УДК 004.04

НОВАЯ ПАРАДИГМА ОБРАБОТКИ АРГУМЕНТОВ КОМАНДНОЙ СТРОКИ

О.В. СУХОРУКОВ
(Представлено: канд. физ.-мат. наук, доц. О.В. ГОЛУБЕВА)

Описываются существующие методы обработки аргументов командной строки. Приводятся как их сильные, так и слабые стороны. Анализируя методы, выдвигаются два основных критерия, формирующих цель авторской разработки. Первый критерий – универсальность метода. Второй – создание максимального комфорта, нативно понятного интерфейса, быстрого старта при первом использовании не требующего глубокого изучения документации.

Универсальность авторского метода заключается в том, что за обработку опций отвечает отдельное приложение, принимающее в виде конфигурационного файла или строки структуру используемых опций, а также командную строку, требующую разбора. А после ее разбора и обработки выдает по требованию всю необходимую информацию.

Используя для декларации опций в конфигурационном файле сценарный язык программирования «JavaScript», автор добивается не только нативно понятного синтаксиса, но и тем самым увеличивает его гибкость и эффективность.

Технологии программирования не стоят на месте, эволюционируют сама парадигма программирования, определяющая не только совокупность идей и понятий, а также сам стиль написания компьютерных программ [1]. Выходя на более высокий уровень абстракций, код, по нашему мнению, становится более интуитивно понятным, читабельным. Развитие вычислительной техники все больше разворачивает стиль написания программ от кода, «как это будет удобно компилятору, процессору», до кода, «как это будет удобно человеку». Если рассматривать разработку библиотек, применяемых при создании программного обеспечения, во-первых, разработка ведется уже не только с учетом максимальной скорости работы ее методов, но и, во-вторых, с учетом создания максимального комфорта, быстрого старта при их использовании, нередко даже в ущерб первому.

Функции «getopt()» и «getopt_long()», являющиеся на сегодняшний день де-факто стандартным инструментом обработки аргументов командной строки в UNIX/Linux среде, были разработаны еще в 1980-х годах [2, с. 43]. Изначально функции создавались для языка «C», однако практически для любого языка на сегодняшний день существуют специальные библиотеки, реализующие их функционал или работающие на их базе. К примеру, «Argp», созданный для линейки языков «C» [3], «getopts» – для «bash» [4], «optparse» – для «Python» [5] и пр.

Функции «getopt()», «getopt_long()», а также их производные выглядят на сегодняшний день, на наш взгляд, крайне архаично. Для столь тривиальной, далеко не основной, но крайне важной задачи, как обработка аргументов командной строки [6], потребуется потратить, возможно, не один человеко-час, чтобы разобраться, как правильно ими пользоваться. При этом, по личному опыту, хорошим примером будет заполнение структур «Argp» [3], далеко не факт, что, единожды разобравшись, как это делается, вам не придется при следующем использовании разбираться снова.

Описанная проблема формирует очевидную цель – при разработке нового метода обработки аргументов командной строки приоритетной задачей является создание максимального комфорта, нативно понятного интерфейса, быстрого старта, при первом использовании не требующего глубокого изучения документации.

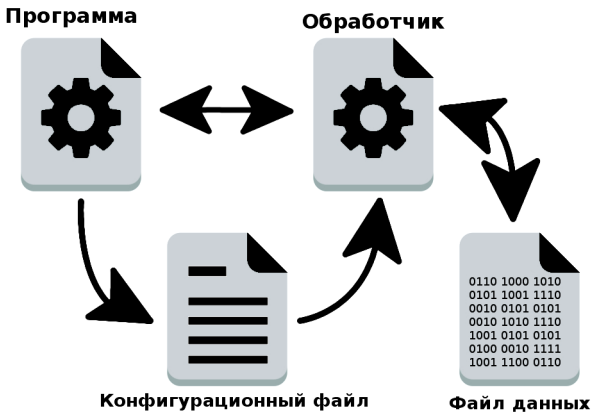
Не только нам интерфейс описанных методов казался излишне запутанным, устаревшим или имеющим недостаточный функционал. По этому поводу были написаны многочисленные статьи, не раз принимались и до сих пор принимаются в ряде случаев успешные попытки создать более простой интерфейс, при этом расширяющий функционал существующих методов. Хорошим примером будет служить библиотека «shflags», которая использует все ту же функцию «getopt()», но помимо парсинга опций умеет также и проверять их значения и даже самостоятельно обзывает переменные для опций согласно их длинному имени [7]. Подобные (удачные) аналоги стандартных функций можно отыскать практически в любом языке, к примеру, более продвинутая библиотека «argparse» для «Python» [8], заменяющая собой уже ставшую в ряде случаев стандартной, библиотеку «optparse» [5].

