

# Irenka Search Query Specification

Ver. 0.2.0

2007/11/23

The Ashikunep Kotan

# もくじ

---

第 1 章 イントロダクション.....	1
第 2 章 文法.....	2
第 1 節 字句構造.....	2
第 2 節 構文.....	6
第 3 章 クエリ.....	7
第 1 節 制約による構造の特定.....	7
第 2 節 プレースホルダと検索結果.....	7
第 4 章 制約.....	8
第 1 節 制約の評価.....	8
第 2 節 関係制約.....	9
第 3 節 要素型制約.....	14
第 5 章 式.....	16
第 1 節 式の評価.....	16
第 2 節 項.....	16
第 3 節 プレースホルダ.....	17
第 4 節 即値.....	18
第 5 節 プロパティ参照.....	19
第 6 節 リスト構築.....	22
第 7 節 リスト参照.....	22
第 6 章 即値.....	24
第 1 節 定数リテラル.....	24
第 2 節 修飾子.....	25
第 3 節 基本型.....	26
第 4 節 プリミティブ型の配列型.....	26
第 5 節 リンク参照.....	27

第7章 検索 .....	33
第1節 検索対象 .....	33
第2節 検索処理 .....	35
第3節 検索例外 .....	37
付録A. Irenka Search QueryのBNF .....	39
付録B. 利用可能なプロパティ .....	44
参考文献 .....	47
用語索引 .....	48

## 第1章 イントロダクション

---

Irenka Search Query はプログラムに含まれる特定の構造を表現するためのクエリ言語で、その表現に即したプログラム片をプロジェクトから検出することができます。この言語が表現可能なクエリは、プログラムの字句構造や構文構造ではなく、プログラムの意味構造を対象としています。ユーザは、同一の意味を持ちながら表記の異なるプログラムについて意識することなく、意味のみを考慮したクエリを記述することができます。

Irenka Search Query が対象とする検索空間は、プログラムを Irenka によってオブジェクト化したモデルを対象とし、これを Irenka DOM と呼びます。このモデルは DOM 要素というプログラムの様々な単位の構成要素の集まりからなり、そしてモデル全体は開発プロジェクトに含まれるプログラム全体を指します。クエリによる検索結果は Irenka DOM に含まれる個々の DOM 要素の集まりで、検索結果に含まれる DOM 要素の特徴を「制約」によって記述し、検索結果を絞り込むことができます。この制約はクエリ中に複数記述することができ、それらの制約を同時に満足するようなモデル内の部分空間を検出します。

Irenka の主要機能である Hack では、この Irenka Search Query で記述されたクエリ(Search Query)が使用されます。Hack はプログラムの監査と操作などを行うことができるユーザプログラムで、その監査対象や操作対象の検出に Irenka Search Query は力を発揮します。DOM 要素の走査や性質の判定など、検出に関する処理はクエリ言語で記述ことができ、Hack の本体であるプログラムの監査や操作などの実際の操作と処理を分離することができるようになります。

本文書では、Irenka Search Query 言語の仕様について詳しく紹介します。検索対象の Irenka DOM については Irenka DOM Specification、開発環境である Irenka Studio の使い方やクエリのサンプルについては Irenka Studio Users' Guide を参照してください。また、本文書では、Search Query と Irenka Search Query という 2 種類の用語を使用する場合があります。前者の Search Query は Hack の中に実際に記述されたクエリを表し、後者の Irenka Search Query は Search Query 言語の仕様、または言語そのものを表します。

## 第2章 文法

---

### 第1節 字句構造

Irenka Search Queryが利用する字句構造は、Java言語仕様に規定された字句構造と非常に似通っています。キーワード(第3項)と演算子(第11項)が大きく異なることを除けば、ほぼJava言語仕様のサブセットとして定義されます。

#### 第1項 空白文字

```
Whitespace: One of
           SP(¥u0020) HT(¥u0009) CR(¥u000d) LF(¥u000a)
```

##### 定義 1. 空白文字の一覧

Irenka Search Query内に記述された空白文字(定義 1)はすべて読み飛ばされます。ただし、文字リテラル(第7項)や文字列リテラル(第8項)内で記述された空白文字は、読み飛ばされることなくそのリテラルの中に文字として含まれます。

#### 第2項 コメント

```
EndOfLineComment:
  "//" ~( LF | CR )* ( LF | CR )
```

##### 定義 2. コメントの構造

Irenka Search Queryではコメントに一行コメント(JLS3-3.7, *EndOfLineComment*)を利用することができます(定義 2)。コメントは空白(第1項)と同様に取り扱われ、Search Queryの実行結果に影響を及ぼしません。

#### 第3項 識別子

```
<NAME>:
  JavaChar ( JavaChar | JavaDigit )*
JavaChar: One of
  A-Z a-z _ $
JavaDigit: One of
  0 1 2 3 4 5 6 7 8 9
```

##### 定義 3. 識別子の形式

Irenkaで利用可能な識別子は定義 3にある形式で表現されます。これは、Java言語仕様にある識別子(JLS3-3.8, *Identifier*)のうち、ASCII文字のみで構成可能なものを抜き出したものです。ただし、キーワード(定義 4)、真偽リテラル(第 6 項第 9 項)のいずれかと衝突する文字列は Irenka Search Queryの識別子として利用することはできません。

