链滴

我还在生产玩 JDK7，JDK 15 却要来了!| 新特性尝鲜

作者：9526xu
原文链接：https://ld246.com/article/1596266360488
来源网站：链滴

自从 JDK9 之后，每年 3 月与 9 月 JDK 都会发布一个新的版本，而2020 年 9 月即将引来 JDK15。

恰巧 IDEA 每四五个月会升级一个较大的版本，每次升级之后都会支持最新版本 JDK 引入的新功能。

这几天升级了 IDEA，顺便体验了一下 JDK15 的新特性。

虽然我知道你们可能跟我一样JDK8 都还没用熟，但是无妨，看看新版本 JDK 来酸一下。

Text Blocks 最终定板

之前版本的 JDK，如果我们需要插入 **HTML**，**XML**，**SQL** 或 **JSON** 片段，非常麻烦，需要对里面符进行各种转义。

所以我每次都会在其他编辑器将 **HTML** ,**XML** 等编辑好，然后直接复制到 IDEA 中，IDEA 自动会对些字符转义。

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello, world</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

每次复制进去就变成上图的效果，如果上面字符再多点，阅读起来就会更难，并且难以维护。

所幸 IDEA 提供了一个 **Inject Language** 功能，我们可以在里面快速方便的编辑。

Java 开发者也关注到这个问题，他们在 JDK13 引入的一个新的预览特性「**Text Blocks**」，可以使三引号将复杂的字符串赋值，从而让我们从各种转义中解脱出来，可以更加方便的编辑字符串。

这个功能在其他语言还是比较常见的，比如 Python 等。

Text Blocks 新功能在 JDK14 再次以预览功能引入，最终在 JDK15 成为新版本的正式功能。

下面我们来对比一下使用 **Text Blocks** 与之前区别：

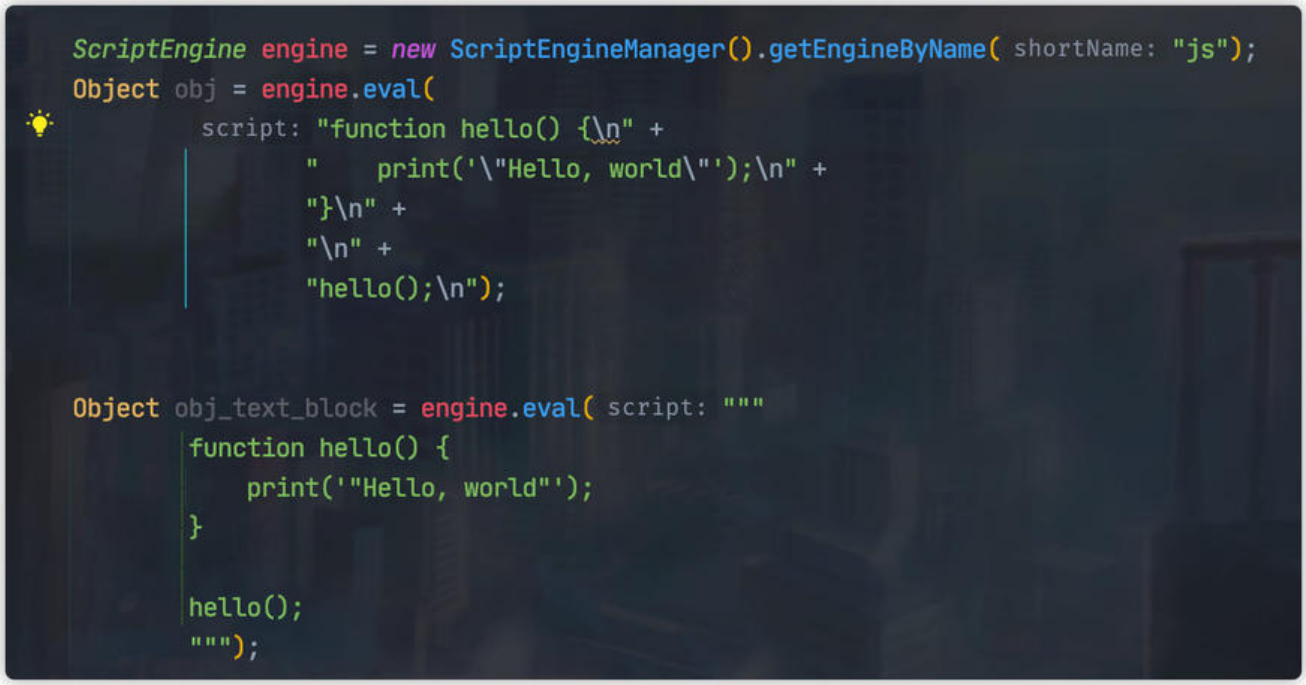
```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello, world11</p>\n" +
              "    </body>\n" +
              "</html>\n";

String text_block_html = """
              <html>
                  <body>
                      <p>Hello, world</p>
                  </body>
              </html>
              """;
```

```
// SQL
String query = "SELECT \"EMP_ID\", \"LAST_NAME\" FROM \"EMPLOYEE_TB\"\n" +
        "WHERE \"CITY\" = 'INDIANAPOLIS'\n" +
        "ORDER BY \"EMP_ID\", \"LAST_NAME\";\n";

String text_block_sql = """
        SELECT "EMP_ID", "LAST_NAME" FROM "EMPLOYEE_TB"
                    WHERE "CITY" = 'INDIANAPOLIS'
                    ORDER BY "EMP_ID", "LAST_NAME";
        """;
```

