ritopt Tutorial

Damian Eads

October 20, 2015

Contents

1 Introduction

...

2 Option Conventions

Although, several command line option standards have been preposed, ritopt follows the conventions prescribed in the opt package. I will attempt, however, to outline them in this section.

2.1 Short Options

A short option usually begins with a dash character '--' followed by the short option name as shown in the following example.

```
 # A short option.
 myprogram -s
```

2.1.1 Invoking several short options

Several short options may be invoked simply by listing them after the dash character.

```
 # Several short options.
 myprogram -abcdef
 # This is equivalent.
 myprogram -a -b -c -d -e -f
```

Values may not be supplied when options are invoked as a short option list.

2.1.2 A mixture of short options and short option lists

Supplying short option lists and options with values is perfectly legal.

```
 myprogram -a "hello" -bcd -e=15
```

2.2 Long Options

Long options are preceded by two dash characters '----' as shown in the following example.

```
 myprogram --myoption
```

2.2.1 Long Option Lists

Unlike short options, long options cannot be listed at this time. However the feature is being considered for a future release using the following convention.

```
 myprogram --longoption1,longoption2,longoption3
```

2.3 Supplying Values

There are several ways to supply values to an option. ritopt supports three forms: assignment, open, and boolean forms.

2.3.1 Assignment Form

Place an equal sign '=' directly after the short or long option name followed by the value. The convention is shown below.

```
 myprogram --longoption=value -s=value
```

2.3.2 Open Form

The open form is where the value is supplied as a separate argument directly after the option as shown.

```
 myprogram --longoption value
```

2.3.3 Boolean Form

The boolean form is usually used exclusively for boolean options however will work for string options as well. Simply put a plus or minus sign directly after the short or long option. The convention is shown below.

```
 # Let's set a to true and b to false
 myprogram -a+ -b- --longa+ --longb-
```

2.4 String Values

To prevent parsing errors enclose string values in quotes.

```
# The following will work.
myprogram -s "Hello Sir," --long "How do you do?"


# The following will cause errors.
myprogram -s=Hello Sir, --long How do you do?
```

2.5 Boolean Values

Boolean options may be invoked in a non-boolean form using several values. The example below shows the values that may be used to represent true or false.

```
# Let's set each option to true.
myprogram -a true -b yes -c on -d activated -e active


# Let's set each option to false.
myprogram -a false -b no -c off -d "not activated" -e inactive
```

2.6 Array Values

The convention for option values of an array type is a comma delimited list as shown.

```
myprogram --myarrayoption=value1,value2
myprogram --myarrayoption value1,value2,value3
myprogram --mystringarrayoption="value one","value two","value three"


# The following, however will cause errors.
myprogram --myarrayoption value1, value2, value3
```

2.7 Option Modules

As mentioned earlier, options may overlap if they are categorized into modules. When an option is invoked, the processor checks to see what module is active. The processor invokes the option from that module.

```
myprogram [module1] --myoption 5 [module2] --myoption 8
```

2.7.1 Changing the Active Module

Enclose the module name in square brackets and provide it as an argument to set active module. Consider the following example that demonstrates how option modules may be used.

2.7.2 Example

```
server-manager [http] --timeout=15 [ftp] --timeout=30 [pop] --timeout=20
```

In the example above, three separate options were invoked from three separate modules.

2.8 Option Files

3 Basic Option Parsing

Unlike getopt which many people are familiar with, options must be registered in a repository before they are recognized by the parser. The repository and registrar for options is facilitated in the `Options` class.

3.1 Creating an Options Repository

Creating an options repository is done simply by invoking the default constructor.

```
Options options = new Options();
```

In theory, this is all that needs to be done to get started. The default constructor will provide the repository with a default program name and version information. To demonstrate this simply do a `----help`.

```
cookies@crazymonster$ java MyProgram --help
java program @optionfile [module] OPTIONS ... [module] OPTIONS

 Use --menu to invoke the interactive built-in menu.

 Option Name     Type         Description

 -h, --help      <NOTIFY>     Displays help for each option.
 -m, --menu      <NOTIFY>     Displays the built-in interactive menu.
 -v, --version   <NOTIFY>     Displays version information.
```

3.2 Specifying the program name

A different program name may be specified by passing the name to the constructor.

```
Options options = new Options( "gnu-food" );
```

Whenever ritopt displays program information it will use the name it is initialized.

3.3 Version information

Similarly, version information may be specified by invoking the `setVersion` method.

```
 options.setVersion( "gnu-food 0.2" );
```

When the `----version` option is specified, version information is displayed.

```
 cookies@crazymonster$ gnu-food --version
 gnu-food 0.2
 cookies@crazymonster$
```

3.4 The Basic Option

The main class that represents the principal abstraction for an option is called `Option`. There are several classes which inherit from `Option` that process option values in different ways.

3.5 Option Subclasses

ritopt provides an implementation for each of the primitive types and the String class. The class names are provided below.

- `StringOption` for `String` values.

- `IntOption` for `int` values.

