MainAnnotation

Edit this page on GitHub

`MainAnnotation` provides a generic way to define main annotations such as `@main`.

When a users annotates a method with an annotation that extends `MainAnnotation` a class with a `main` method will be generated. The main method will contain the code needed to parse the command line arguments and run the application.

```
/** Sum all the numbers  *
 *  @param first Fist number to sum
 *  @param rest The rest of the numbers to sum
 */
@myMain def sum(first: Int, second: Int = 0, rest: Int*): Int = first + second
```

```
object foo {
  def main(args: Array[String]): Unit = {
    val mainAnnot = new myMain()
    val info = new Info(
      name = "foo.main",
      documentation = "Sum all the numbers",
      parameters = Seq(
        new Parameter("first", "scala.Int", hasDefault=false, isVarargs=false,
        new Parameter("second", "scala.Int", hasDefault=true, isVarargs=false,
        new Parameter("rest", "scala.Int" , hasDefault=false, isVarargs=true,
      )
    )
    val mainArgsOpt = mainAnnot.command(info, args)
    if mainArgsOpt.isDefined then
      val mainArgs = mainArgsOpt.get
      val args0 = mainAnnot.argGetter[Int](info.parameters(0), mainArgs(0), No
using a parser of Int
      val args1 = mainAnnot.argGetter[Int](info.parameters(1), mainArgs(1), So
using a parser of Int
      val args2 = mainAnnot.varargGetter[Int](info.parameters(2), mainArgs.dro
using a parser of Int
      mainAnnot.run(() => sum(args0(), args1(), args2()*))
  }
}
```

The implementation of the `main` method first instantiates the annotation and then call `command`. When calling the `command`, the arguments can be checked and preprocessed. Then it defines a series of argument getters calling `argGetter` for each parameter and `varargGetter` for the last one if it is a varargs. `argGetter` gets an optional lambda that computes the default argument. Finally, the `run` method is called to run the application. It receives a by-name argument that contains the call the annotated method with the instantiations arguments (using the lambdas from `argGetter` / `varargGetter`).

Example of implementation of `myMain` that takes all arguments positionally. It used `util.CommandLineParser.FromString` and expects no default arguments. For simplicity, any errors in preprocessing or parsing results in crash.

```
// Parser used to parse command line arguments
import scala.util.CommandLineParser.FromString[T]

// Result type of the annotated method is Int and arguments are parsed using Fi
@experimental class myMain extends MainAnnotation[FromString, Int]:
  import MainAnnotation.{ Info, Parameter }

  def command(info: Info, args: Seq[String]): Option[Seq[String]] =
    if args.contains("--help") then
      println(info.documentation)
      None // do not parse or run the program
    else if info.parameters.exists(_.hasDefault) then
      println("Default arguments are not supported")
      None
    else if info.hasVarargs then
      val numPlainArgs = info.parameters.length - 1
      if numPlainArgs <= args.length then
        println("Not enough arguments")
        None
      else
        Some(args)
    else
      if info.parameters.length <= args.length then
        println("Not enough arguments")
        None
      else if info.parameters.length >= args.length then
        println("Too many arguments")
        None
      else
        Some(args)
  def argGetter[T](param: Parameter, arg: String, defaultArgument: Option[() =
    () => parser.fromString(arg)

  def varargGetter[T](param: Parameter, args: Seq[String])(using parser: FromSt
    () => args.map(arg => parser.fromString(arg))

  def run(program: () => Int): Unit =
    println("executing program")

    try {
      val result = program()
      println("result: " + result)
      println("executed program")
end myMain
```

‹ Explicit... Captur... ›