定義 4は識別子として利用することができないキーワードの一覧です。これらはJava言語仕様で定義されるキーワード(JLS3-3.9, *Keyword*)の一部に、包括制約(第 4 章第 2 節第 2 項)を行う演算子"in"を加えたものです。

```
abstract boolean byte char double extends final float in int long
native private protected public short static strictfp super
synchronized transient void volatile
```

#### 定義 4. キーワードの一覧

識別子はプレースホルダ(第 5 章第 3 節)の名前、プロパティ(第 5 章第 5 節)の名前、宣言型(第 6 章第 5 節第 1 項)の名前の一部などに利用されます。たとえば、"self", "method", "stmt", "leftHandSide"などはいずれも有効な識別子です。

### 第4項 整数リテラル

```
<INTEGER_LITERAL>:
    IntegerValue IntegerSuffix?
IntegerValue:
    0
    NonZeroDigit Digit*
    0x HexDigit+
    0 OctDigit+
NonZeroDigit: One of
    1 2 3 4 5 6 7 8 9
Digit: One of
    0 NonZeroDigit
HexDigit: One of
    Digit a b c d e f A B C D E F
OctDigit: One of
    0 1 2 3 4 5 6 7
IntegerSuffix: One of
    l L
```

#### 定義 5. 整数リテラルの形式

整数リテラルはJava言語仕様で定義される整数リテラル(JLS3-3.10.1, *IntegerLiteral*)と同等の形式で、int型とlong型の値を表現することができます(定義 5)。

## 第5項 浮動小数点数リテラル

```

<REAL_LITERAL>:
    Digit+ . Digit* ExpormentPart? RealSuffix?
    . Digit+ ExpormentPart? RealSuffix?
    Digit+ ExpormentPart? RealSuffix
    0x HexDigit+ .? BinaryExpormentPart RealSuffix?
    0x HexDigit* . HexDigit+ BinaryExpormentPart RealSuffix?

ExpormentPart:
    e SignedInteger
    E SignedInteger

BinaryExpormentPart:
    p SignedInteger
    P SignedInteger

SignedInteger:
    ( + | - ) Digit+

RealSuffix: One of
    f F d D
  
```

### 定義 6. 浮動小数点数の形式

浮動小数点数リテラルはJava言語仕様で定義される浮動小数点数リテラル(JLS3-3.10.2, *FloatingPointLiteral*)と同等の形式で、float型とdouble型の値を表現することができます(定義 6)。

## 第6項 真偽リテラル

```

<BOOLEAN_LITERAL>: One of
    true false
  
```

### 定義 7. 真偽リテラルの一覧

真偽リテラルはJava言語仕様で定義される真偽リテラル(JLS3-3.10.3, *BooleanLiteral*)と同等で、boolean型の値を表現することができます(定義 7)。

## 第7項 文字リテラル

```

<CHARACTER_LITERAL>:
    ' SingleCharacter '
    ' EscapeSequence '
SingleCharacter:
    ~( ' | ¥ )
EscapeSequence:
    ¥ ( b | t | n | f | r | " | ' | ¥ )
    ¥ u HexDigit HexDigit HexDigit HexDigit
    ¥ ( 0 | 1 | 2 | 3 )? OctDigit? OctDigit

```

### 定義 8. 文字リテラルの形式

文字リテラルはJava言語仕様で定義される文字リテラル(JLS3-3.10.4, *CharacterLiteral*)と同等の形式で、char型の値を表現することができます(定義 8)。

## 第8項 文字列リテラル

```

<STRING_LITERAL>:
    " StringCharacter* "
StringCharacter:
    ~( " | ¥ )
    EscapeSequence

```

### 定義 9. 文字列リテラルの形式

文字列リテラルはJava言語仕様で定義される文字列リテラル(JLS3-3.10.5, *StringLiteral*)と同等の形式で、java.lang.String型の値を表現することができます(定義 9)。

## 第9項 null リテラル

```

<NULL_LITERAL>: Just
    null

```

### 定義 10. null リテラル

nullリテラルはJava言語仕様で規定されるnullリテラル(JLS3-3.10.7, *NullLiteral*)と同等で、null参照を表現することができます(定義 10)。

## 第10項 区切り子

```
Separator: One of
( ) { } [ ] # , . :
```

### 定義 11. セパレータ文字の一覧

区切り子(セパレータ)はJava言語仕様で規定されるセパレータ(JLS3-3.11, *Separator*)に文字#を加えたもので、区切り文字として利用されます(定義 11)。他の構文の一部として利用されるため、それぞれに意味はありません。

## 第11項 演算子

```
<OPERATOR>: One of
= in == !=
< > <= >=
=~ !~
```

### 定義 12. 演算子の一覧

演算子は定義 12に列挙される記号列のいずれかで、それぞれ異なる意味を持ちます。解釈については第 4 章第 2 節に説明があります。

## 第2節 構文

Irenka Search Queryについての全体の構文は、付録Aにあります。それぞれの構造については、第 2 章(クエリ)、第 3 章(制約)、第 4 章(式)を参照してください。

