УДК 004.423, 004.424, 004.428

ПРОГРАММА ПЕРЕНОСА ДАННЫХ ИЗ EXCEL-ТАБЛИЦ В ПРОГРАММУ ИХ ОБРАБОТКИ

Шумилов В.Н., Андрианова Е.Г.

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский технологический университет» (МИРЭА), 119454, Россия, г. Москва, проспект Вернадского, 78, e-mail: vasia.shumilov@gmail.com, andrianova@mirea.ru

При обработке различных опросных данных часто возникают технические трудности их переноса из опросных форматов (например, excel-таблиц) в данные программы, непосредственно занимающейся их математической обработкой. Рассмотрен случай, когда надо интегрировать средства программы обсчета данных, реализованной на C++ и программы выгрузки этих данных из excel-таблиц, реализованной на C#. Описана последовательность действий по программной интеграции объектно-ориентированных библиотек. Приведены фрагменты программного кода реализации указанной последовательности действий.

Ключевые слова: excel-таблица, перенос данных, интеграция программных библиотек, C++, C #, класс, библиотека, интерфейс библиотеки, XlsxReader, ReaderFacede, шаблон

DATA TRANSFER SOFTWARE EXCEL-TABLES TO THE SOFTWARE OF THEIR PROCESSING

Shumilov V.N., Andrianova E.G.

Federal State Educational Institution of Higher Education "Moscow Technological University" (MIREA), 119454, Russia, Moscow, Vernadsky avenue, 78, e-mail: vasia.shumilov@gmail.com, andrianova@mirea.ru

When processing different survey data, there are often technical difficulties in transferring them from questionnaire formats (for example, excel-tables) to program data directly involved in their mathematical processing. The case when it is necessary to integrate the means of the program for calculating data, implemented in C ++, and the program for downloading this data from excel-tables, implemented in C #, is considered. A sequence of actions for the program integration of object-oriented libraries is described. The fragments of the program code for the implementation of the specified sequence of actions are given.

Key words: excel-table, data transfer, integration of program libraries, C ++, C #, class, library, library interface, XlsxReader, ReaderFacede, template.

1. Введение

Задача интеграции программных библиотек, написанных на концептуально отличных языках программирования, является востребованной задачей. В данном случае описан процесс переноса данных, собранных в процессе социологических опросов в excel-таблицы в данные программы, реализованной на C#. Т.е. берутся данные из файла табличного редактора программой реализованной на C++ и переносятся в данные программы, реализованной на C#. Подобные ситуации часто встречаются в случаях, когда данные собираются прикладными исследователями, которым для первичной обработки достаточно построения графика наиболее доступными программными средствами. Для более сложной обработки, например, при применении математического аппарата вейвлет-анализа, R/S анализа, почти-периодических функций и т.п. возникает задача сохранения результатов математических операций в некоторой форме (например, векторном описании), удобной для математической обработки и последующей визуализации промежуточных и итоговых результатов.

Начинать интеграцию данных подобных программных средств надо с выделения и реализации интерфейсов, по возможности, максимально упрощённых интерфейсов. Здесь логично использовать структурный шаблон проектирования, позволяющий скрыть сложность системы путём сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы.

Программное обеспечение, написанное на C# компилируется в управляемый код. Управляемый код - термин, введённый фирмой Microsoft, для обозначения кода программы, исполняемой под «управлением» виртуальной машины .NET— Common Language Runtime или Mono, т.е. выполняющимся под управлением

CLR. При этом машинный код, в который компилируется «классический» C++, является, соответственно, неуправляемым, т.е. выполняющимся вне среды выполнения CLR. Существует C++/CLI — язык для среды программирования Microsoft .NET, «гибрид C++ и C#», но, отзывы по применению этого языка не всегда положительны, поэтому для данного проекта этот язык программирования использован не был. Следовательно, для создания требуемого программного обеспечения необходимо разработать решение (сценарий использования библиотек), которое позволит «смешать» управляемый и неуправляемый коды в одном программном приложении.

