.

第十六章

明确赋值

每一个由语句(§14.4.2、§14.14.1、§14.14.2、§14.20.3)所声明的局部变量和每一个空 final 字段(§4.12.4、§8.3.1.2)，在对其值的任何访问发生时，都必须有一个明确的赋值。

对它的值的访问由变量的简单名称(或者，对于字段，由 this 限定的字段的简单名称)组成，它出现在表达式的任何地方，除了作为简单赋值操作符=的左操作数(§15.26.1)。

对于每次访问局部变量或空 final 字段 x，必须在访问之前明确赋值 x，否则将发生编译时错误。

同样，每个空 final 变量必须最多赋值一次;当对它进行赋值时，它必须是明确未赋值的。

这样的赋值被定义为当且仅当变量的简单名称(或者，对于字段，由 this 限定的它的简单名称)出现在赋值操作符的左边时发生。

对于每个空 final 变量的赋值，在赋值之前变量必须是明确未赋值的，否则将发生编译时错误。

本章的其余部分将专门对“以前明确赋值”和“以前明确未赋值”这两个词作一个精确的解释。

明确赋值背后的思想是，对局部变量或空 final 字段的赋值必须发生在访问的每一个可能的执行路径上。类似地，明确未赋值背后的思想是，在赋值的任何可能的执行路径上，都不允许对空 final 变量进行其他赋值。

分析考虑了语句和表达式的结构；它还提供了对表达式操作符&&, ||, !, ?:, 和布尔值常量表达式的特殊处理。

除了条件布尔运算符&&, ||,?:和布尔值常量表达式，表达式的值在流分析中不被考虑。

例子 16-1. 明确赋值考虑语句和表达式的结构

Java 编译器识别出，在代码中访问(作为方法调用的参数)之前，k 是明确赋值的:

```
{

    int k;
    if (v > 0 && (k = System.in.read()) >= 0)
        System.out.println(k);
}
```

因为访问发生只有当表达式的值:

```
v > 0 && (k = System.in.read()) >= 0
```

为 true, 只有在执行了对 k 的赋值(更确切地说，是求值)时，该值才可能为 true。

类似地，Java 编译器将在代码中识别:

```
{
    int k;
    while (true) {
        k = n;
        if (k >= 5) break;
        n = 6;
    }
    System.out.println(k);
}
```

变量 k 肯定是由 while 语句明确赋值的，因为条件表达式 true 从来没有值 false，所以只有 break 语句才能使 while 语句正常完成，并且 k 肯定是在 break 语句之前赋值的。

另一方面，代码:

```
{
    int k;
    while (n < 4) {
        k = n;
        if (k >= 5) break;
        n = 6;
    }
    System.out.println(k); /* k is not "definitely assigned before this statement
                              */
}
```

qinglin

必须被 Java 编译器拒绝，因为在这种情况下，就明确赋值规则而言，不能保证 while 语句执行其主体。

例子 16-2. 确定赋值不考虑表达式的值

Java 编译器必须为代码生成编译时错误:

```
{
    int k;
    int n = 5;
    if (n > 2)
        k = 3;
    System.out.println(k); /* k is not "definitely assigned" before this statement
                              */
}
```

即使 n 的值在编译时是已知的，并且原则上在编译时可以知道，对 k 的赋值将始终被执行(更确切地说，是计算)。Java 编译器必须根据本节中列出的规则进行操作。规则仅识别常量表达式；在本例中，表达式 n>2 不是§15.29 中定义的常量表达式。

作为另一个示例，Java 编译器将接受以下代码:

```
void flow(boolean flag) { int k;
    if (flag)
        k = 3;
    else
        k = 4;
```

```
        System.out.println(k);
    }
```

就 k 的确定赋值而言，因为本节中概述的规则允许它告诉 k 被赋值，无论 flag 是 true 还是 false。但规则不接受这种变化：

```
    void flow(boolean flag) {
        int k;
        if (flag)
            k = 3;
        if (!flag)
            k = 4;
        System.out.println(k); /* k is not "definitely assigned" before this statement
                                  */
    }
```

因此，编译此程序必然会导致发生编译时错误。

例子 16-3. 明确未赋值

Java 编译器将接受以下代码：

```
    void unflow(boolean flag) {
        final int k;
        if (flag) {
            k = 3;
            System.out.println(k);
        }
        else {
            k = 4;
            System.out.println(k);
        }
    }
```

至于 k 的明确未赋值，因为本节中概述的规则允许它告诉 k 至多赋值一次(实际上，恰好是一次)，无论 flag 是 true 还是 false。但规则不接受这种变化：

```
    void unflow(boolean flag) {
        final int k;
        if (flag) {
            k = 3;
            System.out.println(k);
        }
        if (!flag) {
            k = 4;
            System.out.println(k); /* k is not "definitely unassigned" before this
                                      statement */
        }
    }
```

因此，编译此程序必然会导致发生编译时错误。

为了准确地说明所有明确赋值的情况，本节中的规则定义了几个技术术语：

- 变量是否在语句或表达式之前明确赋值
- 变量是否在语句或表达式之前明确未赋值

qingliu

- 变量是否在语句或表达式之后明确赋值
- 变量是否在语句或表达式之后明确未赋值

对于布尔值表达式，后两种情况被细化为四种情况：

- 如果为 true，是否在表达式之后明确赋值变量
- 如果为 true，是否在表达式之后明确不赋值变量
- 如果为 false，是否在表达式之后明确赋值变量
- 如果为 false，是否在表达式之后明确不赋值变量

在这里，为 true 和为 false 指的是表达式的值。

> 例如，局部变量 k 在表达式求值后被明确赋值：
>
> ```
> a && ((k=m) > 5)
> ```
>
> 当表达式为 true 时，而不是当表达式为 false 时(因为如果 a 为 false，则不执行对 k 的赋值(更准确地说，是计算))。

“V 在 X 之后明确赋值”(其中 V 是一个局部变量，X 是一个语句或表达式)表示“如果 X 正常完成，V 在 X 之后明确赋值”。如果 X 突然结束，赋值就不需要发生，这里所述的规则考虑到了这一点。