## 第3章 クエリ

```
<QUERY>
 ::= "@when"? <CONSTRAINT>*
```

### 定義 13. クエリの構造

Irenka Search Query 言語におけるクエリ(QUERY)は、定義 13のBNFで表現されます。クエリは複数の制約(CONSTRAINT)からなり、Irenka Search Queryはクエリに含まれるすべての制約を同時に満足するような構造を検索することができます。制約については第 4 章を参照してください。

Irenka Search Query 言語で記述されたクエリは、Irenka によってモデル化されたプログラム (Irenka DOM)を取り扱うことができます。Irenka DOM はプログラムの意味的な構造を元にしたモデルで、DOM 要素というプログラムの様々な単位の構成要素の集まりからなり、全体として開発プロジェクトに含まれるすべてのプログラムを含みます。制約では様々な方法でそれぞれの DOM 要素が持つ特徴を記述することができます。DOM 要素について詳しくは Irenka DOM Specification に説明があります。そちらを参照してください。

## 第1節 制約による構造の特定

クエリの目的は、与えられた検索対象(第 7 章第 1 節)の中からクエリに含まれるすべての制約(第 4 章)を満足する値の組み合わせを検出することです。クエリに含まれる一つ一つの制約は単純な表現が可能で、それらを複数組み合わせることによって複雑な検索条件を表現することができます。

また、それぞれの制約は、2つの式(第 5 章)とその間に記述された1つの制約演算子(第 4 章第 2 節)から構成されています。制約演算子は2つの式の間を記述することができ、2つの式に指定した演算を適用した結果が真である場合のみ制約が満足されます。

## 第2節 プレースホルダと検索結果

クエリの中には、プレースホルダ(第 5 章第 3 節)という、検索対象の一部を保存する領域を宣言することができます。それぞれのプレースホルダは名前を持っており、クエリの処理中に式の評価結果などの任意の検索対象が格納されます。

クエリは、そこで宣言されたプレースホルダに様々な要素を格納し、すべての制約を満足する組み合わせをすべて検出します。すべての制約を満足する組み合わせが発見された場合、その時のクエリ上に存在するそれぞれのプレースホルダには、クエリに記述された全ての制約を同時に満足するような要素の組み合わせが格納されています。そして、各プレースホルダに格納されたそれぞれの値が検索対象に対するクエリの検索結果となり、それぞれのプレースホルダの名前を利用して、検索結果に含まれる値を参照することができます。

## 第4章 制約

```

<CONSTRAINT>
  ::= <RELATIONAL_CONSTRAINT>
     | <ELEMENT_TYPE_CONSTRAINT>

<RELATIONAL_CONSTRAINT>
  ::= <EXPRESSION> <OPERATOR> <EXPRESSION>

<ELEMENT_TYPE_CONSTRAINT>
  ::= <EXPRESSION> ":" <ELEMENT_TYPES>

```

### 定義 14. 制約の構造

制約は 2 つの式(EXPRESSION)と 2 つの式の間記述された制約演算子(OPERATOR)からなる関係制約(RELATIONAL\_CONSTRAINT)と、1 つの式(EXPRESSION)と要素の型の列挙(ELEMENT\_TYPES)となる要素型制約(ELEMENT\_TYPE\_CONSTRAINT)があります。前者は両式の間記述された関係を表し、後者は式が取りうる型を表しますが、どちらもクエリの制約 1 つ分を表します。クエリの実行は制約の評価(第 1 節)によって行われ、評価の方法は演算子によって規定されます。

制約に含まれる式については第 5 章、制約演算子については第 2 節、要素の型については第 3 節を参照してください。

### 第1節 制約の評価

関係制約の評価は、制約に含まれる 2 つの式に対して制約演算子を用いて行います。制約演算子は何らかの方法で 2 つの式を比較し、その比較が成立するかどうかを検証します。ある制約において比較が成立することを"制約を満足する"と呼び、比較が成立しないことを"制約に違反する"と呼びます。また、いくつかの関係制約は事前条件を持つことがあります。事前条件を満たさない制約は、制約に含まれる式の評価を中断して制約全体を失敗させます。このように事前条件が満足されなかったことで制約が違反となることを、制約の評価に失敗するといいます。

要素型制約の評価は、制約に含まれる式を評価した結果の型について行います。式を評価した結果が指定の DOM 要素型のいずれかであった場合のみ対象の要素型制約を満足し、そうでない場合には要素型制約に違反します。

制約はクエリ中に複数書くことができ、すべての制約が満足して初めてクエリは検索に成功します。

## 第2節 関係制約

関係制約は、第2章第1節第11項で定義された制約演算子のいずれかを利用することができます。それぞれの演算子は異なる意味を持ち、その制約の評価方法を規定します。各演算子による制約の評価方法についてはそれぞれの項を参照してください。

制約演算子は次の5種類に分類することができます。

1. 同一制約(第1項)
2. 包括制約(第2項)
3. 同値比較制約(第3項)
4. 順序比較制約(第4項)
5. 照合制約(第5項)