2. Решение в общем виде

Реализация операций по работе с excel-файлами спрятана в библиотеках. Пользователю обе библиотеки доступны. В первой библиотеке код, выполняющий чтение из файла и всю необходимую обработку, а во второй библиотеке - код, который необходим для взаимодействия со внешним клиентом. Значит, вторая библиотека должна иметь функции по выполнению операций перевода всех недопустимых для интеракции (взаимодействия, обмена данными) типов в допустимые. А первой библиотеке хорошо было бы сохранить совместимость с клиентами C++ в доступном для понимания виде, не заставляя программистов глубоко изучать код данной библиотеки. Также для клиента должны быть исключены все внешние зависимости, которые содержат в себе эти библиотеки. На рис. 1 представлена схема классов библиотеки XlsxReader. Стрелки показывают, что один класс унаследован от другого. Другие взаимосвязи между классами также есть, но в данном случае мы их не рассматриваем.

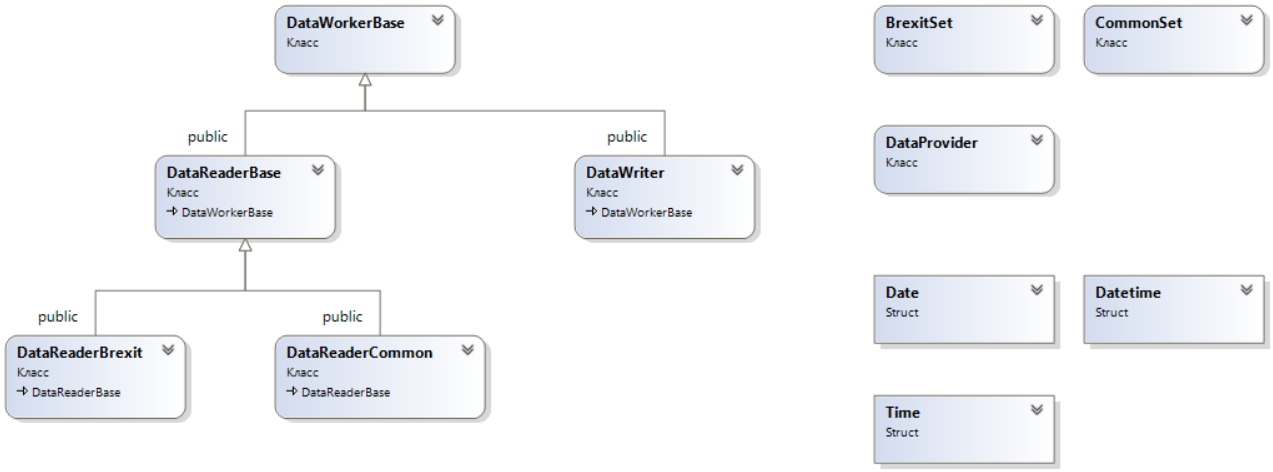

Рис. 1. Схема классов

Левая часть схемы классов (рис. 1) — это иерархия классов, выполняющих чтение и запись в файлы. Назначение классов структур справа на схеме (BrexitSet, CommonSet и DataProvider, Datetime, Date и Time) будет рассмотрено ниже. Это сделано потому, что данные классы и структуры не являются обязательными, например, можно успешно писать и читать файлы xlsx без использования класса DataProvider. Но без использования данного класса пользователю пришлось бы помимо библиотеки XlsxReader подключить ещё и библиотеку xlnt, которая выполняет непосредственно работу с файлами. Пользователь не обязан знать детали имплементации, и тем более разбираться в них. Это важный принцип. Наличие и применение класса DataProvider при работе с файлами делает библиотеку XlsxReader завершённой. Класс DataProvider выступает как некоторая прослойка между внешним и внутренним кодом, связывающая их в единое целое. Вся реализация операций чтения и записи с файлами ложится на классы DataReaderBrexit, DataReaderCommon и DataWriter.

3. Библиотека ReaderFacade

Библиотека ReaderFacade являет собой прослойку, необходимую для совмещения не самых совместимых языков программирования. Поэтому сама по себе эта библиотека ничего не делает, она только ссылается на библиотеку XlsxReader, которая выполняет основные операции по передаче данных.

Пользователю предлагается только интерфейс библиотеки, а имплементация скрыта, т.е. библиотека ReaderFacade это прослойка над классом DataProvider.