> 这个定义的一个特别的结果是“V 是在 break 之后明确赋值的;”总是正确的! 因为 break 语句永远不会正常完成，所以如果 break 语句正常完成，则 V 被赋予了一个值。

“V 在 X 之后是明确未赋值的”这句话(其中 V 是一个变量，而 X 是一个语句或表达式)的意思是“如果 X 正常完成，那么在 X 之后 V 是明确未赋值的”。

> 这个定义的一个更奇怪的结果是“V 在 break 后是明确未赋值的；”总是正确的！因为 break 语句永远不会正常完成，所以如果 break 语句正常完成，那么 V 没有被赋值。(就这一点而言，如果 break 语句正常完成，那么月亮是由绿色奶酪做成的这一说法也毫无疑问是正确的。)

总之，在执行一个语句或表达式之后，变量 V 有四种可能：

- V 是明确赋值的，不是明确未赋值的。

  (流分析规则证明已经发生了对 V 的赋值。)

- V 是明确未赋值的，不是明确赋值的。

  (流分析规则证明尚未发生对 V 的赋值。)

- V 不是明确赋值的，不是明确未赋值的。

  (这些规则不能证明是否发生了对 V 的赋值。)

- V 是明确赋值的，是明确未赋值的。

  (语句或表达式不可能正常完成。)

为了缩短规则，习惯上的缩写“iff”用来表示“当且仅当”。 我们还使用一个缩写约定:如果一个规则包含一个或多个“[un]assigned”，那么它代表两个规则，一个是每次出现“[un]assigned”都被“明确赋值”取代，另一个是每次出现“[un]assigned”都被“明确未赋值”取代。

例如:

- V（未）赋值给空语句，当且仅当它在空语句之前（未）被赋值。

应该理解为代表两条规则:

- V 明确赋值给空语句，当且仅当它在空语句之前被明确赋值。
- V 未明确赋值给空语句，当且仅当它在空语句之前未被明确赋值。

在本章的其余部分，除非另有明确说明，否则我们将使用 V 来表示局部变量或在作用域中的空 final 字段(§6.3)。同样地，我们将用 a、b、c 和 e 来表示表达式，用 S 和 T 来表示语句。我们将使用“a 是 V”这个短语来表示 a 要么是变量 V 的简单名称，要么是由 this 限定的 V 的简单名称(忽略括号)。我们会用"a 不是 V"这个短语来表示"a 是 V"的否定。

循环语句的明确未赋值分析提出了一个特殊的问题。考虑语句 while (*e*) *S*。为了确定 V 在 e 的某个子表达式中是否明确未赋值，我们需要确定 V 在 e 之前是否明确未赋值。有人可能会争辩说，通过类比明确赋值规则(§16.2.10)，V 在 e 之前是明确未赋值的，当且仅当它在 while 语句之前是明确未赋值的。然而，这样的规则不足以满足我们的目的。如果 e 的计算结果为真，则将执行语句 S。稍后，如果 V 由 S 赋值，则在接下来的迭代中，当 e 被求值时，V 将已经被赋值。根据上面建议的规则，可以多次赋值 V，这正是我们试图通过引入这些规则来避免的。

修改后的规则是：“V 在 e 之前是明确未赋值的当且仅当它在 while 语句之前是明确未赋值的，在 S 之后是明确未赋值的”。 然而，当我们为 S 制定规则时，我们发现：“当为 true 时，V 在 S 之前是明确未赋值的当且仅当它在 e 之后是明确未赋值的”。 这导致了一个循环。实际上，V 在循环条件 e 之前是明确未赋值的，只有当它作为一个整体在循环之后未赋值时才是如此!

我们通过对循环条件和循环体的假设分析来打破这种恶性循环。例如，如果我们假设 V 在 e 之前是明确未赋值的(不管 V 是否确实在 e 之前是明确未赋值的)，然后可以证明在 e 之后 V 是明确未赋值的，那么我们知道 e 不赋值 V。这一点更正式地表述为:

假设 V 在 e 之前是明确未赋值的，则 V 在 e 之后是明确未赋值的。

上述分析的变体用于为 Java 编程语言中的所有循环语句定义有充分依据的明确未赋值规则。

16.1 明确赋值和表达式

16.1.1 布尔常量表达式

- 当为 false 时，V（未）赋值给任何值为 true 的常量表达式 (§15.29)。
- 当为 true 时，V（未）赋值给任何值为 false 的常量表达式 (§15.29)。
- 当为 true 时，V（未）赋值给任何值为 true 的常量表达式当且仅当 V 在常量表达式之前（未）被赋值。
- 当为 false 时，V（未）赋值给任何值为 false 的常量表达式当且仅当 V 在常量表达式之

前（未）被赋值。

- V（未）赋值给一个布尔值的常量表达式 e 当且仅当当为 true 时 V（未）赋值给 e，当为 false 时 V（未）赋值给 e。

  这等价于 V（未）赋值给 e 当且仅当 V 在 e 之前（未）被赋值。

  因为值为 true 的常量表达式永远不会有值 false，而值为 false 的常量表达式从来不会有值 true，所以前两个规则是完全满足的。它们有助于分析涉及运算符&& (§16.1.2), || (§16.1.3), ! (§16.1.4), 和 ? : (§16.1.5)的表达式。

16.1.2 条件与操作符 &&

- 当为 true 时，V（未）赋值给 a && b (§15.23) 当且仅当当为 true 时，V（未）赋值给 b。
- 当为 false 时，V（未）赋值给 a && b 当且仅当当为 false 时，V（未）赋值给 a，当为 false 时，V（未）赋值给 b。
- V 在 a 之前（未）被赋值当且仅当 V 在 a && b 之前（未）被赋值。
- V 在 b 之前（未）被赋值当且仅当当为 true 时，V（未）赋值给 a。
- V（未）赋值给 a && b 当且仅当当为 true 时 V（未）赋值给 a && b，当为 false 时 V（未）赋值给 a && b 。

16.1.3 条件或操作符 ||

- 当为 true 时 V（未）赋值给 a || b (§15.24) 当且仅当当为 true 时 V（未）赋值给 a，当为 true 时 V（未）赋值给 b。
- 当为 false 时 V（未）赋值给 a || b 当且仅当当为 false 时 V（未）赋值给 b。
- V 在 a 之前（未）被赋值当且仅当 V 在 a || b 之前（未）被赋值。
- V 在 b 之前（未）被赋值当且仅当当为 false 时，V（未）赋值给 a。
- V（未）赋值给 a || b 当且仅当当为 true 时 V（未）赋值给 a || b，当为 false 时 V（未）赋值给 a || b 。

16.1.4 逻辑补运算符 !