### 第1項 同一制約

同一制約は同一演算子(=)を利用して評価を行います。これは評価対象にとる両式が同一である場合のみ制約を満足させます。式の同一性は定義 15に定義されます。

式 A, 式 B に対し、同一性について次のことがいえる。

ただし、eval(X)は任意の式 X を評価した結果を表す。

1. eval(A), eval(B)がともに DOM 要素であり、eval(A)と eval(B)は同一である場合、AとBもまた同一である
2. そうでなく、eval(A), eval(B)がともにリストであり、eval(A)の要素数を n としたとき、eval(B)の要素数が n であり、かつ全ての  $i=1..n$  に対し eval(A)の i 番目の要素と eval(B)の i 番目の要素が同一である場合、AとBもまた同一である
3. それ以外の場合、AとBは同一でない

#### 定義 15. 式の同一性

同一制約の対象とする式のうち一方が即値(第5章第4節)であった場合、その式の評価に先立って捕捉変換が適用されます。捕捉変換は即値が同一性比較を行う対象を捕捉し、即値を捕捉対象と同一のDOM要素に置き換えるような変換です。個々の規則については第5章第4節の各項を参照してください。

### 第2項 包括制約

包括制約は包括演算子(in)を利用して評価を行います。これは、左式の評価結果が右式の評価結果に包括されていた場合のみ制約を満足させます。式の包括は定義 16に定義されます。

式 A, 式 B に対し、包括性について次のことがいえる。

ただし、 $\text{eval}(X)$  は任意の式  $X$  を評価した結果を表す。

1.  $\text{eval}(A)$  が DOM 要素でない場合、制約の評価に失敗する
2. そうでなく、 $\text{eval}(B)$  がともに DOM 要素であり、 $b$  は  $\text{eval}(B)$  を構造展開した結果のリストとしたとき、 $b$  に含まれる要素のうち  $\text{eval}(A)$  と同一である要素が存在する場合、 $B$  は  $A$  を包括する
3. そうでなく、 $\text{eval}(B)$  がリストであるとき、 $\text{eval}(B)$  に含まれる要素のうち  $\text{eval}(A)$  と同一である要素が存在する場合、 $B$  は  $A$  を包括する
4. それ以外の場合、 $B$  は  $A$  を包括しない

#### 定義 16. 式の包括

定義 16 に出現する構造展開は定義 17 に定義されるアルゴリズムで、名前空間、文、式に対して再帰的に子要素の展開を行います。

構造展開の対象とする要素を  $a$  とおき、初期の結果を空のリストとする。

1.  $a$  がパッケージ (CtPackage)を表す要素である場合
  1.  $a$  の直接の子要素のうち、パッケージ(CtPackage)を表す要素を結果に追加する
  2.  $a$  の直接の子要素のうち、宣言型(CtDeclaredType)を表す要素を結果に追加する
  3. 追加した子要素についてそれぞれ再帰的にこのアルゴリズムを適用し、適用結果を全体の結果に追加する
2.  $a$  が文(CtStatement)を表す要素である場合
  1.  $a$  の直接の子要素のうち、式(CtExpression)を表す要素を結果に追加する
  2.  $a$  の直接の子要素のうち、文(CtStatement)を表す要素を結果に追加する
  3. 追加した子要素についてそれぞれ再帰的にこのアルゴリズムを適用し、適用結果を全体の結果に追加する
3.  $a$  が式(CtExpression)を表す要素である場合
  1.  $a$  の直接の子要素のうち、式(CtExpression)を表す要素を結果に追加する
  2. 追加した子要素についてそれぞれ再帰的にこのアルゴリズムを適用し、適用結果を全体の結果に追加する
4.  $a$  がそれ以外の要素である場合
  1. 結果は空のリストである

### 定義 17. 構造展開のアルゴリズム

包括制約の評価を行う際、比較対象の集合に含まれる各要素に対して同一制約による評価が発生します。これは、即値(第 5 章第 4 節)に対する捕捉変換の適用が行われる場合があります。

## 第3項 同値比較制約

同値比較制約は同値比較演算子(==, !=)のいずれかを利用して評価を行います。式の同値性は定義 18に定義され、DOM要素の同値性についてはIrenka DOM Specification内で定義されています。

式 A, 式 B に対し、同値性について次のことがいえる。

ただし、eval(X)は式 X を評価した結果を表す。

1. eval(A), eval(B)がともに DOM 要素であり、eval(A)と eval(B)は同値である場合、A と B もまた同値である
2. eval(A), eval(B)がともにリストであり、eval(A)の要素数を n としたとき、eval(B)の要素数が n であり、かつ全ての  $i=1..n$  に対し eval(A)の i 番目の要素と eval(B)の i 番目の要素が同値である場合、A と B もまた同値である
3. それ以外の場合、A と B は同値でない

### 定義 18. 式同値性

各演算子の評価方法は表 1の通りです。

表 1. 同値比較演算子の振る舞い

演算の内容	制約を満足させる条件
$A == B$	A と B が同値である
$A != B$	A と B が同値でない

## 第4項 順序比較制約

順序比較制約は順序比較演算子( $<$ ,  $>$ ,  $<=$ ,  $>=$ )のいずれかを利用して評価を行います。式の順序関係は定義 19に定義されます。