原文链接：我还在生产玩 JDK7，JDK 15 却要来了!| 新特性尝鲜



Records (Second Preview)

JDK14 引入一个新的预览特性 **record** 语法，可以快速创建一个纯数据类，并且不用去生成 **getter**，**oString** 等。

使用下面的语法就可以快速创建一个数据类：

```
public record Point(int x,int y) {
}
```

JDK15 是 **record** 这个语法的第二次预览，这个版本增加一个新的功能 「**local record**」，可以在一方法在快速创建一个类，以便于方法中业务逻辑计算。

在以下示例中，使用本地记录 **MerchantSales** 对商人和每月销售额的汇总进行建模，使用此记录可高以下流操作的可读性：

下面例子的中我们新建一个类 **MerchantSales**，然后按照销售人员对每月的销售额汇总排序。

```
List<Merchant> findTopMerchants(List<Merchant> merchants, int month) {
    // Local record
    record MerchantSales(Merchant merchant, double sales) {}

    return merchants.stream()
        .map(merchant -> new MerchantSales(merchant, computeSales(merchant, month)))
        .sorted((m1, m2) -> Double.compare(m2.sales(), m1.sales()))
        .map(MerchantSales::merchant)
        .collect(toList());
}
```

原先如果需要使用这种功能，我们不得不创建一个内部类，后续可能再也不会用到，使用 **local record** 解决这个尴尬的问题。

除了 **local record** 我们还可以创建 **local enums** 以及 **local interface**。

```
// local enums
public void organisePeople(List<Person> people) {
    enum Role {
        Employee, Customer, Both, None
    }
    HashMap<Role, List<Person>> peopleByRole = new HashMap<>();
    people.stream()
            .filter(Person::isCustomer)
            .forEach(person -> peopleByRole.computeIfAbsent(Role.Customer, role -> new ArrayL
st<>())
                    .add(person));
    // 其他业务逻辑

}

// local interface
public void localInterface() {
    interface MyInterface {
        void doSomething();
    }
    MyInterface testInterface = new MyInterface() {
        @Override
        public void doSomething() {
            System.out.println("Hello World!");
        }
    };
    // 其他业务逻辑

}
```

最后使用这个特性需要注意一点，**local record** , **local enums** ,**local interface** 创建都是一个局部量，是不能被传递其他方法引用。

Pattern Matching for instanceof (Second Preview)

我们应该都看到过下面这种代码：

```
if (obj instanceof String) {
    String str = (String) obj;
    // use str
}
```

上面代码意图非常简单，当 **obj** 对象是 **String** 类，就将其强制转换，然后进行其他业务操作。

这种写法，类型转换还是比较繁琐，**Pattern Matching for instanceof** 这个新语法特性，可以帮们省略这种类型转换动作。这是一个在 JDK14 引入一个预览特性，JDK 15 开始第二次预览。

上面的代码使用 **pattern matcher**，就可以被修改如下：

```
if (obj instanceof String s) {
    s.contains("T");
} else {
    // 编译错误
    //s.contains("T");
```

```
}
```

