Документация
Тема 9: XML парсер

Глава 1. Увод

Целта на проекта е програма, реализираща четене и операции с XML файлове. Програмата съхранява и обработва XML елементите като работата с програмата се осъществява в диалогов режим.

От така формулираната цел произлизат следните задачи:

- Дефиниране на изисквания към системата.
- Конструиране и реализация на представянето на атрибутите на XML елементите.
- Конструиране и реализация на представянето на самите XML елементи.
- Конструиране и реализация на влагане на XML елементи.
- Реализация на общите операциите за работа с командния ред.
- Реализация на операциите за четене и обработка на XML елементи от файл.

Те ще бъдат разгледани в:

- Глава 2. Преглед на предметната област
- Глава 3. Проектиране
- Глава 4. Реализация

Глава 2. Преглед на предметната област

Системата трябва да поддържа операциите:

- Извеждане на екрана прочетената информация от XML файл.
- Извеждане стойност на атрибут по даден идентификатор на елемента и ключ на атрибута.
- Присвояване на стойност на атрибут.
- Извеждане на списък с атрибути на вложените елементи.
- Достъп до n-тия наследник на елемент.
- Достъп до текста на елемент.
- Изтриване на атрибут на елемент по ключ.
- Добавяне на нов наследник на елемент.

Глава 3. Проектиране

Проектът включва класовете:

- Array – конструиране и реализация на саморазширяващ се масив от указатели към обекти на потребителски дефиниран тип.
- Attribute – конструиране и реализация на представянето на атрибутите на XML елементите.
- Element – конструиране и реализация на представянето на самите XML елементи.
- Helper – помощен клас, съдържащ някои статични функции.
- MyException – клас, наследник на std::exception.

- CommandParser – клас, отговарящ за разбиването на въведените от потребителя операции на отделни аргументи.
- CommandExecutor – клас, реализиращ общите операции за работа с командния ред и операциите за четене и обработка на XML елементи от файл.
- Parser – клас, съдържащ метод, който поддържа диалоговия режим.

Глава 4. Реализация

- Клас **Array**

Шаблон на клас, реализиращ саморазширяващ се масив от указатели към обекти от даден тип, поддържащ дефинираните операции в класа. Член-данните са със спецификатор за достъп private (няма пряк външен достъп до тях):

`T** elements;` - масив от указатели към обекти от тип T (динамично заделен).

`size_t size;` - брой добавени указатели.

`size_t capacity;` - капацитет на масива elements.

Две статични константи:

`static const size_t INITIAL_CAPACITY = 2;` - начален капацитет на масива от указатели.

`static const size_t INCREASE_STEP = 2;` - при преоразмеряване удвояваме капацитета на масива от указатели.

Обектите на класа Array се създават посредством конструктор по подразбиране. Заради динамично заделената член-данна elements, за коректната обработка на паметта, са реализирани конструктор за копиране, деструктор и операция =.

Член-функции със спецификатор за достъп private (няма пряк външен достъп до тях):

`void allocate(const size_t size);` - помощна функция, заделя памет за size на брой указателя към обекти от тип T.

`void copy(const Array& other);` - помощна функция за копиране на обект на класа Array в текущия обект.

`void deallocate();` - помощна функция за освобождаване на заделена памет за текущия обект.

`void resize();` - помощна функция за преоразмеряване на член-данната elements.

Член-функции със спецификатор за достъп public:

`size_t getSize() const;` - селектор за броя елементи на член-данната elements.

`void add(const T& element);` - добавя указател към елемента element към член-данната elements.

`void deleteAt(int index);` - изтрива елемента на позиция index.

Предефинирани са операции за индексиране:

```
T& operator [] (int index);
const T& operator [] (int index) const;
```

Предефинирана е операция за изход (<<) като приятелска функция:

```
template <typename U>
friend std::ostream& operator << (std::ostream& out, const Array<U>& array);
```

- Клас **Attribute**

Член-данните са със спецификатор за достъп private (няма пряк външен достъп до тях):

`static const size_t MAX_LEN = 255;` - статична константа за максималната дължина на ключа и неговата стойност.

`char* key;` - ключ на атрибута (динамично заделена член-данна).

`char* value;` - стойност на ключа на атрибута (динамично заделена член-данна).