式 A, 式 B に対し、順序性について次のことがいえる。

ただし、eval(X)は式 X を評価した結果を表す。

1. eval(A), eval(B)がともに整数リテラル、浮動小数点数リテラルのいずれかであり、それらを数値として解釈した値をそれぞれ a, b とした場合
  1. a と b が同値である場合、 $A = B$  もまた成り立つ
  2. そうでなく、a が b より大きい場合、 $A > B$  もまた成り立つ
  3. そうでなく、a が b 未満である場合、 $A < B$  もまた成り立つ
2. そうでなく、eval(A), eval(B)がともに文字リテラルであり、それらの Unicode を数値として解釈した値をそれぞれ a, b とした場合
  1. a と b が同値である場合、 $A = B$  もまた成り立つ
  2. そうでなく、a が b より大きい場合、 $A > B$  もまた成り立つ
  3. そうでなく、a が b 未満である場合、 $A < B$  もまた成り立つ
3. そうでなく、eval(A), eval(B)がともに型であり、それらを型として評価した値をそれぞれ a, b とした場合
  1. a と b が同値である場合、 $A = B$  が成り立つ
  2. そうでなく、a が b のスーパータイプである場合、 $A > B$  が成り立つ
  3. そうでなく、a が b のサブタイプである場合、 $A < B$  が成り立つ
4. それ以外の場合、制約の評価に失敗する

### 定義 19. 式の順序関係

各演算子の評価方法は表 2の通りです。

表 2. 順序比較演算子の振る舞い

演算の内容	制約を満足させる条件
$A > B$	$A > B$ が成立する
$A < B$	$A < B$ が成立する
$A \leq B$	$A < B$ が成立する、または $A=B$ が成立する
$A \geq B$	$A > B$ が成立する、または $A=B$ が成立する

## 第5項 照合制約

照合制約は照合演算子(=~, !~)のいずれかを利用して評価を行います。これは正規表現による文字列のパターンマッチングを行う制約で、照合については定義 20に定義されます。

式 A, 式 B に対し、照合性について次が成り立つ場合のみ B は A を照合する。

ただし、eval(X)は式 X を評価した結果を表す。

1. eval(A), eval(B)がともに文字列リテラルであり、それらを文字列として解釈した値をそれぞれ a, bとした場合
  1. b が Java の正規表現を表す文字列として適切である場合
    1. a は正規表現 b に含まれる場合、B は A を照合する
    2. そうでない場合、B は A を照合しない
  2. そうでない場合
    1. 制約の評価に失敗する
2. それ以外の場合
  1. 制約の評価に失敗する

### 定義 20. 式の照合

各演算子の評価方法は表 3の通りです。

表 3. 照合演算子の振る舞い

演算の内容	制約を満足させる条件
A =~ B	B は A を照合する
A !~ B	B は A を照合しない

## 第3節 要素型制約

要素型制約は式の評価結果(第 5 章第 1 節)として得られるDOM要素の型を限定するための制約で、区切り子(第 2 章第 1 節第 10 項) ":" (¥u003a)の左側に式(第 5 章)、左側に式を評価した際の要素の型をリンク参照(第 6 章第 5 節)として指定することができます。

式の評価結果は必ずDOM要素、DOM要素の一次元リスト、不定のいずれかとなります。このため、リンク参照が指す要素もまたDOM要素の型を表すCtElement型またはそのサブタイプでなけ

ればなりません<sup>1</sup>。CtElement型以外を指すリンク参照を指定した場合、処理系は何らかの例外(第7章第3節)を発生させる場合があります。

また、要素の型は区切り子(第2章第1節第10項)";" (¥u003a)区切りで複数指定することができます。この場合、式の評価結果は複数指定した要素型のうち、いずれかの型またはそのサブタイプである必要があります。そうでない場合、対象の要素型制約は失敗します。

---

<sup>1</sup> この制限により、リンク参照は必ず宣言型(第6章第5節第1項)を指します

## 第5章 式

```

<EXPRESSION>
  ::= <TERM>
     | <LITERAL>
     | <MODIFIER>
     | <BASIC_TYPE>
     | <PRIMITIVE_TYPE_ARRAY>

```

### 定義 21. 式の構造

式(EXPRESSION)は制約の中に出現する評価可能な構文要素です(定義 21)。式はさらに分類され、以下のいずれかとなります。個々の表記の方法や評価の方法については各節の解説を参照してください。

- 項(TERM, 第 2 節)
- 定数リテラル(LITERAL, 第 6 章第 1 節)
- 修飾子(MODIFIER, 第 6 章第 2 節)
- 基本型(BASIC\_TYPE, 第 6 章第 3 節)
- プリミティブ型の配列型(PRIMITIVE\_TYPE\_ARRAY, 第 6 章第 4 節)

### 第1節 式の評価

全ての式は評価することができ、その評価結果は必ず DOM 要素、DOM 要素の一次元リスト、不定のいずれかとなります。評価結果はその式を包括する制約または他の式に引き渡され、そこで利用されます。

評価結果が不定となった場合、その式を直接または間接的に包括する制約を違反させます。

### 第2節 項

```

<TERM>
  ::= <PLACEHOLDER>
     | <LINK>
     | <LIST_CONSTRUCTION>
     | <PROPERTY>
     | <LIST_ACCESS>

```

### 定義 22. 項の構造

式は通常の式と項(TERM)に分類されます(定義 22)。項はそのまま式として使用することができるほか、プロパティ参照(第 5 節)およびリスト参照(第 7 節)の参照先として利用することができます。