- 当为 true 时 V（未）赋值给 !a (§15.15.6) 当且仅当当为 false 时 V（未）赋值给 a。
- 当为 false 时 V（未）赋值给 !a 当且仅当当为 true 时 V（未）赋值给 a。
- V 在 a 之前（未）被赋值当且仅当 V 在 !a 之前（未）被赋值。
- V（未）赋值给 !a 当且仅当当为 true 时 V（未）赋值给!a ，当为 false 时 V（未）赋值给 !a 。

  这等价于 *V*（未）赋值给!*a* 当且仅当 *V*在 a 之后（未）被赋值。

qingliu

16.1.5 条件操作符？：

假设 b 和 c 是布尔值表达式。

- 当为 true 时 V（未）赋值给 a ? b : c (§15.25) 当且仅当当为 true 时 V（未）赋值给 b，当为 true 时 V（未）赋值给 c。
- 当为 false 时 V（未）赋值给 a ? b : c 当且仅当当为 false 时 V（未）赋值给 b，当为 false 时 V（未）赋值给 c。
- V 在 a 之前（未）被赋值当且仅当 V 在 a ? b : c 之前（未）被赋值。
- V 在 b 之前（未）被赋值当且仅当当为 true 时 V（未）赋值给 a。
- V 在 c 之前（未）被赋值当且仅当当为 false 时 V（未）赋值给 a。
- V（未）赋值给 a ? b : c 当且仅当当为 true 时 V（未）赋值给 a ? b : c ，当为 false 时 V（未）赋值给 a ? b : c 。

假设 b 和 c 不是布尔值的表达式。

- V（未）赋值给 a ? b : c 当且仅当 V（未）赋值给 b 且 V（未）赋值给 c。
- V 在 a 之前（未）被赋值当且仅当 V 在 a ? b : c 之前（未）被赋值。
- V 在 b 之前（未）被赋值当且仅当当为 true 时 V（未）赋值给 a。
- V 在 c 之前（未）被赋值当且仅当当为 false 时 V（未）赋值给 a。

16.1.6 `switch` 表达式

假设 switch 表达式(§15.28)有结果表达式 e_1，…， e_n，它们都是布尔值。

以下规则仅适用于 switch 表达式的 switch 块包含 switch 标签语句组(§14.11.1):

- 当为 true 时 V 明确赋值给 switch 表达式当且仅当对于每个在 switch 块中带有表达式 e(§14.21)的 yield 语句，它可以退出 switch 表达式, 当为 true 时 V 明确赋值给 e。
- 当为 false 时 V 明确赋值给 switch 表达式当且仅当 对于每个在 switch 块中带有表达式 e 的 yield 语句，它可以退出 switch 表达式, 当为 false 时 V 明确赋值给 e。
- 当为 true 时 V 明确未赋值给 switch 表达式当且仅当 对于每个在 switch 块中带有表达式 e 的 yield 语句，它可以退出 switch 表达式, 当为 true 时 V 在 yield 语句之前明确未赋值且在 e 之后明确未赋值。
- 当为 false 时 V 明确未赋值给 switch 表达式当且仅当 对于每个在 switch 块中带有表达式 e 的 yield 语句，它可以退出 switch 表达式,当为 false 时 V 在 yield 语句之前明确未赋值且 V 在 e 之后明确未赋值。
- V 在选择器表达式之前（未）被赋值当且仅当 V 在 switch 表达式之前（未）被赋值。

qinghu

- V 在 switch 块的第一个 switch 标签语句组的第一个语句之前（未）被赋值当且仅当 V 在选择器表达式之后（未）被赋值。
- V 在除了第一个以外的任何 switch 标签语句组的第一条语句之前（未）被赋值当且仅当 V 在选择器表达式之后（未）被赋值且 V 在前面的语句之后（未）被赋值。

以下规则仅适用于 switch 表达式的 switch 块包含 switch 规则(§14.11.1):

- 当为 true 时 V 明确赋值给 switch 表达式当且仅当对每个 switch 规则，以下之一为 true：
  - 它引入一个 switch 规则表达式 e 并且当为 true 时 V 在 e 之后明确赋值。
  - 它引入了一个 switch 规则块 B，对于 B 中包含表达式 e 的每一个 yield 语句，如果它可以退出 switch 表达式，那么当为 true 时，v 在 e 之后明确赋值。
  - 它引入了一个 switch 规则 throw 语句。
- 当为 false 时 V 明确赋值给 switch 表达式当且仅当对每个 switch 规则，以下之一为 true：
  - 它引入一个 switch 规则表达式 e 并且当为 false 时 V 在 e 之后明确赋值。
  - 它引入了一个 switch 规则块 B，对于 B 中包含表达式 e 的每一个 yield 语句，如果它可以退出 switch 表达式，那么当为 false 时，v 在 e 之后明确赋值。
  - 它引入了一个 switch 规则 throw 语句。
- 当为 true 时 V 明确未赋值给一个 switch 表达式当且仅当对每个 switch 规则，以下之一为 true：
  - 它引入一个 switch 规则表达式 e 并且当为 true 时 V 在 e 之后明确未赋值。
  - 它引入了一个 switch 规则块 B，对于在 B 中包含表达式 e 的每个 yield 语句，该语句可以退出 switch 表达式, 在 yield 语句之前 V 明确未赋值且当为 true 时 V 在 e 之后明确未赋值。
  - 它引入了一个 switch 规则 throw 语句。
- 当为 false 时 V 明确未赋值给一个 switch 表达式当且仅当对每个 switch 规则，以下之一为 true：
  - 它引入一个 switch 规则表达式 e 并且当为 false 时 V 在 e 之后明确未赋值。
  - 它引入了一个 switch 规则块 B，对于在 B 中包含表达式 e 的每个 yield 语句，该语句可以退出 switch 表达式, 在 yield 语句之前 V 明确未赋值且当为 false 时 V 在 e 之后明确未赋值。
  - 它引入了一个 switch 规则 throw 语句。
- V 在任何 switch 块的 switch 规则表达式或 switch 规则语句之前（未）被赋值当且仅当 V 在选择器表达式之后（未）被赋值。