За кажущимся выходом из положения кроется не очевидная на первый взгляд проблема. Функции «getopt()» и «getopt_long()» были универсальными, так как были единственными, а значит не приходилось вникать в нюансы и отличия многочисленных методов. Сейчас же получается, что в каждом языке программирования, тем или иным способом устраняя недостатки первоначальных функций, имеется своя

реализация обработки аргументов командной строки, со своей библиотекой и набором методов, которая с каждой последующей версией все больше удаляется от исходных функций 80-х годов.

Автору, позиционирующему себя как системный программист-администратор, часто приходится писать небольшие вспомогательные программы, при этом быстро переключаясь с одного языка на другой. Это могут быть как всевозможные скрипты, написанные на одном из языков сценария, так и программы, написанные с использованием линейки языков «C». Обработка командной строки с учетом всех возможных соглашений «POSIX» (POSIX – Portable Operating System Interface) не может являться тривиальной задачей [6], что, в свою очередь, влияет на сложность применения всех известных автору методов обработки. Зачастую, по опыту автора, код обработки командной строки может оказаться длиннее и даже сложнее, чем код основной программы, к примеру, короткого скрипта. При этом постоянные переключения между методами обработки опций, разных языков программирования лишь усугубляют положение, дополнительно отнимая время и усилия.

Поэтому необходимо разработать, не только простой, но еще и обязательно универсальный метод обработки аргументов командной строки, не требующий переключения между языками. Это один из простых способов создания универсального метода, если за обработку опций будет отвечать отдельное приложение (рисунок). В этом случае «программа» передает «обработчику» в виде конфигурационного файла или строки структуру используемых опций, а также командную строку, требующую обработки. «Обработчик» разбирает командную строку согласно переданной структуре опций, сохраняет полученные данные во временный файл и завершает свою работу. По мере необходимости «программа» посылает запросы «обработчику», «обработчик» сверяется с файлом и отправляет «программе» соответствующий ответ.



Общая схема обработки

Схема с обработчиком в виде отдельной программы – цена, которую необходимо платить за универсальность. Такой способ нельзя назвать быстрым, однако обработка аргументов командной строки – это процедура, которая обычно и не требуется, – быстрых вычислений. Количество запросов можно сократить до минимума, вплоть до получения всех «формализованных» данных в ответ на передачу командной строки. Если скорость обмена информацией между «программой» и «обработчиком» важна, тогда по примеру баз данных вместо разовых вызовов достаточно открыть сессию, при которой «обработчик» будет завершать работу не после отработки очередного запроса, а после истечения определенного интервала времени. Если во время ожидания поступает новый запрос – интервал ожидания продлевается. При такой схеме работы «обработчику» не требуется при каждом запросе считывать или сбрасывать данные во временный файл, он сделает это только после истечения времени ожидания. Файл данных удаляется «обработчиком», по запросу, в конце жизненного цикла «программы».

Если за универсальность метода отвечает схема с обработчиком в виде отдельной программы, то за комфорт и нативно понятный интерфейс будет отвечать структура конфигурационного файла. На роль конфигурационного файла можно выбрать любой формат – от «YAML» [9], «XML» [10], «JSON» [11] до собственных разработок, однако, по нашему мнению, лучшим решением для формирования конфигурационного файла будет использование языка «JavaScript» [12].

Формат конфигурационного файла «YAML», к примеру, хотя и можно назвать «дружественным», однако нельзя назвать стандартным. Разделение синтаксических элементов производится посредством последовательности знаков тире завершающих специальным символом [9]. Это может оказаться непривычным для людей, не знакомых с форматом, а значит, может принести с собой дополнительные неудобства в работе.

Формат «XML» имеет избыточную разметку, за которой в итоге теряются основные данные [10].