Обектите на класа Attribute се създават посредством конструктор по подразбиране и конструктор с параметри. Заради динамично заделените член-данни key и value, за коректната обработка на паметта, са реализирани конструктор за копиране, деструктор и операция =.

Член-функции със спецификатор за достъп public:

`const char* getKey() const;` - селектор за ключа на атрибута.

`const char* getValue() const;` - селектор за стойността на ключа на атрибута.

`void setKey(const char* key);` - мутатор за ключа на атрибута.

`void setValue(const char* value);` - мутатор за стойността на ключа на атрибута.

Предефинирани са операции за вход (>>) и изход (<<) като приятелски функции:

```
friend std::ostream& operator << (std::ostream& out, const Attribute& attribute)
friend std::istream& operator >> (std::istream& in, Attribute& attribute);
```

- Клас **Element**

Член-данните са със спецификатор за достъп private (няма пряк външен достъп до тях):

`static const size_t MAX_LEN = 255;` - максимална дължина на член-данната label.

`static const size_t MAX_TEXT_LEN = 1000;` - максимална дължина на член-данната text.

`static unsigned int nextId;` - статична променлива, пазеща стойността на следващия генериран от програмата идентификатор, за елементите без такъв във входния файл.

`char* label;` - етикет на елемента.

`Attribute id;` - идентификатор на елемента.

`Array<Attribute> attributes;` - списък от атрибути на елемента.

`Array<Element> nestedElements;` - списък от вложени елементи.

`unsigned int level;` - ниво на влагане на елемента.

`char* text;` - текст на елемента.

Член-функции със спецификатор за достъп private:

`void copy(const Element& other);` - помощна функция за копиране на обект на класа Element в текущия обект.

`void deallocate();`- помощна функция за освобождаване на заделена памет за текущия обект.

`void setLabel(const char* label);` - мутатор за етикета на елемента.

`void setId();`- мутатор за идентификатора на елементите, за които той се генерира от програмата.

`void addSuffix(const char* suffix);` - мутатор за добавяне на суфикс на идентификатора на елемента, извикал функцията, за постигане на уникалност на идентификаторите.

`void processDuplicateIds();` - търси равни идентификатори и добавя суфикс към тях за постигане на уникалност.

`void setText(const char* text);` - мутатор за текста на елемента.

`int functionHelper(const Element* element, const char* id, const char* key) const;` - помощна функция за намиране на индекса на атрибут с ключ key в елемента element, чийто идентификатор има стойност id. Ако търсенето е неуспешно, се връща -1.

`bool getChildrenAttributesHelper(const Element* element, const char* id, Array<Attribute>& array) const;` - помощна функция, добавяща атрибутите на вложените елементи (на елемента element, чийто идентификатор има стойност id), които са различни от идентификатори, в аргумента array. Връща булев резултат дали такива са намерени.

Обектите на класа Element се създават посредством конструктор по подразбиране. Заради динамично заделената член-данни label и text, за коректната обработка на паметта, са реализирани конструктор за копиране, деструктор и операция =.

Член-функции със спецификатор за достъп public:

`const char* getAttributeValue(const char* id, const char* key) const;` - връща стойността на атрибут по даден идентификатор на елемента и ключ на атрибута. При неуспех се връща nullptr.

`bool setAttributeValue(const char* id, const char* key, const char* value);` - променя стойността на атрибут по даден идентификатор на елемента, ключ на атрибута и нова стойност. Връща се булев резултат дали промяната е успешна.

`bool deleteAttribute(const char* id, const char* key);` - изтрива атрибута на елемента по даден идентификатор на елемента и ключ на атрибута. Връща се булев резултат дали операцията е успешна.

`bool addNestedElement(const char* id);` - добавя нов наследник на елемент, по даден идентификатор на елемента. Връща се булев резултат дали операцията е успешна.

`Array<Attribute> getChildrenAttributes(const char* id) const;` - връща списък с атрибутите на вложените елементи.

`const Element getChild(const char* id, unsigned int n, bool& isFound) const;` - връща копие на n-тото дете (n > 0) на елемент, чийто идентификатор има стойност id. В isFound се пази дали е намерено това дете.

`const char* getText(const char* id) const;` - селектор за текста на елемент, чийто идентификатор има стойност id. Ако няма елемент с този идентификатор или той няма текст, се връща nullptr.