假设 switch 表达式有结果表达式 e_1，…， e_n，不是所有的都是布尔值。

- V（未）赋值给 switch 表达式当且仅当以下所有为 true：
  - 在可能退出 switch 表达式的每个 yield 语句之前 V（未）被赋值。
  - 对于每一个 switch 块中的 switch 规则，在由 switch 规则引入的 switch 规则表达式、switch 规则块或 switch 规则 throw 语句之后 V（未）被赋值。
- V 在 switch 表达式的选择器表达式之前（未）被赋值当且仅当 V 在 switch 表达式之前（未）被赋值。
- V 在 switch 块中由 switch 规则引入的 switch 规则表达式、switch 规则块或 switch 规则 throw 语句之前（未）被赋值当且仅当 V 在 switch 表达式的选择器表达式之后（未）被赋值。
- V 在 switch 块中的 switch 标签与剧组的第一个块语句之前（未）被赋值当且仅当以下都为 true：
  - V（未）赋值给 switch 表达式的选择器表达式。
  - 如果 switch 标签语句组不是 switch 块中的第一个，在之前的 switch 标签语句组的最后一个块语句之后 V（未）被赋值。
- V 在 switch 块中不是第一个 switch 标签语句组的块语句之前（未）被赋值当且仅当 V 在前一块语句之后（未）被赋值。

16.1.7 布尔类型的其他表达式

假设 e 是布尔类型的表达式而不是布尔常量表达式、逻辑补表达式 !a、条件与表达式 a && b, 条件或表达式 a || b, 或条件表达式 a ? b : c。

- 当为 true 时 V（未）赋值给 e 当且仅当 V（未）赋值给 e。
- 当为 false 时 V（未）赋值给 e 当且仅当 V（未）赋值给 e。

16.1.8 赋值表达式

考虑赋值表达式 *a* = *b*, *a* += *b*, *a* -= *b*, *a* *= *b*, *a* /= *b*, *a* %= *b*, *a* <<=*b*, *a* >>= *b*, *a* >>>= *b*, *a* &= *b*, *a* | = *b*, 或 *a* ^=*b* (§15.26)。

- 在赋值表达式之后 V 被明确赋值当且仅当以下之一为真：
  - a 是 V, 或者
  - V 在 b 之后明确赋值。
- V 在赋值表达式之后明确未赋值当且仅当 a 不是 V 并且 V 在 b 之后明确未赋值。
- V 在 a 之前（未）赋值当且仅当在赋值表达式之前 V（未）赋值。

xingliu

- V 在 b 之前（未）赋值当且仅当 V 在 a 之后（未）赋值。

  请注意，如果 a 是 V，并且 V 没有在复合赋值(如 a &= b)之前明确赋值，那么编译时错误就一定会发生。上面提到的明确赋值的第一条规则包括析取“a 是 V”，即使是复合赋值表达式，而不仅仅是简单的赋值，所以 V 在代码的后面会被认为是明确赋值的。包括析取“a 是 V”不会影响关于程序是否可接受或将导致编译时错误的二元判定，但它会影响代码中有多少不同的点可能被视为错误，因此在实践中它可以提高错误报告的质量。类似的评论也适用于将“a 不是 V”这一连词列入上文所述的第一条规则中的无限期转让。类似的评论也适用于上述明确未赋值的第一条规则中包含的连词“a 不是 V”。

16.1.9 操作符 ++ 和 --

- V 在++a (§15.15.1), --a (§15.15.2), a++ (§15.14.2), 或 a-- (§15.14.3)之后明确赋值当且仅当 a 是 V 或 V 在操作数表达式之后明确赋值。
- V 在++a, --a, a++, 或 a-- 之后明确未赋值当且仅当 a 不是 V 并且 V 在操作数表达式之后明确未赋值。
- V 在 a 之前（未）赋值当且仅当 V 在 ++a, --a, a++, 或 a--之前（未）赋值。

16.1.10 其他表达式

如果一个表达式不是布尔常量表达式，且不是前增量表达式++a，前减量表达式--a，后增量表达式 a++，后减量表达式 a--，逻辑补码表达式!a，条件与表达式 a && b，条件或表达式 a || b，条件表达式 a ?b: c，赋值表达式，或 lambda 表达式，然后应用以下规则:

- 如果表达式没有子表达式, 在表达式之后 V（未）被赋值当且仅当在表达式之前 V（未）被赋值。

  这种情况适用于文字、名称、this(包括限定和非限定)、不带参数的非限定类实例创建表达式、带有不包含表达式的初始化器的数组创建表达式、超类字段访问表达式、不带参数的非限定和类型限定方法调用表达式、不带参数的超类方法调用表达式、以及超类和类型限定方法引用表达式。

- 如果表达式有子表达式, 在表达式之后 V（未）被赋值当且仅当 V 在它最右边的直接子表达式之后（未）赋值。

  断言变量 V 在方法调用表达式之后是明确未赋值的，这背后有一个微妙的推理。从表面上看，这样的断言并不总是正确的，因为被调用的方法可以执行赋值。但必须记住，对于 Java 编程语言而言，明确未赋值的概念仅适用于空 final 终变量。如果 V 是一个空 final 局部变量，则只有其声明所属的方法才能对 V 执行赋值。如果 V 是一个空 final 字段，那么只有包含 V 声明的类的构造函数或初始化器才能对 V 执行赋值；没有方法可以执行对 V 的赋值。最后，显式构造函数调用（§8.8.7.1）是专门处理的（§16.9）；虽然它们在语法上类似于包含方法调用的表达式语句，但它们不是表达式语句，因此本节的规则不适用于显式构造函数调用。

