Java 11 教程（译）

2018-10-31

Java 11已经发布，很多人还在使用Java 8。这篇教程讲述一些重要的语言特性和API。

局部变量类型推断

局部变量指在方法体内声明的变量。Java10就已经引进一个新的关键字var，用于代替在声明局部变量时候的类型声明。

在Java 10之前，你必须这样声明一个变量：

```
String text = "Hello Java 9";
```

现在你可以使用var代替String。编译器会自动从变量的赋值推断出正确的类型。如文本的类型为String：

```
var text = "Hello Java 10";
```

使用var声明的变量仍然是静态变量，不可以在声明后赋值为其它类型：

```
var text = "Hello Java11";
text = 23;  // 编译错误，不兼容的类型
```

同样可以使用final声明变量为常量：

```
final var text = "Banana";
text = "Joe";   // 编译错误
```

当然，在编译器无法推断出类型的场景下，不可以使用var，比如这些情况：

```
var a;
var nothing = null;
var lamdba = () -> System.out.println("Pity!");
var method = this::someMethod;
```

当变量声明包含泛型时，var的优势尤为突出，下面的示例就用var来代替冗长的Map<String, List>：

```
var myList = new ArrayList<Map<String, List<Integer>>>();

for (var current : myList) {
    // current 的类型会被推断为 Map<String, List<Integer>>
    System.out.println(current);
}
```

Java 11的var关键字同样支持在lamdba表达式的参数中使用，并且支持为这些参数添加注解：

```
Predicate<String> predicate = (@Nullable var a) -> true;
```

> 小技巧：在Intellij IDEA中按住CTRL键可以查看变量的推断类型。

HTTP Client

从Java 9开始引进试用新的API HttpClient，用于处理HTTP请求。现在Java 11将其标准化，我们可以从模块java.net中获取使用。

新的HttpClient在同步和异步场景下都可以使用。同步请求会阻塞线程，直到获取到响应。BodyHandlers定义了响应数据的类型（如String、Byte[]、File）。

```
var request = HttpRequest.newBuilder()
        .uri(URI.create("https://blog.smallyu.net"))
        .GET()
        .build();
var client = HttpClient.newHttpClient();
var response = client.send(request, HttpResponse.BodyHandlers.ofString());
System.out.println(response.body());

// 记得在module-info.java中导入java.net.http模块
```

同样可以使用异步的方式实现请求，调用sendAsync方法并不会阻塞当前线程，它会构建异步操作流，在接收到响应后执行相应操作：

```
var request = HttpRequest.newBuilder()
        .uri(URI.create("https://blog.smallyu.net"))
        .build();
var client = HttpClient.newHttpClient();
client.sendAsync(request, HttpResponse.BodyHandlers.ofString())
        .thenApply(HttpResponse::body)
        .thenAccept(System.out::println);

// 线程睡眠，防止在返回响应前当前线程就结束
Thread.sleep(3000);
```

.GET()方法会作为默认的请求方式。

下一个示例通过POST方式发送请求到指定URL。与BodyHandlers相似，使用BodyPublishers定义要发送的数据类型：

```
var request = HttpRequest.newBuilder()
        .uri(URI.create("https://postman-echo.com/post"))
        .header("Content-Type", "text/plain")
        .POST(HttpRequest.BodyPublishers.ofString("Hi there!"))
        .build();
var client = HttpClient.newHttpClient();
var response = client.send(request, HttpResponse.BodyHandlers.ofString());
System.out.println(response.statusCode());      // 200
```

最后一个示例演示了如何使用BASIC-AUTH执行权限验证：

```
var request = HttpRequest.newBuilder()
    .uri(URI.create("https://postman-echo.com/basic-auth"))
    .build();
var client = HttpClient.newBuilder()
    .authenticator(new Authenticator() {
        @Override
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication("postman", "password".toCharArray());
        }
    })
    .build();
var response = client.send(request, HttpResponse.BodyHandlers.ofString());
System.out.println(response.statusCode());      // 200
```

集合框架

集合框架如List、Set和Map都增加了新的方法。List.of方法根据给定参数创建一个不可变列表，List.copyOf创建一个已存在列表的副本。

```
var list = List.of("A", "B", "C");
var copy = List.copyOf(list);
System.out.println(list == copy);   // true
```

因为列表已经不可变，所以拷贝出的列表和原列表是同一实例。如果拷贝了一个可变列表，拷贝出的列表会是一个新的实例，不会对原列表产生副作用：

```
var list = new ArrayList<String>();
var copy = List.copyOf(list);
System.out.println(list == copy);   // false
```

创建不可变映射不必自己创建映射实体，只需要将key和value交替传入作为参数：

```
var map = Map.of("A", 1, "B", 2);
System.out.println(map);    // {B=2, A=1}
```

> Java 11中的不可变列表和旧版本的列表使用相同的接口，但是如果你对不可变列表进行修改，如添加或移除元素，程序会抛出java.lang.UnsupportedOperationException异常。幸运的是，当你试图修改不可变列表，Intellij IDEA会检查并给出警告。

Streams

Streams从Java 8开始引进，现在新增了三个方法。Stream.ofNullable从单个元素构建流：

```
Stream.ofNullable(null)
    .count()   // 0
```

dropWhile和takeWhile方法都是用于放弃流中的一些元素：

```
Stream.of(1, 2, 3, 2, 1)
    .dropWhile(n -> n < 3)
    .collect(Collectors.toList());  // [3, 2, 1]

Stream.of(1, 2, 3, 2, 1)
    .takeWhile(n -> n < 3)
    .collect(Collectors.toList());  // [1, 2]
```

字符串

String类也新增了一些方法：

```
" ".isBlank();                // true
" Foo Bar ".strip();          // "Foo Bar"
" Foo Bar ".stripTrailing();  // " Foo Bar"
" Foo Bar ".stripLeading();   // "Foo Bar "
"Java".repeat(3);             // "JavaJavaJava"
"A\nB\nC".lines().count();    // 3
```

其他JVM特性

Java 11包含许多新特性，以上只提及冰山一角，权作抛砖引玉，更多内容等待你探索……

参考

- Java 11 Tutorial