«JSON», в отличие от «XML», имеет минималистскую разметку, не бросающуюся в глаза [11], а его расширенный формат – «JSON5» – поддерживает комментарии, имеет более упрощенный формат, допуская наличие мелких ошибок (к примеру, допускается наличие запятой после последнего элемента в объекте или списке) и пр. [13].

«JSON» основан на «JavaScript» [11], и его расширенный формат, в принципе, уже идеально подходит для использования в схеме «программа – обработчик». Однако используя вместо формата «JSON5», язык программирования, на котором он был основан, – «JavaScript» [12], мы не ограничиваем себя форматом, а получаем весь доступный функционал языка программирования, что открывает новые широкие возможности.

Использование сценарных языков программирования в конфигурационных файлах – метод не новый, но превосходно себя зарекомендовавший. Хорошим примером будет являться конфигурационный файл домена «XEN», являющийся, по сути, скриптом на языке «Python». А значит, в конфигурационном файле могут применяться любые конструкции этого языка, делая его гибким и эффективным. Например, можно объединить конфигурационные файлы нескольких машин в один, и для запуска любой из них использовать этот файл, с параметром, указывающим, домен какой машины должен стартовать и т.д. [14]. При стандартном использовании, в частности инициализации переменных, языки программирования по способу применения не отличаются от способа, применяемого в простых конфигурационных файлах, к примеру, в том же файле «INI», распространенном в «MS Windows» [15], а значит, не усложняют код.

Используя «JavaScript», «обработчик» в этом случае будет выступать в роли интерпретатора, производя построчный анализ, обработку и выполнение программы, описанную в конфигурационном файле. Так как код интерпретатора «JavaScript» в виде «обработчика» реализуется самостоятельно и полностью доступен, значит доступна и возможность дополнения, изменения среды, относящейся к языку программирования «JavaScript», согласно которому будет производиться построчный анализ и обработка выполнения программы. Согласно стандарту «POSIX» аргументы командной строки делятся на опции и аргументы (операнды). Аргументы, в свою очередь, делятся на свободные и зависимые (аргументы опций) [6]. Для реализации деления по типу и дальнейшего описания аргументов командной строки введем в среду языка программирования «JavaScript» два дополнительных конструктора для объекта опции и аргумента с именами «Option» и «Argument» соответственно. Конструкторы объектов будут являться частью языка и доступны для использования в конфигурационном файле наравне с другими конструкциями «JavaScript».

Конструктор объекта «Option» имеет свойство «keys», представляющее собой массив, в котором будут храниться строковые идентификаторы коротких и длинных имен, означающих эту опцию. К примеру, программой декларируется стандартная опция вывода справки, длинное имя которой согласно соглашениям «POSIX» будет «--help». Тогда для инициализации этой опции в конфигурационном файле (скрипте) необходимо ввести следующую команду:

```
var opt = new Option();
opt.keys = ["help"];
```

В первой строке производится декларация указателя «opt1» на объект и вызов конструктора. Во второй строке – инициализация массива. Конструктор объекта «Option» принимает элементы массива «keys» в виде аргументов, поэтому предыдущий код можно переписать в одну строку:

```
var opt = new Option("help");
```

Соглашением «POSIX» не дано определение короткого имени опции для вызова справки [16], однако часто в качестве такого имени используется символ «?» или «h». В этом случае команда декларации может иметь следующий вид:

```
var opt = new Option("help", "?", "h");
```

Пусть имя программы будет «program», тогда следующие команды при ее вызове будут интерпретироваться одинаково и после обработки вызывать справку:

```
program --help
program -?
program -h
```

Конструктор объекта «Argument» из основных свойств имеет имя объекта «objectName», и флаг «norequest» при значении «true» – указывающий, что аргумент является не обязательным. Имя объекта – это имя, которое будет выводиться в справке. Конструктор объекта принимает значения этих свойств в виде аргументов. Зависимость аргумента от опции определяется, тем остоятельством, является ли указатель на объект «Argument» свойством опции или нет.