如果表达式是 lambda 表达式，则应用以下规则:

- 在表达式之后 V（未）被赋值当且仅当在表达式之前 V（未）被赋值。
- 在表达式或 lambda 体的块之前 V 被明确赋值当且仅当在 lambda 表达式之前 V 被明确赋值。

  任何规则都不允许 V 在 lambda 体之前明确未赋值。这是经过设计的：在 lambda 正文结束之前明确未

赋值的变量可能会在稍后被赋值，因此我们不能得出在执行该正文时变量未赋值的结论。

对于表达式 x 的任何直接子表达式 y，其中 x 不是 lambda 表达式, V 在 y 之前（未）被赋值当且仅当以下之一为 true:

- y 是 x 最左边的直接子表达式 V 在 x 之前（未）被赋值。
- y 是二元运算符的右操作数 并且 V 在左操作数之后（未）被赋值。
- x 是一个数组访问，y 是括号内的子表达式，V 在括号前的子表达式之后（未）被赋值。
- x 是一个主方法调用表达式，y 是方法调用表达式中的第一个参数表达式，V 在计算目标对象的主表达式之后（未）被赋值。
- x 是方法调用表达式或类实例创建表达式;y 是一个参数表达式，但不是第一个;V 在 y 左边的参数表达式之后（未）被赋值。
- x 是一个限定类实例创建表达式，y 是类实例创建表达式中的第一个参数表达式，V 在计算限定对象的主表达式之后（未）被赋值。
- x 是一个数组创建表达式；y 是一个维数表达式，但不是第一个;而 V 在 y 左边的维数表达式之后（未）被赋值。
- x 是一个通过数组初始化器初始化的数组创建表达式;y 是 x 中的数组初始化器;而 V 在 y 左边的维数表达式之后（未）被赋值。

dingliu

16.2 明确赋值和语句

16.2.1 空语句

• V 在空语句(§14.6)之后（未）被赋值当且仅当它在空语句之前（未）被赋值。

16.2.2 块

- 在 V 作用域内的任何方法的主体块(§14.2)和在 V 作用域内声明的任何类的声明块(§14.2)之前，对一个空 final 成员字段 V 明确赋值(而且不是明确未赋值)。
- 局部变量 V 在声明 V 的构造函数、方法、实例初始化器或静态初始化器的代码块之前是明确未赋值的(而且也不是明确赋值的)。
- 设 C 是在 V 的作用域内声明的类。然后 V 在 C 中声明的任何构造函数、方法、实例初始化器或静态初始化器的块之前明确赋值，当且仅当 V 在 C 的声明之前明确赋值。.

  请注意，没有任何规则允许我们得出结论:在 C 中声明的任何构造函数、方法、实例初始化器或静态初始化器的主体块之前，V是明确未赋值的。我们可以非正式地得出结论，在C中声明的任何构造函数、方法、实例初始化器或静态初始化器的主体块之前，V 都不是明确未赋值的，但没有必要显式地声明这样的规则。

- V 在一个空块之后（未）被赋值当且仅当 V 在空块之前（未）被赋值。

• V 在一个非空块之后（未）被赋值当且仅当 V 在块中最后一个语句之后（未）被赋值。

• V 在块中第一个语句之前（未）被赋值当且仅当 V 在块前（未）被赋值。

• V 在块中任何其他语句 S 之前（未）被赋值当且仅当 V 在在块中紧跟在 S 之前的语句之后（未）被赋值。

我们说 V 在块 B 中的任何位置明确未赋值当且仅当：

• V 在 B 之前明确未赋值。

• 在每个发生在 B 中的赋值表达式 V = e, V += e, V -=e, V *= e, V /= e, V %= e, V <<= e, V >>= e, V >>>= e, V &= e, V |= e, 或 V ^= e 中的 e 之后 V 被明确赋值。

• 在每个发生在 B 中的表达式++V, --V, V++, 或 V--之前 V 被明确赋值。

这些条件是违反直觉的，需要一些解释。考虑一个简单的赋值 *V = e*。如果 V 在 e 之后被明确赋值，那么以下之一为真：

• 赋值发生在死代码中，V 是明确赋值的。在这种情况下，赋值实际上不会发生，我们可以假设 V 没有被赋值表达式赋值。或者：

• V 已经由 e 之前的表达式赋值。在这种情况下，当前赋值将导致编译时错误。

因此，我们可以得出这样的结论:如果一个程序满足了这些条件，并且没有导致编译时错误，那么在 B 中对 V 的任何赋值实际上都不会在运行时发生。

16.2.3 局部类和接口声明

• V 在局部类或接口声明后（未）被赋值(§14.3) 当且仅当 V 在局部类或接口声明之前（未）被赋值。

16.2.4 局部变量声明语句

• V 在不包含变量初始化器的局部变量声明语句(§14.4.2)之后（未）被赋值当且仅当 V 在局部变量声明语句之前（未）被赋值。

• V 在包含至少一个变量初始化器的局部变量声明语句之后明确赋值当且仅当要么 V 在局部变量声明语句的最后一个变量初始化器之后赋值，要么声明中的最后一个变量初始化器在声明 V 的声明符中。

• 在包含至少一个变量初始化器的局部变量声明语句之后，V 是明确未赋值的当且仅当在局部变量声明语句的最后一个变量初始化器之后，V 是明确未赋值的，声明中的最后一个变量初始化器也不在声明 V 的声明符中。

• 在局部变量声明语句的第一个变量初始化器之前 V（未）被赋值当且仅当在局部变量声明语句之前 V（未）被赋值。

• 在局部变量声明语句中除第一个变量初始化器 e 之外的任何变量初始化器 e 之前明确赋值 V 当且仅当 要么在 e 左边的变量初始化器之后明确赋值 V，要么 e 左边的初始化器表达式在声明 V 的声明符中。

cnglu

• 在局部变量声明语句中除第一个变量初始化器 e 之外的任何变量初始化器 e 之前，v 是明确未赋值的当且仅当在 e 左边的变量初始化器之后，V 是明确未赋值的，并且 e 左边的初始化器表达式不在声明 V 的声明符中。