另外如果在 IDEA 中还可以提示我们将代码转化成 **pattern matcher** 。

大家应该都看过 **Effective Java** 这本神书吧，里面第八条关于 **Equals** 有一个例子：

```
@Override
public boolean equals(Object o) {
    return o instanceof CaseInsensitiveString && ((CaseInsensitiveString) o).s.equalsIgnoreCase(s);
}
```

使用 **pattern matcher** 我们就可以使用下面更加清晰的代码代替：

```
@Override
public boolean equals(Object o) {
    return (o instanceof CaseInsensitiveString cis) && cis.s.equalsIgnoreCase(s);
}
```

Sealed Classes (Preview)

Java 中一个正常普通类/接口允许被其他子类继承/实现，但是有时在日常开发中，我们可能希望只有定的类才能继承扩展。

现有的 Java 语法中存在一些方法，可以限制子类扩展，比如说：我们可以使用 **final** 修饰类

```
public final class String
```

不过这样之后，我们就没办法再继承这个类。

其次我们可以限制的类的范围，比如说不使用 **public** 修饰类/接口，即使用 **default** 范围，这样只有

一个包才能继承/实现。

```
interface DefaultExample {
}
```

不过使用这种方式，又很尴尬，这个类就无法被其他包使用。

为了解决上述问题，JDK 15 引入一个新的预览特性 **Sealed Classes**，即可以限定类的扩展，也可以外部使用。

```
public sealed class Shape
    permits Circle, Rectangle, Square {...}
```

使用 **sealed** 修饰之后，Shape 类只能被 Circle，Rectangle，Square继承，再也不能被其他类继承。

同时 **Shape** 的子类存在一些限制，必须使用 **final** 修饰，表明这个类无法再被扩展：

```
public final class Circle extends Shape {...}
```

或者继续使用 **sealed** 表示子类只能被指定类继承：

```
public sealed class Rectangle extends Shape
    permits TransparentRectangle, FilledRectangle {...}
public final class TransparentRectangle extends Rectangle {...}
```

又或者说使用 **non-sealed** 表明这个子类不限制子类扩展，可以被其他任何类扩展实现。

另外 **sealed class** 还可以跟上述 **record** 语法一起使用。

```
public sealed interface Expr
    permits ConstantExpr, PlusExpr, TimesExpr, NegExpr {...}

public record ConstantExpr(int i)       implements Expr {...}
public record PlusExpr(Expr a, Expr b)  implements Expr {...}
public record TimesExpr(Expr a, Expr b) implements Expr {...}
public record NegExpr(Expr e)           implements Expr {...}
```

ZGC

ZGC(Z Garbage Collector) 这是一款在 JDK11 引入的的具有实验性质的低延迟的 GC 收集器。

这款 GC 收集器的希望在尽可能对吞吐量影响不大的前提下，实现在任意堆内存大小都可以把垃圾收器的停顿时间限制在十毫秒以内的低延迟。

ZGC 经过这两三年的迭代优化，终于在 JDK15 中正式引入，标志着 **ZGC** 可以正式应用于生产应用。

JDK15 中默认虚拟机还是 **G1**，如果需要使用 ZGC，需要在启动参数中加入如下参数：

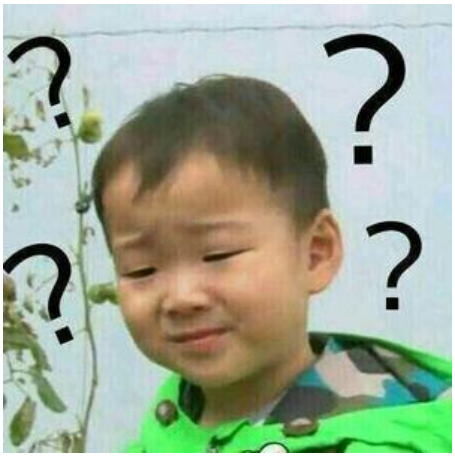
```
-XX:+UseZGC command-line
```

最后

本来这篇文章是准备写下 IDEA 2020.2 新版本特性，顺带介绍一下 JDK15 新特性的。

可是没想到写着写着，JDK15 相关的篇幅就过长了，所以就单独拿出来了。

最后，最后，JDK 都发布到 15 了，而我却还在用 JDK 7 ，真是个悲伤的故事，逃了逃了！



我是楼下小黑哥，每天学习一点点，成长亿点点！！

参考链接

https://openjdk.java.net/projects/jdk/15/