К примеру, декларация независимой, обязательной опции «ARG» имеет следующий вид:

```
var arg = new Argument("ARG", false);
```

Декларация этого же аргумента, но в качестве зависимого, от вышеописанной опции, аргумента «opt», имеет следующий вид:

```
opt.arg = new Argument("ARG", false);
```

Это лишь, неполный пример синтаксиса, дополнительно введенных структур «JavaScript», реализующихся в «обработчике», так как формат данной статьи не позволяет раскрыть его полностью, однако должен хорошо демонстрировать принцип использования. Для декларации использующихся опций и аргументов достаточно создать соответствующие объекты и передать их декларацию в виде конфигурационного файла вместе с командной строкой «обработчику», который выполнит разбор командной строки и выдаст по требованию всю необходимую информацию. Распространенный и очень гибкий синтаксис «JavaScript» делает декларацию опций нативно понятной, а схема «программа – обработчик» делает метод универсальным.

ЛИТЕРАТУРА

1. Большая Энциклопедия Нефти Газа [Электронный ресурс] // Развитие – программирование : сайт. – Электрон. дан. – [Б. м.], 2016. – Режим доступа: http://www.ngpedia.ru/id360477p1.html. – Загл. с экрана.
2. Роббинс, А. Linux: программирование в примерах : пер. с англ. [Текст]: учеб.-метод. пособие / А. Роббинс. – М. : КУДИЦ-ОБРАЗ, 2005. – 656 с.
3. GNU Operating System [Электронный ресурс] // Parsing Program Options with Argp : сайт. – Электрон. текстов. дан. – [Б. м.], 2015. – Режим доступа: http://www.gnu.org/software/libc/ manual/html_node/Argp. html# Argp. – Загл. с экрана.
4. CIT Forum [Электронный ресурс] // getopts – разбор опций команды: сайт. – Электрон. текстов. дан. – [Б. м.], 2016. – Режим доступа: http://citforum.ru/operating_systems/manpages/GETOPTS.1.shtml. – Загл. с экрана.
5. Python [Электронный ресурс] // optparse – Parser for command line options: сайт. – Электрон. дан. – [Б. м.], 2016. – Режим доступа: https://docs.python.org/dev/library/optparse.html. – Загл. с экрана.
6. Сухоруков – соглашения «Posix» по обработке опций и аргументов командной строки.
7. Хабрахабр [Электронный ресурс] // Администрирование → bash скрипт с поддержкой длинных (gnu-style) опций : сайт. – Электрон. дан. – [Б. м.], 2016. – Режим доступа: https://habrahabr.ru/post/133860. – Загл. с экрана.
8. Python [Электронный ресурс] // Argparse Tutorial : сайт. – Электрон. дан. – [Б. м.], 2016. – Режим доступа: https://docs.python.org/3/howto/argparse.html. – Загл. с экрана.
9. The Offical YAML Web Site [Электронный ресурс] // YAML 1.2 : сайт. – Электрон. дан. – [Б. м.], 2016. – Режим доступа: http://yaml.org/. – Загл. с экрана.
10. XML.com [Электронный ресурс] // XML : сайт. – Электрон. дан. – [Б. м.], 2014. – Режим доступа: http://www.xml.com/. – Загл. с экрана.
11. JSON [Электронный ресурс] // Introducing JSON : сайт. – Электрон. дан. – [Б. м.], 2016. – Режим доступа: http://www.json.org/. – Загл. с экрана.
12. JavaScript [Электронный ресурс] // JavaScript: сайт. – Электрон. дан. – [Б. м.], 2016. – Режим доступа: http://www.javascript.com/. – Загл. с экрана.
13. JSON5 [Электронный ресурс] // JSON5: сайт. – Электрон. дан. – [Б. м.], 2016. – Режим доступа: http://www.json5.org/. – Загл. с экрана.
14. XGU [Электронный ресурс] // Конфигурационный файл XEN : сайт. – Электрон. дан. – [Б.м.], 2009. – Режим доступа: http://xgu.ru/wiki/Конфигурационный_файл_Xen. – Загл. с экрана.
15. Non GNU [Электронный ресурс] // INI formats: сайт. – Электрон. дан. – [Б. м.], 2016. – Режим доступа: http://www.nongnu.org/chmspec/latest/INI.html. – Загл. с экрана.
16. GNU Operating System [Электронный ресурс] // GNU Coding Standards: сайт. – Электрон. текстов. дан. – [Б. м.], 2015. – Режим доступа: http://www.gnu.org/prep/standards. – Загл. с экрана.