下記は項として取り扱うことのできる式の一覧です。これ以外の式は項として取り扱うことができないため、プロパティ参照やリスト参照の参照先とすることはできません。

- プレースホルダ(PLACEHOLDER, 第 3 節)
- リンク参照(LINK, 第 6 章第 5 節)
- プロパティ参照(PROPERTY, 第 5 節)
- リスト構築(LIST\_CONSTRUCTION, 第 6 節)
- リスト参照(LIST\_ACCESS, 第 7 節)

## 第3節 プレースホルダ

```
<PLACEHOLDER>
 ::= <NAME>
```

### 定義 23. プレースホルダの構造

プレースホルダはSearch Query内に出現する要素で、検索対象(第 7 章第 1 節)に含まれる任意のDOM要素、またはそれらから構成される一次元のリストを格納することができます。プレースホルダはそれぞれ名前を有し、それらはJava言語の変数名(第 2 章第 1 節第 3 項)と同様の方法で表記が可能です。ただし、プレースホルダの名前は、Irenka Search Queryが規定する予約語(定義 4)であってはなりません(定義 23)。

プレースホルダは Irenka Search Query の中で非常に重要な役割を果たします。プレースホルダに格納された値がいずれかの制約を違反させる場合、その値は検索結果に出現しません。つまり、検索結果に出現する値は、全ての制約を満足する値の組み合わせとなります。それぞれの値はプレースホルダに格納され、対応するプレースホルダの名前を利用して検索結果を利用することができます。

言い換えると、すべての制約はプレースホルダに格納される値を限定させます。制約がプレースホルダに対して何らかの制限を行うことを、プレースホルダに制約を課すと呼びます。Irenka Search Query ではそれぞれのプレースホルダに制約を課し、それらの制約を全て満足する全ての組み合わせを検索の結果とします。

## 第1項 プレースホルダの評価

プレースホルダの評価結果はかならずプレースホルダに格納された値そのものになります。ただし、プレースホルダに値が格納されていない場合、その結果は不定となります。

検索対象が存在しない場合などでは、プレースホルダに値が格納されません。この場合はプレースホルダの評価結果が不定となり、結果としてそのプレースホルダを利用する制約を違反させます。つまり、検索対象が存在しない場合には検索結果も存在しないということになります。

## 第2項 同名のプレースホルダ

同一の Search Query 中に出現する同名のプレースホルダは、同一のプレースホルダとして取り扱われます。同一のプレースホルダにはかならず同一である値が格納されており、これを利用して1つの値に対して複数の制約を課することができます。

図 1は、1つのプレースホルダが格納する値に対して、6種類の制約を課している例です。出現するそれぞれの制約はいずれもプレースホルダ"method"を利用しており、これらは全て同一の値を保持しています。これは、methodに格納されたある1つの要素が、すべての制約を同時に満足する場合のみクエリ全体が成功します。その際、プレースホルダmethodに格納されている値はいずれかの型で宣言されたmainメソッド<sup>2</sup>を表現しています。

```
{@link CtMethod} = method
public in method.modifiers
static in method.modifiers
void = method.returnType
"main" = method.simpleName
({@link String[]}) = method.parameters.type
```

図 1. プレースホルダに対して複数の制約を課す例

## 第4節 即値

Irenka Search Query は次のような即値を記述することができます。

- 定数リテラル(第 6 章第 1 節)
- 修飾子(第 6 章第 2 節)
- 基本型(第 6 章第 3 節)
- プリミティブ型の配列型(第 6 章第 4 節)
- リンク参照(第 6 章第 5 節)

リンク参照を除くそれぞれの即値は、Java言語に規定される表記と同等の意味を持つDOM要素として評価されます。リンク参照の場合は、その参照先に対応するDOM要素として評価されます。

<sup>2</sup> 修飾子がpublicかつstatic、戻り値型がvoid型、単純名が"main"、引数がString[]型のみからなるメソッド。

## 第1項 即値の評価

即値の評価結果として得られる DOM 要素は特殊な性質を持っていて、同一制約が課せられた際に捕捉変換という処理が適用されます。Search Query 上に記載された即値は一部を除き、検索対象の DOM 要素と記載された場所が異なるため同一ではありません。捕捉変換では、即値を表す式と比較対象の同一制約が評価される際に、適切であると判断された場合に即値を比較対象と同一である要素として評価します。

捕捉変換の適用は定義 24に従って行われ、対象要素を捕捉可能であるかどうかは即値の種類ごとに規定されています。対象要素を捕捉可能である場合には即値に対して捕捉変換が適用され、即値の評価結果は対象の評価結果と同一なものになります。

eval(X)を任意の式 X の評価結果として得られる DOM 要素、

resolve(I)を任意の即値 I の DOM 要素としての表現とする

1. X と I の同一性評価が行われ、式 X と即値 I について、I が eval(X)を捕捉可能である場合
  - I の評価結果を eval(X)によって得られる DOM 要素とする
2. X と I の同一性評価が行われ、式 X と即値 I について、I が eval(X)を捕捉可能でない場合
  - I の評価結果を resolve(I)によって得られる DOM 要素とする
3. X と I の同一性評価が行われず、即値 I が評価される場合
  - I の評価結果を resolve(I)によって得られる DOM 要素とする