Необходимость её использования обусловлена тем, что она является шаблоном, через который могут быть внедрены DataProvider, ReaderFacade и другие классы, написанные с использованием C++, а вызывающая программа — с использованием C#.

Передача значений между такими разными программными средами напрямую возможна только для примитивных типов и указателей.

3.1 Обзор типов

Рассмотрим основные типы данных библиотеки ReaderFacade. Тип BrexitSetF сходен с типом BrexitSet (почти идентичен ему). Но указатели требуют особого обращения. Выделение памяти под поля company и method, а также их копирование для хранения в классе, происходит в конструкторе.

```
class BrexitSetF
{
public:
   /* Детали определения */

   Date date;
   double leave;
   double remain;
   double dontKnow;
   char* company;
   char* method;
};
using TBrexitArr = BrexitSetF * ;
```

Для поддержания однородности кода введён отдельный тип CommonSetF, определение которого полностью идентично с типом CommonSet, но позволяет создавать программные приложения независимо от реализации библиотеки XlsxReader, которой принадлежит тип CommonSet.

3.2 Класс ReaderFacade

Класс ReaderFacade является интерфейсом одноимённой библиотеки. Рассмотрим интерфейсную часть этого класса, которая очень похожа на интерфейсную часть класса DataProvider.

```
class ReaderFacade
{
public:
   static void startup();

   static void loadObamaRomneyList(const char* filename);
   static void loadObamaMcCainList(const char* filename);
   static void loadBrexitPollingList(const char* filename);
   static void loadConvertList(const char* filename);

   static void store(const TBrexitArr& list, size_t len, const char* filename);
   static void store(const TCommonArr& list, size_t len, const char* filename);

   static TCommonArr getObamaRomneyList(size_t& len);
   static TCommonArr getObamaMcCainList(size_t& len);
   static TBrexitArr getBrexitPollingList(size_t& len);
   static TCommonArr getConvertList(size_t& len);

   static void setObamaRomneyIndexes(size_t index1, size_t index2, size_t index3);
   static void setObamaMcCainIndexes(size_t index1, size_t index2, size_t index3);
   static void setBrexitPollingIndexes(size_t index1);
   static void setConvertIndexes(size_t index1, size_t index2, size_t index3);
};
```

Отличие между ними заключается в используемых типах и появлении метода startup. Этот метод создаёт новый DataProvider внутри структуры Impl в ReaderFacade. Вместо сложных типов из библиотеки STL C++

использованы встроенные (указатели, массивы). Нельзя просто добавить в область видимости типы excel файлов, которые содержат интересующие нас данные, необходимо добавить новые методы, что, конечно, ухудшает читаемость кода, но зато делает библиотеку более удобной для использования извне.

3.3 Детали внутренней реализации

Из-за того, что DataProvider принимает пути к файлам типа std::wstring (wide string) необходимо выполнить преобразование исходного типа char* к данному. Тип std::wstring является на самом деле basic_string<wchar_t>, а string — basic_string<char>. Таким образом, суть преобразования сводится к переводу строки char* в строку wchar_t*. Это чисто технический момент, описание которого в этом тексте не представлено. Важно знать, что строки не преобразуются сами собой. Ознакомиться с подробностями реализации всегда можно в исходном коде.

Рассмотрим функцию получения статистических данных из DataProvider и перевода их в соответствующий формат.

```
template <typename T, typename F>
T* Impl::getList(F&& func, size_t& len)
{
   auto list = (dataProvider.get()->*func)();

   auto listLen = list.size();

   auto arr = new T[listLen];

   std::transform(list.begin(), list.end(), stdext::make_checked_array_iterator(arr, listLen), [](auto& set)
   {
         return Impl::convert(set);
   });

   len = listLen;

   return arr;
}
```

Ссылка на нестатический метод не может быть использована без связи с любым экземпляром класса, на метод которого она ссылается. Отсюда синтаксис вызова: (dataProvider.get()->*func)().

Происходит переход от «умного» указателя к «сырому» путём вызова метода get() класса std::unique_ptr<DataProvider>. Параметр func передаётся как универсальная ссылка. Параметр шаблона T — это тип хранения одной строки таблицы excel файла (BrexitSetF или CommonSetF в контексте ReaderFacade).

