlComment(\_Iterator ptr, \_Iterator end);

Node ： 定义tree节点的结构

成员变量

std::stringmText;    //节点的内部文本

std::stringmClosingText;

unsignedint mOffset;      //节点在原文档中的偏移值

unsignedint mLength;     //节点的字符长度的

std::stringmTagName;   //节点的Tag名

std::map<std::string,std::string> mAttributes; //存放属性的二维数组

boolmIsHtmlTag;    //节点是否是HtmlTag标签

boolmComment;     //节点是否是注释

成员函数

## 各个数据成员的简单的赋值取值函数

## 操作符重载

bool Node::operator==(const Node &n)const //判断两个Node是否都为tag标签，且tag名是否相等

 Node::operator string() const        //根据是否是tag标签，返回tagName或者text

ostream &Node::operator<<(ostream&stream) const //按对象输出Node

## 标签属性的解析

**parseAttributes()**

函数基本流程图

parseContent

foundText

parsecomment

foundcomment

ParserDom::parseTree(html)

ParserSax::parse(conststd::string&html)

ParserSax::parse(\_Iteratorbegin,\_Iteratorend)

ParserDom:: getTree()

ParserSax::parse(\_Iterator&begin,\_Iterator&end,std::forward\_iterator\_tag)

ParserDom::beginParsing() //初始化

开始解析“<”

Isalpa tag

/ comment

! comment

? % comment

中间段content的处理

skipHtmlTag

parseHtmlTag

foundTag

flatten

Cout<<dom<<endl;//输出所有超链接节点

依次输出每个节点及其属性 parseAttributes()

ParserSax

ParserDom

Class Node . parseAttributes()

解析已被标记为tag的“< >”尖括号之间内容

<td name=”class”> key : name val :class

解析出第一段不为空的字符为tagname 继续向后找一段连续的字符（=之前结束）作为key

等号之后的连续字符作为val ，遇到空格开始下一对属性的解析