16.2.5 标签语句

• V（未）赋值给标签语句 L : S (其中 L 是一个标签) (§14.7)当且仅当 V（未）赋值给 S 并且 V 在每个 break 语句之前（未）被赋值，这个 break 语句可以退出标签语句 L: S。

• V 在 S 之前（未）被赋值当且仅当 V 在 L: S 之前（未）被赋值。

16.2.6 表达式语句

• V（未）赋值给表达式语句 e; (§14.8) 当且仅当它（未）赋值给 e。

• V 在 e 之前（未）被赋值当且仅当它在 e;之前（未）被赋值。

16.2.7 `if` 语句

以下规则适用于一个语句 if (*e*) *S* (§14.9.1):

• V（未）赋值给 if (*e*) *S* 当且仅当 V（未）赋值给 *S* 并且当为 false 时 V（未）赋值给 *e* 。

• V 在 e 之前（未）被赋值当且仅当 V 在 if (e) S 之前（未）被赋值。

• V 在 S 之前（未）被赋值当且仅当当为 true 时 V（未）赋值给 e。

以下规则适用于一个语句 if (e) *S* else *T* (§14.9.2):

• V（未）赋值给 if (e) S else T 当且仅当 V（未）赋值给 S 并且 V（未）赋值给 T。

• V 在 e 之前（未）被赋值当且仅当 V 在 if (e) S else T 之前（未）被赋值。

• V 在 S 之前（未）被赋值当且仅当当为 true 时 V（未）赋值给 e。

• V 在 T 之前（未）被赋值当且仅当当为 false 时 V（未）赋值给 e。

16.2.8 `assert` 语句

下面的规则同时适用于 assert e_1 语句和 assert e_1 : e_2 语句(§14.10):

• V 在 e_1 之前（未）被赋值当且仅当 V 在 assert 语句之前（未）被赋值。

• V（未）明确赋值给 assert 语句当且仅当 V 在 assert 语句之前被明确赋值。

• 在 assert 语句之后，V 明确未赋值当且仅当在 assert 语句之前 V 明确未赋值并且当为 true 时 V 在 e_1 之后明确未赋值。

下面的规则适用于语句 assert e_1 : e_2:

• V 在 e_2 之前（未）被赋值当且仅当当为 false 时 V（未）赋值给 e_1 。

qinglin

16.2.9 `switch` 语句

- V（未）赋值给 switch 语句(§14.11) 当且仅当以下所有为 true:
  - V 在每个 break 语句(§14.15)之前（未）被赋值，break 语句可以退出 switch 语句。
  - 对于 switch 块中的每个 switch 规则(§14.11.1)，V 在 switch 规则表达式、switch 规则块或由 switch 规则引入的 switch 规则 throw 语句之后（未）被赋值。
  - 如果在 switch 块中有一个 switch 标签语句组，那么在最后一个 switch 标签语句组的最后一个块语句之后（未）分配 V。
  - 如果在 switch 块中没有 default 标签，或者 switch 块以 switch 标签后跟}分隔符结束，那么在选择器表达式之后（未）赋值 V。
- V 在 switch 语句的选择器表达式之前（未）被赋值当且仅当 V 在 switch 语句之前（未）被赋值。
- V 在 switch 块中的 switch 规则引入的 switch 规则表达式、switch 规则块或 switch 规则 throw 语句之前（未）被赋值当且仅当 V 在 switch 语句的选择器表达式之后（未）被赋值。
- V 在 switch 块中 switch 标签语句组的第一个块语句之前（未）被赋值当且仅当以下都为 true:
  - V（未）赋值给 switch 语句的选择器表达式。
  - 如果 switch 标签语句组不是 switch 块中的第一个，V（未）赋值给之前的 switch 标签语句组的最后一个块语句。
- V 在 switch 块中不是第一个 switch 标签语句组的块语句之前（未）被赋值当且仅当 V（未）赋值给之前的块语句。

16.2.10 `while` 语句

- V（未）赋值给 while (e) S (§14.12) 当且仅当当为 false 时 V（未）赋值给 e 并且 V 在以 while 语句为 break 目标的每个 break 语句之前（未）被赋值。
- V 在 e 之前明确赋值当且仅当 V 在 while 语句之前明确赋值。
- V 在 e 之前明确未赋值当且仅当以下所有为 true:
  - V 在 while 语句之前明确未赋值。
  - 假设 V 在 e 之前明确未赋值，V 在 S 之后明确未赋值。
  - 假设 V 在 e 之前明确未赋值，在每个以 while 语句为 continue 目标的 continue 语句之前，V 明确未赋值。
- V 在 S 之前（未）赋值当且仅当当为 true 时 V（未）赋值给 e。

qingliu

16.2.11 do 语句

- V（未）赋值给 do S while (e); (§14.13) 当且仅当当为 false 时 V（未）赋值给 e 并且 V 在 do 语句为 break 目标的每个 break 语句之前（未）被赋值。
- V 在 S 之前明确赋值当且仅当 V 在 do 语句之前明确赋值。
- V 在 S 之前明确未赋值当且仅当以下所有为 true：
  - V 在 do 语句之前明确未赋值。
  - 假设 V 在 S 之前明确未赋值，当值为 true 时 V 在 e 之后明确未赋值。
- V 在 e 之前（未）赋值当且仅当 V（未）赋值给 S 并且 V 在 do 语句为 continue 目标的每个 continue 语句之前（未）被赋值。

16.2.12 for 语句

这里的规则涵盖了语句的基本内容(§14.14.1)。由于增强的 for 语句(§14.14.2)是通过转换为基本的 for 语句来定义的，因此不需要为它提供特殊的规则。