Предефинирани са операции за вход (>>) и изход (<<) като приятелски функции:

```
friend std::ostream& operator << (std::ostream& out, const Element& element);
```

```
friend std::istream& operator >> (std::istream& in, Element& element);
```

- Клас **CommandParser**

Член-данните са със спецификатор за достъп private (няма пряк външен достъп до тях):

`static const size_t MAX_NUMBER_OF_ARGS = 10;` - максимален брой аргументи, включително самата операция.

`static const size_t MAX_LEN = 255;` - максимална дължина на аргументите и операцията.

`char** arguments;` - динамично заделен масив от символни низове, пазещ аргументите на операцията и самата нея.

`size_t numberOfArgs;` - пази броят на аргументите на операцията, включително самата операция.

Член-функции със спецификатор за достъп private:

`void allocate();` - заделя паметта за член-данната arguments.

`void deallocate();` - освобождава заделената памет.

`void processTheLine(const char* line);` - валидира и разделя операцията на аргументи.

Обектите на класа CommandParser се създават посредством конструктор с параметри. Заради динамично заделената член-данна arguments, за коректната обработка на паметта, са реализирани конструктор за копиране, деструктор и операция =.

Член-функции със спецификатор за достъп public:

`size_t getNumberOfArguments() const;` - връща броя на аргументите на операцията.

Предефинирана е операция за индексиране:

```
const char* operator [] (int index) const;
```

- Клас **CommandExecutor**

Реализиран на принципа на обектно-ориентирания шаблон за дизайн Singleton, притежава една инстанция и не могат да се създават обекти от него, както и да се копират такива, затова конструкторът за копиране и операцията за присвояване са изтрити.

Член-данните са със спецификатор за достъп private (няма пряк външен достъп до тях):

`static const size_t MAX_LEN = 255;` - максимална дължина на името на файла.

`Element element;` - композиция на обект от класа Element, в който четем съдържанието на файла.

`char fileName[MAX_LEN];` - пази името на входния файл.

`bool isSaved;` - показва дали са запазени всички направени промени по входния файл.

`bool isFirstCommand;` - показва дали е отворен файл, за да могат да се изпълнят другите операции.

Член-функции със спецификатор за достъп private:

`void open(const char* fileName);` - зарежда съдържанието на даден файл. Ако такъв не съществува, се създава нов с празно съдържание.

`void close();` - затваря текущо отворения файл.

`void save();` - записва направените промени обратно в същия файл, от който са били прочетени данните.

`void saveAs(const char* fileName);` - записва направените промени във файл, като позволява на потребителя да укаже неговия път.

`static void help();` - Извежда кратка информация за поддържаните от програмата команди.

`void exit(const CommandParser& parser);` - излиза от програмата.

`void print() const;` - извежда на екрана прочетената информация от XML файла.

`void select(const char* id, const char* key) const;` - извежда стойност на атрибут по даден идентификатор на елемента и ключ на атрибута.

`void set(const char* id, const char* key, const char* value);` - присвояване на стойност на атрибут.

`void children(const char* id) const;` - списък с атрибути на вложените елементи.

`void child(const char* id, unsigned int n) const;` - достъп до n-тия наследник (дете) на елемент.

`void text(const char* id) const;` - достъп до текста на елемент.

`void deleteAttribute(const char* id, const char* key);` - изтриване на атрибут на елемент по ключ.

`void newchild(const char* id);` - добавяне на нов наследник на елемент. Новият елемент няма никакви атрибути, освен идентификатор.

`bool isValidCommand(const CommandParser& parser) const;` - проверява дали въведената команда е в списъка на поддържани операции и дали броят подадени аргументи е правилен.

Член-функции със спецификатор за достъп public:

`bool execute(const CommandParser& parser);` - делегира изпълнението на въведената от потребителя команда. Връща булева стойност дали е извикан exit, тоест дали програмата трябва да завърши.

- Клас **Parser**

Съдържа статичен метод със спецификатор на достъп public, грижещ се за четенето на командите и операциите, делегира тяхното разбиване на аргументи и изпълнение, посредством класовете CommandParser и CommandExecutor:

`static void useParser();`

Глава 5. Заключение

Разработената програма реализира четене и операции с файлове, имащи основните характеристики на XML. Работата с програмата се осъществява в диалогов режим. Подобрението би включвало реализация на допълнителна функционалност, като заявки,

имащи основните характеристики на XPath. Вероятно може да се постигне подобрение в архитектурата.

Изготвил: Николета Тонева Щерева