Documentation

The Java™ Tutorials

Trail: The Reflection API
Lesson: Members
Section: Methods

Invoking Methods

Reflection provides a means for invoking methods on a class. Typically, this would only be necessary if it is not possible to cast an instance of the class to the desired type in non-reflective code. Methods are invoked with `java.lang.reflect.Method.invoke()`. The first argument is the object instance on which this particular method is to be invoked. (If the method is `static`, the first argument should be `null`.) Subsequent arguments are the method's parameters. If the underlying method throws an exception, it will be wrapped by an `java.lang.reflect.InvocationTargetException`. The method's original exception may be retrieved using the exception chaining mechanism's `InvocationTargetException.getCause()` method.

Finding and Invoking a Method with a Specific Declaration

Consider a test suite which uses reflection to invoke private test methods in a given class. The `Deet` example searches for `public` methods in a class which begin with the string "`test`", have a boolean return type, and a single `Locale` parameter. It then invokes each matching method.

```
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.lang.reflect.Type;
import java.util.Locale;
import static java.lang.System.out;
import static java.lang.System.err;

public class Deet<T> {
    private boolean testDeet(Locale l) {
        // getISO3Language() may throw a MissingResourceException
        out.format("Locale = %s, ISO Language Code = %s%n", l.getDisplayName(), l.getISO3Language());
        return true;
    }

    private int testFoo(Locale l) { return 0; }
    private boolean testBar() { return true; }

    public static void main(String... args) {
        if (args.length != 4) {
            err.format("Usage: java Deet <classname> <langauge> <country> <variant>%n");
            return;
        }

        try {
            Class<?> c = Class.forName(args[0]);
            Object t = c.newInstance();

            Method[] allMethods = c.getDeclaredMethods();
            for (Method m : allMethods) {
                String mname = m.getName();
                if (!mname.startsWith("test")
                    || (m.getGenericReturnType() != boolean.class)) {
                    continue;
                }
                Type[] pType = m.getGenericParameterTypes();
                if ((pType.length != 1)
                    || Locale.class.isAssignableFrom(pType[0].getClass())) {
                    continue;
                }

                out.format("invoking %s()%n", mname);
```

```
                    try {
                        m.setAccessible(true);
                        Object o = m.invoke(t, new Locale(args[1], args[2], args[3]));
                        out.format("%s() returned %b%n", mname, (Boolean) o);

                    // Handle any exceptions thrown by method to be invoked.
                    } catch (InvocationTargetException x) {
                        Throwable cause = x.getCause();
                        err.format("invocation of %s failed: %s%n",
                                   mname, cause.getMessage());
                    }
                }

            // production code should handle these exceptions more gracefully
            } catch (ClassNotFoundException x) {
                x.printStackTrace();
            } catch (InstantiationException x) {
                x.printStackTrace();
            } catch (IllegalAccessException x) {
                x.printStackTrace();
            }
        }
    }
```

`Deet` invokes `getDeclaredMethods()` which will return all methods explicitly declared in the class. Also, `Class.isAssignableFrom()` is used to determine whether the parameters of the located method are compatible with the desired invocation. Technically the code could have tested whether the following statement is `true` since `Locale` is `final`:

```
Locale.class == pType[0].getClass()
```

However, `Class.isAssignableFrom()` is more general.

```
$ java Deet Deet ja JP JP
invoking testDeet()
Locale = Japanese (Japan,JP),
ISO Language Code = jpn
testDeet() returned true
```

```
$ java Deet Deet xx XX XX
invoking testDeet()
invocation of testDeet failed:
Couldn't find 3-letter language code for xx
```

First, note that only `testDeet()` meets the declaration restrictions enforced by the code. Next, when `testDeet()` is passed an invalid argument it throws an unchecked `java.util.MissingResourceException`. In reflection, there is no distinction in the handling of checked versus unchecked exceptions. They are all wrapped in an `InvocationTargetException`

Invoking Methods with a Variable Number of Arguments

`Method.invoke()` may be used to pass a variable number of arguments to a method. The key concept to understand is that methods of variable arity are implemented as if the variable arguments are packed in an array.

The `InvokeMain` example illustrates how to invoke the `main()` entry point in any class and pass a set of arguments determined at runtime.

```
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.Arrays;

public class InvokeMain {
    public static void main(String... args) {
        try {
            Class<?> c = Class.forName(args[0]);
            Class[] argTypes = new Class[] { String[].class };
            Method main = c.getDeclaredMethod("main", argTypes);
            String[] mainArgs = Arrays.copyOfRange(args, 1, args.length);
            System.out.format("invoking %s.main()%n", c.getName());
            main.invoke(null, (Object)mainArgs);

        // production code should handle these exceptions more gracefully
        } catch (ClassNotFoundException x) {
            x.printStackTrace();
```

```
            } catch (NoSuchMethodException x) {
                x.printStackTrace();
            } catch (IllegalAccessException x) {
                x.printStackTrace();
            } catch (InvocationTargetException x) {
                x.printStackTrace();
            }
        }
    }
```

First, to find the `main()` method the code searches for a class with the name "main" with a single parameter that is an array of `String` Since `main()` is `static`, `null` is the first argument to `Method.invoke()`. The second argument is the array of arguments to be passed.

```
$ java InvokeMain Deet Deet ja JP JP
invoking Deet.main()
invoking testDeet()
Locale = Japanese (Japan,JP),
ISO Language Code = jpn
testDeet() returned true
```

Previous page: Retrieving and Parsing Method Modifiers
Next page: Troubleshooting