### 定義 24. 捕捉変換の適用規則

## 第5節 プロパティ参照

<PROPERTY>

::= <TERM> "." <NAME>

### 定義 25. プロパティ参照式の構造

プロパティ参照(PROPERTY)は項(TERM, 第 2 節)とプロパティの名前(NAME, 第 2 章第 1 節第 3 項)の間に区切り子"."(¥u002e)を挟んだ形式で(定義 25)、項を持つプロパティを参照します。プロパティの構造に含まれる項をプロパティの参照先と呼び、名前をプロパティの名前と呼びます。また、参照先の型から指定の名前を持つプロパティの実体を特定することを、プロパティを解決すると呼びます。

プロパティ参照は、参照先の要素に定義された指定の名前を持つプロパティの値を参照するための文法です。プロパティの参照には任意の項を指定することができ、プロパティ参照もまた項とし

て取り扱うことができるため、プロパティの参照結果に対するプロパティ参照などを行うことができます。

プロパティ参照の評価結果は、参照先を評価した際の値やプロパティの名前によって決まります。参照先がDOM要素である場合にも、またはその一次元のリストである場合にもプロパティ参照を利用することができます。プロパティ参照の評価方法について、詳しくは第2項を参照してください。

## 第1項 プロパティ

Irenka が提供する DOM は Java のプログラムを表すツリー構造をしていて、それぞれの子要素は親要素のプロパティとして公開されています。Irenka Search Query におけるプロパティのほとんどは、それぞれのDOMが公開するプロパティとほぼ対応しており、参照先に対して適切なプロパティ名を与えることにより、その結果の子要素を利用できるようになります。

ほとんどのプロパティは、それぞれのDOM要素が公開するgetterメソッドと対応しており、それぞれの名前から先頭の"get"または"is"を除去して最初の文字を小文字に変換した結果の文字列("getSuperClass" -> "superClass")がIrenka Search Queryにおけるプロパティの名前となります。また、Irenka Search Queryではいくつか特殊なプロパティを公開している場合があります。プロパティの一覧と、それらの振る舞いについては付録Bを参照してください。また、DOM要素が公開するメソッドについてはIrenka End User API Referenceを参照してください。

また、DOM要素が公開するgetterメソッドのいくつかは、結果としてDOM要素またはその一次元のリストのどちらでもない値を返す場合があります。これは式の評価結果がとりうる値(第1節)として適切でないため、表4の規則に従ってDOM要素として加工された値が返されます。

表 4. getter メソッドの結果が加工される方法

getter の型	対応するプロパティを評価した値が持つ型
int	org.ashikunep.irenka.CtLiteral<Integer>
boolean	org.ashikunep.irenka.CtLiteral<Boolean>
String	org.ashikunep.irenka.CtLiteral<String>
T 型の列挙	org.ashikunep.irenka.CtEnumConstant<? extends T>

## 第2項 プロパティ参照の評価

プロパティ参照は、参照先を評価した結果がDOM要素となるかその一次元リストとなるかによって大きく評価方法が異なります。DOM要素はそれぞれいくつかのプロパティを有しており、プロパティはDOM要素に定義されたそれをそのまま参照しますが、クエリの処理中に出現する一次元リストはプロパティを直接有しません。そこで、一次元リストではそこに含まれる要素のプロパティをそれぞれ参照し、その結果をリスト化したもの<sup>3</sup>をプロパティ参照の評価結果とします。

<sup>3</sup> リストに対するmap演算のようなものです。個々の要素にプロパティ演算を適用し、その結果から新しいリストを作ります。

まず、プロパティの参照先である項を評価した結果がDOM要素となる場合、定義 26の手順でプロパティ参照が評価されます。

参照先の項を評価した結果の DOM 要素を T、プロパティの名前を P とおき、  
refer(X, P)を任意の DOM 要素 X が持つ P という名前のプロパティが有する値とする

1. T が P という名前のプロパティを有する場合
  - プロパティ参照の結果は、refer(T, P)となる
2. そうでない場合
  - プロパティ参照の結果は、不定となる

#### 定義 26. DOM 要素を参照先とするプロパティの評価方法

また、プロパティの参照先である項を評価した結果がDOM要素の一次元リストとなる場合、定義 27の手順でプロパティが評価されます。これはつまり、"(hoge, foo, bar).prop"という式を "(hoge.prop, foo.prop, bar.prop)"と評価することとほとんど同様の効果<sup>4</sup>が得られます。

参照先の項を評価した結果のリストを L、プロパティの名前を P とおき、  
refer(X, P)を任意の DOM 要素 X が持つ P という名前のプロパティが有する値とする

1. Lに含まれるDOM要素がそれぞれPという名前のプロパティを有し、いずれのプロパティも同様の種類であった場合
  1. すべてのLに含まれる要素eについて、refer(e, P)の結果がDOM要素となる場合
    - プロパティ参照の結果は、Lの各要素eに対してrefer(e, P)を適用した結果のリストである
  2. そうでなく、refer(e, P)の結果がリストとなるような要素eがLに含まれる場合
    - プロパティの結果は、不定となる
2. そうでない場合
  1. プロパティ参照の結果は、不定となる