Преобразование «сырого» массива (TBrexitArr или TCommonArr) в std::vector (TBrexitList или TCommonList) происходит следующим образом:

```
template <typename R, typename T>
void Impl::store(T&& list, const size_t len, const char* filename)
{
   std::vector<R> newList;

   std::transform(list, list + len, std::back_inserter(newList), [](auto& set) {
         return Impl::convert(set);
   });

   dataProvider->store(newList, convert(filename));
}
```

Параметры: универсальная ссылка на список, его длина и путь к файлу для сохранения. В параметрах шаблона R — BrexitSet или CommonSet. T — тип «сырого» списка. Использование back_inserter вместо обычного итератора обусловлено тем, что BrexitSet и CommonSet не имеют конструктора по умолчанию.

4. Библиотека XlsxReader

4.1 Класс DataProvider

Данные excel-таблиц копируются в данные программы в формате, определяемом классами BrexitSet и CommonSet. Эти классы представляют собой совокупность информации из одной конкретной строки таблицы excel-файла. Псевдонимы TBrexitList и TCommonList представляют типы для хранения списков классов, упомянутых выше. Контейнером для хранения данных выбран std::vector. Экспортируемым классом является DataProvider. При его проектировании была применена идиома Pimpl для сокрытия имплементации.

Такой подход позволил снизить зависимость внешнего кода от деталей определения класса. В качестве указателя на определение использован std::unique_ptr<Impl>, так как он предоставляет эксклюзивное владение ресурсом, на который указывает. Impl — тип структуры, в которой содержатся все необходимые поля класса. Определены копирующий и перемещающий конструкторы, а также копирующий и перемещающий операторы присваивания, потому что нет явной причины запрещать какие-либо из этих операций.

Определение перемещающих операций, генерируемое компилятором записаны как = default. Если бы это явное указание компилятору не было произведено, то они не были бы сгенерированы автоматически (стандарт C++, начиная с C++11). Спецификатор noexcept для перемещающих операций говорит о том, что они не генерируют исключений — это позволяет компилятору генерировать более оптимальный код.

DataProvider предназначен для работы с различными типами файлов формата xlsx. Чтобы прочитать какой-либо файл, необходимо воспользоваться методом load с подходящими параметрами. Стоит заметить, что функция может генерировать исключения из-за неправильных аргументов или каких-либо проблем в работе с файлом. И, конечно, так как методы не статичные, объект класса DataProvider должен быть создан перед использованием. Рассмотрим заголовок вышеупомянутого метода.

```
void load(docType type, const std::wstring& filename) const
```

Второй аргумент — это путь к файлу, с которым предстоит работать. А чтобы понять, что передавать в качестве первого аргумента, нужно посмотреть на определение его типа.

```
enum class doctype                    // это перечисление с областью видимости
{                                     // может принимать 3 значения
obamaRomney, obamaMcCain, brexitPolling, convert
};
```

Каждый элемент этого перечисления соответствует типу входного файла.

После успешной загрузки файла, можно воспользоваться одним из трёх методов для получения нужных данных. Лучше всего тем, что соответствует типу, указанному при вызове метода load. Ниже приведены заголовки этих методов.

```
TElectionList getObamaRomneyList() const;
TElectionList getObamaMcCainList() const;
TBrexitList getBrexitPollingList() const;
TCommonList getConvertList() const;
```

Результат можно сохранить в xlsx-файл, воспользовавшись методом store.

Метод имеет 2 перегрузки для типов TCommonList и TBrexitList. Первый аргумент — то, что нужно записать в файл, второй — путь к файлу. Так как форматирование файлов может различаться, предусмотрена возможность настройки класса, но это не значит, что форматирование может быть любым — получить результат возможно только если строки файла могут быть представлены как CommonSet или BrexitSet.

```
void store(const TCommonList& list, const std::wstring& filename) const;
void store(const TBrexitList& list, const std::wstring& filename) const;
```

Эти методы обеспечивают вышеупомянутую настройку. Первый параметр — тип файла, второй — индексы колонок с нужными данными в таблице excel. Для файла brexitPolling допустимо указание только одного индекса — индекса начала блока с нужными данными, то есть индекса той колонки, с которой начинается

последовательность колонок с нужными данными. Для остальных типов это индексы колонок, содержащих в себе «первое значение», «второе значение» и дату.