- `BooleanOption` for `boolean` values.

- `CharOption` for `char` values.

- `FloatOption` for `float` values.

- `DoubleOption` for `double` values.

- `ShortOption` for `short` values.

- `LongOption` for `long` values.

The standard wrapper classes provided in the `java.lang` package were not used because they are immutable.

5

3.6 Creating an Option

A default value is provided if the default constructor is used for any of the Option subclasses. To initialize the Option with a default value, pass the value to the constructor using the unwrapped type.

```
StringOption name = new StringOption( "Fred" );
IntOption age = new IntOption( 19 );
FloatOption bankBalance = new FloatOption( 100.15f );
BooleanOption married = new BooleanOption( false );
```

3.7 Registering an Option

Follow the interface provided in the `OptionRegistrar` interface. The registration method requires either a short or long option.

```
// Let's specify both a short and long option.
options.register( "name", 'n', name );

// or just a long option.
options.register( "age", age );

// or just a short option.
options.register( 'm', married );

// How 'bout a description?
options.register( "bank-balance", 'b', "Bank Balance",  bankBalance );
```

3.8 Processing Arguments

Arguments are processed for options by invoking the `process` method on the repository. The left-over arguments are returned to the caller as shown below.

```
String leftoverArgs[] = options.process( args );
```

3.9 Retrieving the Value of a Processed Option

Although it is not specified in the `Option` class, by convention all subclasses implement `getValue()`, `getStringValue()`, and `getObject()` methods. To retrieve an option as its native type, invoke the `getValue()` method. The `getStringValue()` method should return a string representation of the option's value. As one might expect, the `getObject()` method returns the value as an object.

3.10 Putting it all together

The reader should have enough information to create and initialize a repository, register options, and process the command-line. How does it all fit together in a single program? If the program we are writing is simple, I usually declare all of my options as static members and process the arguments in the main method.

```
import gnu.dtools.ritopt.*;

public class RegisterPersonal {

    private static StringOption name = new StringOption( "Fred" );
    private static IntOption age = new IntOption( 19 );
    private static FloatOption bankBalance = new FloatOption( 100.15f );
    private static BooleanOption married = new BooleanOption( false );

    public static void main( String args[] ) {

        // Create an options repository.
        Options options = new Options( "RegisterPersonal" );

        // Set the version.
        options.setVersion( "RegisterPersonal Version 1.0" );

        // Register the options.
        options.register( "name", 'n', name );
        options.register( "age", age );
        options.register( 'm', married );
        options.register( "bank-balance", 'b', "Bank Balance", bankBalance );

        // Process the command line and ignore the leftover arguments.

        options.process( args );

        // Print out the results.

        System.out.println( "Name: " + name.getValue() );
        System.out.println( "Age: " + age.getValue() );
        System.out.println( "Married: " + married.getValue() );
        System.out.println( "BankBalance: " + bankBalance.getValue() );
    }
}
```

Let's compile the program, create a script, and run it a few times.

```
nomad@nomansland$ javac -classpath ritopt-x.x.x-bin.jar:. RegisterPersonal.java
nomad@nomansland$ # Let's create a script
nomad@nomansland$ echo 'java -cp ritopt-x.x.x-bin:'${PWD}' RegisterPersonal' $* \
                  > RegisterPersonal
nomad@nomansland$ chmod +x RegisterPersonal
nomad@nomansland$ ./RegisterPersonal
Name: Fred
Age: 19
Married: false
BankBalance: 100.15
nomad@nomansland$ ./RegisterPersonal --help
RegisterPersonal @optionfile [module] OPTIONS ... [module] OPTIONS

Use --menu to invoke the interactive built-in menu.

Option Name      Type       Description

-h, --help      <NOTIFY>   Displays help for each option.
-m, --menu      <NOTIFY>   Displays the built-in interactive menu.
    --age       <INTEGER>  No description given
-v, --version   <NOTIFY>   Displays version information.
-b, --bank-bala <FLOAT>    Bank Balance
-n, --name      <STRING>   No description given
-m              <BOOLEAN>  No description given

nomad@nomansland$ ./RegisterPersonal --version
RegisterPersonal Version 1.0
nomad@nomansland$ # Let's set the name
nomad@nomansland$ ./RegisterPersonal --name="Jonathan Doe" --age=20
Name: Jonathan Doe
Age: 19
Married: false
BankBalance: 100.15
nomad@nomansland$ ./RegisterPersonal --name="Jonathan Doe" --age=20 -m
Name: Jonathan Doe
Age: 19
Married: false
BankBalance: 100.15
nomad@nomansland$ ./RegisterPersonal --married
Error: Option --married does not exist in module 'General'.
Name: Jonathan Doe
Age: 19
Married: false
BankBalance: 100.15
```

4 Parsing Options of Array Types

5 Event Notification and Handling Interface

6 Deprecating options

7 Licensing and Copyright

The ritopt Tutorial is Copyright ©Damian Ryan Eads, 2001. All Rights Reserved.

ritopt is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

ritopt is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with ritopt; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA