(http://baeldung.com)

Inferencia de tipos de Java 10 LocalVariable

Última modificación: 4 de junio de 2018

por Ganesh Pagade (http://www.baeldung.com/author/ganesh-pagade/)
(http://www.baeldung.com/author/ganesh-pagade/)

Java (http://www.baeldung.com/category/java/) +

Java 10 (http://www.baeldung.com/tag/java-10/)

Este artículo es parte de una serie:

1. Información general

Una de las mejoras más visibles en JDK 10 es la inferencia tipo de variables locales con inicializadores.

Este tutorial proporciona los detalles de esta característica con ejemplos.

2. Introducción

Hasta Java 9, tuvimos que mencionar explícitamente el tipo de variable local y asegurarnos de que era compatible con el inicializador utilizado para inicializarlo:

```
String message = "Good bye, Java 9";
```

En Java 10, así es como podríamos declarar una variable local:

```
@Test
public void whenVarInitWithString_thenGetStringTypeVar() {
    var message = "Hello, Java 10";
    assertTrue(message instanceof String);
}
```

No proporcionamos el tipo de datos del *mensaje* . En cambio, marcamos *el mensaje* como una *var* , y el compilador deduce el tipo de *mensaje* del tipo de inicializador presente en el lado derecho.

En el ejemplo anterior, el tipo de *mensaje* sería *String* .

Tenga en cuenta que esta característica está disponible solo para variables locales con el inicializador. No se puede usar para variables miembro, parámetros de método, tipos de retorno, etc. - se requiere el inicializador sin el cual el compilador no podrá inferir el tipo.

Esta mejora ayuda a reducir el código repetitivo; por ejemplo:

```
Map<Integer, String> map = new HashMap<>();
```

Esto ahora puede ser reescrito como:

```
var idToNameMap = new HashMap<Integer, String>();
```


Esto también ayuda a enfocarse en el nombre de la variable más que en el tipo de variable.

Otra cosa a tener en cuenta es que ***var* no es una palabra clave** ; esto garantiza la compatibilidad con versiones anteriores para programas que usan *var* say, como una función o nombre de variable. *var* es un nombre de tipo reservado, al igual que *int* .

Finalmente, tenga en cuenta que no **hay sobrecarga de tiempo de ejecución al usar *var* ni hace que Java sea un lenguaje de tipeo dinámico.** El tipo de la variable aún se deduce en el momento de la

compilación y no se puede cambiar más adelante.

3. Uso ilegal de *var*

Como se mencionó anteriormente, *var* no funcionará sin el inicializador:

```
var n; // error: cannot use 'var' on variable without initializer
```

Tampoco funcionaría si se inicializa con *nulo* :

```
var emptyList = null; // error: variable initializer is 'null'
```

No funcionará para variables no locales:

```
public var = "hello"; // error: 'var' is not allowed here
```

La expresión Lambda necesita un tipo de objetivo explícito y, por lo tanto, *var* no se puede usar:

```
var p = (String s) -> s.length() > 10; // error: lambda expression nee
```

Lo mismo es el caso con el inicializador de matriz:

```
var arr = { 1, 2, 3 }; // error: array initializer needs an explicit t
```

4. Pautas para usar *var*

Hay situaciones donde *var* se puede usar legalmente, pero puede no ser una buena idea hacerlo.

Por ejemplo, en situaciones donde el código podría ser menos legible:

```
var result = obj.prcoess();
```

Aquí, aunque un uso legal de *var* , se vuelve difícil entender el tipo devuelto por el *proceso () que* hace que el código sea menos legible.

java.net (http://java.net/) tiene un artículo dedicado a las pautas de estilo para la inferencia de tipo de variable local en Java, (http://openjdk.java.net/projects/amber/LVTIstyle.html) que habla sobre cómo debemos usar el juicio al usar esta función.

Otra situación en la que es mejor evitar *var* es en flujos con una gran cantidad de proyectos:

```
var x = emp.getProjects.stream()
  .findFirst()
  .map(String::length)
  .orElse(0);
```

El uso de *var* también puede dar un resultado inesperado.

Por ejemplo, si lo usamos con el operador de diamante presentado en Java 7:

```
var empList = new ArrayList<>();
```

El tipo de *empList* será *ArrayList <Object>* y no *List <Object>* . Si queremos que sea *ArrayList <Employee>* , tendremos que ser explícitos:

```
var empList = new ArrayList<Employee>();
```

El uso de *var* con tipos no denotables podría causar un error inesperado.

Por ejemplo, si usamos *var* con la instancia de clase anónima:

```
@Test
public void whenVarInitWithAnonymous_thenGetAnonymousType() {
    var obj = new Object() {};
    assertFalse(obj.getClass().equals(Object.class));
}
```

Ahora, si tratamos de asignar otro *Object* a *obj*, obtendríamos un error de compilación:

```
obj = new Object(); // error: Object cannot be converted to <anonymous
```

Esto se debe a que el tipo inferido de *obj* no es *Object*.

5. Conclusión

En este artículo, vimos la nueva característica de inferencia de tipo de variable local Java 10 con ejemplos.

Como de costumbre, los fragmentos de código se pueden encontrar en GitHub (https://github.com/eugenp/tutorials/tree/master/core-java-10) .

Siguiente »

Mejoras de rendimiento de Java 10 (http://www.baeldung.com/java-10-performance-improvements)

2 Deja una respuesta

Join the discussion...

1 1 0

Suscribir

¿Cuál de estos es el más cercano a su trabajo / función actual?

Desarrollador 2

Desarrollador Senior

el más nuevo más antiguo el más votado

Desarrollador principal

Arquitecto

Alonso Isidoro Romano (http://aironman2k.wordpress.com)

Gerente

Huésped

It's funny that precisely what makes it nice for some people to use python or scala instead of java to not have to declare the types, now we seem to realize that it's bad practice not to force the typing with version 10. It has always been a bad idea not to force typing, both because of performance issues and because of code maintenance and readability issues.

+ 0 − Reply

8 days ago

(http://www.baeldung.com/author/grzegorz-author/)

Editor

Grzegorz Piwowarek (http://4comprehension.com)

Everything is contextual, it's just a bit easier to write unreadable code if you're not forced to provide types explicitly.

+ 0 − Reply

6 days ago

CATEGORIES

SPRING (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY/)
PERSISTENCE (HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)
HTTPCLIENT (HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

¿Cuál de estos es el más cercano a su trabajo / función actual?

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

SERIES

JAVA "BACK TO BASICS" TUTORIAL (HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (HTTP://WWW.BAELDUNG.COM/JACKSON)
HTTPCLIENT 4 TUTORIAL (HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/)
SPRING PERSISTENCE TUTORIAL (HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/)
SECURITY WITH SPRING (HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING)

ABOUT

ACERCA DE BAELDUNG (HTTP://WWW.BAELDUNG.COM/ABOUT/)
LOS CURSOS (HTTP://COURSES.BAELDUNG.COM)
TRABAJO DE CONSULTORÍA (HTTP://WWW.BAELDUNG.COM/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
EL ARCHIVO COMPLETO (HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE)
ESCRIBIR PARA BAELDUNG (HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES)
CONTACTO (HTTP://WWW.BAELDUNG.COM/CONTACT)
INFORMACIÓN DE LA COMPAÑÍA (HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
TÉRMINOS DE SERVICIO (HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY)
EDITORES (HTTP://WWW.BAELDUNG.COM/EDITORS)
KIT DE MEDIOS (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF)