Runtime

19 April 2022 09:28

Callback Function: It is a mechanism of passing a function of a specific kind to some code with purpose of allowing this code to indirectly call that function. Java provides syntactical support for callback functions through

1. **Method Reference** - It is a reference type which can be bound to any method with a particular list of parameter types and a particular return type. It is produced by applying double-colon (::) operator to a class or its object along with the name of (static or non-static) method defined by that class or from an anonymous method implementation called a *lambda expression*.
2. **Functional Interface** - It is an interface containing exactly one abstract method defined to be used as a type for a method-reference which compatible with that abstract method. It is automatically implemented at runtime to invoke the method bound to the method reference assigned to its identifier.

Functional Programming: It is a *declarative style* of programming in which data is processed by passing it through a chain of function calls which do not have any side-effects and may accept other functions as their arguments. Java runtime library includes support for processing a sequence of elements known as a *stream* in a functional manner using

1. **Operation Pipeline** - It is a chain of methods consisting of *intermediate* operations each of which consumes a stream of elements and produces a stream of processed elements and a *terminal* operation which does not produce a stream.
2. **Lazy Evaluation** - Each operation in the pipeline performs its own internal iteration and the iteration within an intermediate operation is executed only when the terminal operation is called.

Reflection: It is a mechanism which enables a program to examine the structure of its own data at runtime. In Java the meta-data (description) of a type **T** is provided by an instance of **java.lang.Class<T>** whose reference can be obtained at compile-time using expression **T.class**. The reference to a class object can also be discovered at runtime from

1. An object obj of a reference type

   Class<?> c = obj.getClass();

   Note: c will refer to java.lang.Class of reference type which was used for constructing the instance referred by obj
2. The fully-qualified name N of the type

Class<?> c = Class.forName(N);

Note: The class with specified name N will be loaded using the built-in class-loader of the JVM. The built-in class loader loads the binary representation of type p.T from path p/T.class by searching for this path in each location (directory or archive) specified by the java.class.path property of the JVM which defaults to the current directory and implicitly includes Java runtime library.

Annotation: It is a *metadata representing type* which can be applied as a *custom modifier* to a declaration target such as package, type (class or interface), field and method. It is defined as an interface with one of the following *retention policies* to indicate how it is retained by its declaration target:

1. **SOURCE** - The annotation appears in the source code of the declaration target but is discarded by the compiler. Such an annotation can only be examined at compile-time through its annotation provider.
2. **CLASS** - The annotation is inserted into the binary representation of the declaration target within the class file but it is discarded at runtime. Such an annotation can only be examined at compile-time through its annotation provider.
3. **RUNTIME** - The annotation is loaded into the memory at runtime along with the metadata of its declaration target. Such an annotation can be examined at runtime using reflection.

Native Method: It a a method defined (with native modifier) in a Java class which on invocation calls a *machine-specific global function* exported by a *platform-specific dynamically linkable library*. The function called by a native method is commonly implemented in C/C++ and it interacts with the JVM using its *Java Native Interface* (supported through JNIEnv type which addresses a struct containing pointers to functions published by the JVM).

1. **Purpose**
   (a) to increase performance of a Java program by avoiding excessive verification done by the JVM.
   (b) to consume non-Java code including the services offered by the platform which are not exposed by Java runtime library.
2. **Disadvantages**
   (a) it can compromise the execution safety ensured by the JVM to protect a program from crashing at runtime.
   (b) it can limit the portability of program to different platforms supported by Java.