- V（未）赋值给一个 for 语句当且仅当以下都为 true:
  - 要么条件表达式不存在，要么当为 false 时 V（未）赋值给条件表达式。
  - V 在以 for 语句为 break 目标的每个 break 语句之前（未）被赋值。
- V 在 for 语句初始化部分之前（未）被赋值当且仅当 V 在 for 语句之前（未）被赋值。
- V 在 for 语句的条件部分之前被明确赋值当且仅当 V 在 for 的初始化部分之后被明确赋值。
- V 在 for 语句的条件部分之前明确未赋值当且仅当以下都为 true：
  - V 在 for 语句的初始化部分之后明确未赋值。
  - 假设 V 在 for 语句的条件部分之前明确未赋值，V 在 for 语句的增长部分之后明确未赋值。
- V 在包含语句之前（未）被赋值当且仅当以下之一为 true:
  - 条件表达式存在并且当为 true 时 V（未）赋值给条件表达式。
  - 条件表达式不存在并且在 for 语句的条件部分之前 V（未）被赋值。
- 在 for 语句的增长部分之前 V（未）被赋值当且仅当 V（未）赋值给包含的语句并且 V 在每个以 for 语句为 continue 目标的 continue 语句之前（未）被赋值。

16.2.12.1 for 语句的初始化部分

- 如果 for 语句的初始化部分是局部变量声明语句，则适用§16.2.4 的规则。

qinglu

- 否则，如果初始化部分为空，那么 V（未）赋值给初始化部分当且仅当 V 在初始化部分之前（未）被初始化。
- 否则，三条规则适用:
  - V（未）赋值给初始化部分当且仅当 V（未）赋值给初始化部分的最后一个表达式语句。
  - V 在初始化部分的第一个表达式之前（未）被初始化当且仅当 V 在初始化部分之前（未）被赋值。
  - V 在表达式语句 S 之前（未）被赋值，而不是在初始化部分的第一个位置当且仅当 V 在紧接 S 之前的表达式语句之后（未）被赋值。

16.2.12.2 for 语句的递增部分

- 如果 for 语句的递增部分为空，则 V（未）赋值给递增部分当且仅当 V 在递增部分之前（未）被赋值。
- 否则，三条规则适用:
  - V（未）赋值给递增部分当且仅当 V（未）赋值给递增部分的最后一个表达式语句。
  - V 在递增部分的第一个表达式语句之前（未）被赋值当且仅当 V 在递增部分之前（未）被赋值。
  - V 在表达式语句 S 之前（未）被赋值，而不是递增部分中的第一个语句当且仅当 V 在紧接 S 之前的表达式语句之后（未）被赋值。

16.2.13 `break`, `yield`, `continue`, `return`, 和 `throw` 语句

- 按照惯例，我们认为 V（未）赋值给任何 break, yield, continue, return, 或 throw 语句(§14.15, §14.21, §14.16, §14.17, §14.18)。

  变量"（未）赋值给"语句或表达式的概念实际上意味着"在语句或表达式正常完成后（未）被赋值"。因为 break、yield、continue、return 或 throw 语句永远不会正常完成，所以它完全满足了这个概念。

- 在带有表达式 e 的 yield 语句、return 语句或 throw 语句中，V 在 e 之前（未）被赋值当且仅当 V 在 yield, return, 或 throw 语句之前（未）被赋值。

16.2.14 `synchronized` 语句

- V（未）赋值给 synchronized (e) S (§14.19) 当且仅当 V（未）赋值给 S。
- V 在 e 之前（未）被赋值当且仅当 V 在语句 synchronized (e) S 之前（未）被赋值。
- V 在 S 之前（未）被赋值当且仅当 V（未）赋值给 e。

16.2.15 `try` 语句

这里的规则涵盖了 try-catch 和 try-catch-finally 语句(§14.20.1、§14.20.2)。由于 try-with-

resources 语句(§14.20.3)是通过翻译成 try-catch-finally 语句来定义的，因此不需要为它提供特殊的规则。

这些规则适用于每一个 try 语句(§14.20)，无论它是否有 finally 块:

- V 在 try 块之前（未）被赋值当且仅当 V 在 try 语句之前（未）被赋值。
- V 在 catch 块之前明确赋值当且仅当 V 在 try 块之前明确赋值。
- V 在 catch 块之前明确未赋值当且仅当以下所有为 true:
  - V 在 try 块后明确未赋值。
  - V 在属于 try 块的每个 return 语句之前明确未赋值。
  - 在属于 try 块的形式为 throw e 的每个语句中，V 在 e 之后明确未赋值。
  - 在 try 块中出现的每个 assert 语句之后，V 明确未赋值。
  - 在属于 try 块且其 break 目标包含（或是）try 语句的每个 break 语句之前，V 明确未赋值。
  - V 在属于 try 块且其 continue 目标包含 try 语句的每个 continue 语句之前明确未赋值的。

qinghu

如果 try 语句没有 finally 块，则此规则也适用:

- V（未）赋值给 try 语句当且仅当 V（未）赋值给 try 块并且 V（未）赋值给 try 语句中的每个 catch 块。

如果 try 语句没有 finally 块，则这些规则也适用:

- V 在 try 语句之后明确赋值当且仅当以下至少一个为 true:
  - V 在 try 块之后明确赋值并且 V 在 try 语句的每个 catch 块之后明确赋值。
  - V 在 finally 块之后明确赋值。
- V 在 try 语句之后明确未赋值当且仅当 V 在 finally 块之后明确未赋值。
- V 在 finally 块之前明确赋值当且仅当 v 在 try 语句之前明确赋值。
- V 在 finally 块之前明确未赋值当且仅当以下所有为 true:
  - V 在 try 块之后明确未赋值。
  - V 在属于 try 块的每个 return 语句之前明确未赋值。
  - 在属于 try 块的形式为 throw e 的每个语句中，V 在 e 之后明确未赋值的。
  - 在 try 块中出现的每个 assert 语句之后，V 明确未赋值。
  - 在属于 try 块且其 break 目标包含（或是）try 语句的每个 break 语句之前，V 明确

未赋值。

- V 在属于 try 块且其 continue 目标包含 try 语句的每个 continue 语句之前明确未赋值。
- V 在 try 语句的每个 catch 块之后明确未赋值。