4.2 Классы BrexitSet и CommonSet

Для лучшего понимания работы программы ниже представлены краткие описания классов BrexitSet и CommonSet и типов TBrexitList и TCommonList.

```
class BrexitSet
{
public:
   /* Детали определения */

   Date date;
   double leave;
   double remain;
   double dontKnow;
   std::string company;
   std::string method;
};
using TBrexitList = std::vector<BrexitSet>;
class CommonSet
{
public:
   /* Детали определения */

   double value0;
   double value1;
   Datetime date;
};
using TCommonList = std::vector<CommonSet>;
```

В этом коде нет ничего, что заслуживало бы подробного рассмотрения, тем не менее, без ознакомления с этими типами, дальнейшее рассмотрение программы может быть затруднено. На этом моменте следует остановиться и отметить, что представленного до сих пор описания достаточно, чтобы работать с библиотекой. Далее в тексте будет рассмотрен уже не интерфейс, а её внутренняя реализация, от которой обычный пользователь изолирован.

4.3 Детали внутренней реализации

Начнём рассмотрение внутренней реализации со взгляда на класс DataReaderCommon. Реализация этого класса схожа с реализацией класса, и поэтому, последний класс DataReaderBrexit не будет рассмотрен.

```
class DataReaderCommon
   : public DataReaderBase
{
public:
   DataReaderCommon(std::size_t value0Index, std::size_t value1Index, std::size_t dateIndex);
   DataReaderCommon();
   void setIndexes(std::size_t value0Index, std::size_t value1Index, std::size_t dateIndex);
   TCommonList getCommonList();
private:
   std::size_t value0Index_{ 0 };
   std::size_t value1Index_{ 0 };
   std::size_t dateIndex_{ 0 };
};
```

Здесь всё выглядит прозрачно, и так оно и есть. Например, так выглядит обращение из DataProvider к этому классу.

```
TCommonList DataProvider::getConvertList() const
{
   return impl_->dataReaderConvert.getCommonList();
}
```

И вот так выглядит то, что происходит внутри.

```
TCommonList DataReaderCommon::getCommonList()
{
   if (ws_ == nullptr) throw std::exception("getCommonList: ws_ = nullptr");
   //Проверка, чтобы избежать непоправимых ошибок
   auto rows = ws_.rows(); //may be 'false' in param
//Получение «контейнера со строками таблицы excel», но это не контейнер, а имитация //для совместимости с функциями, принимающими итераторы.
   TCommonList list;

   for (auto& row : rows) //Итерация по всем строкам таблицы
   {
         const auto dt = row[dateIndex_].value<xlnt::datetime>();
         Datetime datetime{ dt.year, dt.month, dt.day, dt.hour, dt.minute, dt.second };

list.emplace_back(row[value0Index_].value<double>(),
row[value1Index_].value<double>(), std::move(datetime));
//Перенос значений в нужном формате из файла в программу
   }

   return list;
}
```

Рассмотрим DataWriter.

```
class DataWriter
   : public DataWorkerBase
{
public:
   DataWriter();
   void setup(const std::wstring& filename);
   void finish() const;

   template <typename T>
   void store(T&& list);
private:
   void writeSet(int j, int i, const BrexitSet& value);
   void writeSet(int j, int i, const CommonSet& value);
   std::wstring filename_;
};
```

Чтобы подготовиться к записи, нужно вызвать метод setup. Далее, чтобы записать информацию в виртуальный файл — store. И чтобы файл сохранить — finish. Метод writeSet обеспечивает перенос одной «строчки» (строки в таблице представлены классами BrexitSet и CommonSet) в нужные конкретные ячейки файла. Для более подробного ознакомления с реализацией библиотеки следует обратиться к исходному коду.

5. Заключение

В данной работе приведено программное решение, позволяющее использовать управляемый и неуправляемый коды в одном программном приложении на примере программного обеспечения переноса данных, полученных в ходе социологического опроса, в данные программы их обработки. Было предложено решение, позволяющее связать библиотеку, выполняющую чтение данных из файла и всю необходимую обработку, и библиотеку - код, который необходим для взаимодействия со внешним клиентом. Все внешние зависимости, которые содержат в себе эти библиотеки, для клиента были скрыты. Обрабатываемые данные могут быть получены при исследовании процессов различной природы - экономических, социальных, физических. Использование C++ для реализации сложной математической обработки информации значительно превосходит по скорости решение на C# или на других высокоуровневых языках. Также многие математические, физические и прочие программные библиотеки не имеют интерфейса под C#, но имеют под C++ (как, например, OpenCV), а если и имеют, то реализацию на C++ выполнить проще, и работает, эта реализация опять же, быстрее. Следовательно, предложенное программное решение позволит быстро осуществлять сложную математическую обработку больших объёмов данных.

Список литературы

1. Scott Meyers Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14. // URL: http://www.amazon.com/Effective-Modern-Specific-Ways-Improve/dp/1491903996/ (Дата обращения: 03.05.2018)
2. Modern C++ Programming Cookbook: Recipes to explore data structure, multithreading, and networking in C++17 Paperback – May 15, 2017// URL: http:/www.amazon.com/Modern-Programming-Cookbook-multithreading-networking/dp/1786465183/ (Дата обращения: 03.05.2018)
3. Campbell Parallel Programming with Microsoft® Visual C++® / Campbell. - Москва: Гостехиздат, 2015- 784 с.
4. Понамарев, В. Программирование на C++/C# в Visual Studio .NET 2003 / В. Понамарев. - М.: БХВ-Петербург, 2015. - 917 с.
5. Герберт Шилдт C# 4.0: The Complete Reference Пер. с англ. - СПб: Питер, 2015. - 1056 с.
6. Лотка, Рокфорд C# и CSLA .NET Framework. Разработка бизнес-объектов / Рокфорд Лотка. - М.: Вильямс, 2010. - 816 с.
7. Рихтер, Джеффри CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Джеффри Рихтер. - М.: Питер, 2013. - 928 с.
8. Троелсен, Эндрю Язык программирования C# 5.0 и платформа .NET 4.5 / Эндрю Троелсен. - М.: Вильямс, 2015. - 486 с.
9. Фримен, Адам ASP.NET MVC 3 Framework с примерами на C# для профессионалов / Адам Фримен , Стивен Сандерсон. - М.: Вильямс, 2011. - 672 с.

References

1. Scott Meyers Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14. // URL: http://www.amazon.com/Effective-Modern-Specific-Ways-Improve/dp/1491903996/ (Data obrashcheniya: 03.05.2018)
2. Modern C++ Programming Cookbook: Recipes to explore data structure. multithreading. and networking in C++17 Paperback – May 15. 2017// URL: http:/www.amazon.com/Modern-Programming-Cookbook-multithreading-networking/dp/1786465183/ (Data obrashcheniya: 03.05.2018)
3. Campbell Parallel Programming with Microsoft® Visual C++® / Campbell. - Moskva: Gostekhizdat. 2015-784 c.
4. Ponamarev. V. Programmirovaniye na C++/C# v Visual Studio .NET 2003 / V. Ponamarev. - M.: BKhV-Peterburg. 2015. - 917 c.
5. Gerbert Shildt C# 4.0: The Complete Reference Per. s angl. - SPb: Piter. 2015. - 1056 s.
6. Lotka. Rokford C# i CSLA .NET Framework. Razrabotka biznes-obyektov / Rokford Lotka. - M.: Viliams. 2010. - 816 c.
7. Rikhter. Dzheffri CLR via C#. Programmirovaniye na platforme Microsoft .NET Framework 4.0 na yazyke C# / Dzheffri Rikhter. - M.: Piter. 2013. - 928 c.
8. Troyelsen. Endryu Yazyk programmirovaniya C# 5.0 i platforma .NET 4.5 / Endryu Troyelsen. - M.: Viliams. 2015. - 486 c.
9. Frimen. Adam ASP.NET MVC 3 Framework s primerami na C# dlya professionalov / Adam Frimen . Stiven Sanderson. - M.: Viliams. 2011. - 672 c.