16.3 明确赋值和参数

- 方法或构造函数的形式参数 v(§8.4.1、§8.8.1)在方法或构造函数体之前明确赋值(而且不是明确未赋值的)。
- catch 子句的异常参数 v(§14.20)在 catch 子句正文之前被明确赋值(而且不是明确未赋值)。

16.4 明确赋值和数组初始化器

- V（未）赋值给空数组初始化器 (§10.6)当且仅当 V 在空数组初始化器之前（未）被赋值。
- V（未）赋值给非空数组初始化器当且仅当 V（未）赋值给数组初始化器中的最后一个变量初始化器。
- V 在数组初始化器的第一个变量初始化器之前（未）被赋值当且仅当 v 在数组初始化器之前（被）赋值。
- V 在数组初始化器的任何其他变量初始化器 e 之前（未）被赋值当且仅当 V（未）赋值给数组初始化器中 e 的左侧的变量初始化器之后。

qingqiu

16.5 明确赋值和枚举常量

在枚举常量(§8.9.1)之前，确定变量何时明确赋值或明确未赋值的规则在§16.8 中给出。

这是因为枚举常量本质上是用类实例创建表达式(§15.9)初始化的静态 final 字段(§8.3.1.1、§8.3.1.2)。

- 在 V 的作用域内声明的不带参数的枚举常量的类体声明之前明确赋值 V 当且仅当 V 在枚举常量之前明确赋值。
- 在 V 的作用域内声明的带参数的枚举常量的类体声明之前，明确赋值 V 当且仅当 V 在枚举常量的最后一个参数表达式之后明确赋值。

枚举常量的类体内的任何构造的明确赋值/未赋值状态由类的通常规则管理。

- 在枚举常量的第一个参数之前 V（未）被赋值当且仅当在枚举常量之前它（未）被赋值。
- V 在 y 之前（未）被赋值 (枚举常量的一个参数，但不是第一个) 当且仅当 V（未）赋值给 y 左边的参数。

16.6 明确赋值和匿名类

- V 在 V 的作用域内声明的匿名类声明(§15.9.5)之前被明确赋值当且仅当 V 明确赋值为声明匿名类的类实例创建表达式。

  应该清楚的是，如果匿名类是由枚举常量隐式定义的，则适用§16.5 的规则。

16.7 明确赋值和成员类和接口

设 C 是一个类，V 是 C 的空 final 字段，那么：

- 在 C 的任何成员类或接口(§8.5、§9.5)的声明之前，V 是明确赋值的(而且不是明确未赋值的)。

设 C 为 V 的作用域内声明的类。那么：

- V 在 C 的成员类或接口的声明之前被明确赋值当且仅当 V 在 C 的声明之前被明确赋值。

16.8 明确赋值和静态初始化器

设 C 为 V 的作用域内声明的类。那么：

- V 在 C 的枚举常量(§8.9.1)或静态变量初始化器(§8.3.2)之前被明确赋值当且仅当 V 在 C 的声明之前被明确赋值。

  请注意，没有规则允许我们得出结论，在静态变量初始化器或枚举常量之前，V 是明确未赋值的。我们可以非正式地得出结论，在 C 的任何静态变量初始化器之前，V 不是明确未赋值的，但是没有必要明确地声明这样的规则。

设 C 为一个类, V 为 C 中声明的空静态 final 成员字段。那么：

- V 在最左边的枚举常量、静态初始化器(§8.7)或 C 的静态变量初始化器之前明确未赋值(而且也不是明确赋值)。
- V 在除了最左边外的枚举常量、静态初始化器或 C 的静态变量初始化器之前（未）被赋值当且仅当 V（未）赋值给前面的枚举常量、静态初始化器或 C 的静态变量初始化器。

设 C 是一个类，V 是 C 的一个空的静态 final 成员字段，在 C 的超类中声明，然后：

- V 在 C 的每个枚举常量之前明确赋值(而且也不是明确未赋值)。
- V 在 C 的静态初始化器的块之前明确赋值(而且也不是明确未赋值)。
- 在 C 的每个静态变量初始化器之前，V 被明确赋值(而且也不是明确未赋值)。

16.9 明确赋值、构造函数和实例初始化器

设 C 为 V 的作用域内声明的类。那么：

- V 在 C 的实例变量初始化器之前被明确赋值当且仅当 V 在 C 的声明之前被明确赋值。

  请注意，没有规则允许我们得出结论，在实例变量初始化器之前，V 是明确未赋值的。我们可以非正式地得出结论，在 C 的任何实例变量初始化器之前，V 不是明确未赋值的，但没有必要显式地声明这样的规则。

设 C 为一个类, 设 V 是 C 的一个空的 final 非静态成员字段，在 C 中声明。那么:

- 在最左边的实例初始化器(§8.6)或 C 的实例变量初始化器之前，V 是明确未赋值的(而且不是明确赋值)。
- V 在除了最左边以外的 C 的实例初始化器或实例变量初始化器之前（未）被赋值当且仅当 V（未）赋值为 前面的实例初始化器或 C 的实例变量初始化器。

以下规则适用于类 C 的构造函数(§8.8.7):

- 在替代构造函数调用（§8.8.7.1）后，V 被明确赋值（而且也不是明确未赋值）。
- 在显式或隐式超类构造函数调用之前（§8.8.7.1），V 是明确未赋值的（而且也不是明确赋值的）。
- 如果 C 没有实例初始化器或实例变量初始化器，则在显式或隐式超类构造函数调用之后，V 不会被明确赋值（而且是明确未赋值的）。
- 如果 C 至少有一个实例初始化器或实例变量初始化器，那么 V（未）赋值给显式或隐式超类构造函数调用当且仅当 V（未）赋值给最右边的实例初始化器或 C 的实例变量初始化器。

设 C 是一个类，V 是 C 的一个空的 final 成员字段，在 C 的超类中声明。那么:

- 在 C 的构造函数或实例初始化器主体的块之前，V 被明确赋值（而且也不是明确未赋值）。
- V 在 C 的每个实例变量初始化器之前被明确赋值（而且也不是明确未赋值）。