#### 定義 27. リストを参照先とするプロパティの評価方法

<sup>4</sup> hoge, foo, barのいずれかに"prop"という名前の別のプロパティが存在する場合、この置き換えは同等ではありません。

## 第6節 リスト構築

```
<LIST_CONSTRUCTION>
 ::= "(" ")"
    | "(" <EXPRESSION> ( "," <EXPRESSION> )* ")"
```

### 定義 28. リスト構築式の構造

リスト構築はDOM要素の一次元リストを作成するための文法です。区切り子"(" (§u0028)、")" (§u0029)の間に"," (§u002c)区切りで要素とする式(EXPRESSION)を表記し(定義 28)、それぞれの式の評価結果を順番に要素に持つ新しいリストを構築することができます。このとき、それぞれの式を要素式と呼びます。

リスト構築は項として取り扱うことができ、プロパティの参照先やリスト参照の参照先とすることができます。特に、リストに対するプロパティ参照の評価方法(第5節第2項)は特殊なので、注意が必要です。

### 第1項 リスト構築の評価

リスト構築の評価結果はリストとなり、評価結果のリストを構成する要素は要素式を左から順に評価した結果となります。ただし、Irenka Search Queryでは二次元以上のリストを取り扱うことができないため、いずれかの要素式の評価結果がリストとなった場合、リスト構築の評価結果は不定となります。

リスト構築は、式の評価をそれ自身では行うことはなく、代わりにリストに含まれる要素を利用する他の制約や式が各式の評価を行います。特に、包括制約(第4章第2節第2項)が右辺にリスト構築式をとる場合、それぞれの要素式に含まれる即値は捕捉変換(第4節第1項)が適用されます。これを利用して、"var in ({@link CtLocalVariable}, {@link CtParameter})" - プレースホルダ"var"はCtLocalVariable型またはCtParameter型 - といった記述が可能です。

## 第7節 リスト参照

```
<LIST_ACCESS>
 ::= <TERM> "[" <EXPRESSION> "]"
```

### 定義 29. リスト参照式の構造

リスト参照(LIST\_ACCESS)はDOM要素の一次元リストに含まれるDOM要素を参照するための文法です。項(TERM, 第2節)の後ろに区切り子"[" (§u005b)、"]" (§u005d)に囲まれた式(EXPRESSION)を表記し(定義 29)、*list[index]*の形式で*list*の*index*番目に格納された要素を参照できます。このとき、項をリスト参照の参照先と呼び、[ ]に囲まれた式をリストの添え字と呼びます。

リスト参照は項として取り扱うことができ、プロパティの参照先やリスト参照の参照先とすることができます。ただし、Irenka Search Query では二次元以上のリストを利用することができないため、リスト参照の結果に対してリスト参照を行うことはできません。

## 第1項 リスト参照の評価

リスト参照の評価結果は、参照先を一次元リストとみなし、添え字をint型の定数リテラル(第6章第1節)とみなして、リストの添え字が指し示す位置の要素となります。ただし、リストの位置は最初の要素を0番目の要素とします。より厳密には、定義30に定義される方法でリスト参照の評価を行います。

参照先を表す式を  $R$ 、添え字を表す式を  $I$  とおき、

$\text{eval}(X)$  を任意の式  $X$  を評価した際の結果とする

1.  $\text{eval}(R)$  がリストでなかった場合、または  $\text{eval}(I)$  が int 型の定数リテラルでなかった場合
  1. リスト参照の結果は、不定となる
2. そうでなく、 $\text{eval}(R)$  が長さ  $n$  のリスト、 $\text{eval}(I)$  が整数  $i$  を表現する整数リテラルであった場合
  1.  $i$  が 0 未満、または  $i$  が  $n$  以上であった場合
    - リスト参照の結果は、不定となる
  2. そうでなく、 $i$  が 0 以上かつ  $n$  未満であった場合
    - リスト参照の結果は、 $\text{eval}(R)$  の  $n$  番目の要素となる

### 定義 30. リスト参照の評価

## 第6章 即値

即値は式(第5章)の一種で、Java言語仕様にある定数リテラル(第1節)、修飾子(第2節)、基本型(第3節)、プリミティブ型の配列型(第4節)、およびJavaドキュメンテーションコメントの要素であるリンク参照(第5節)を利用することができます。それぞれの即値は、クエリ上の表記に対応するDOM要素として評価されます。即値の評価方法については第5章第4節第1項を参照してください。

### 第1節 定数リテラル

```
<LITERAL>
  ::= <INTEGER_LITERAL>
     | <REAL_LITERAL>
     | <BOOLEAN_LITERAL>
     | <CHARACTER_LITERAL>
     | <STRING_LITERAL>
     | <NULL_LITERAL>
```

#### 定義 31. 定数リテラルの種類

定数リテラル(LITERAL)はJava言語で規定される定数リテラルと同等のものが用意されており、Javaプログラム上に出現する定数リテラルに対応するDOM要素として評価されます(定義 31)。以下は利用可能な定数リテラルの一覧です。

- 整数リテラル(INTEGER\_LITERAL, 第2章第1節第4項)
- 浮動小数点数リテラル-REAL\_LITERAL, 第2章第1節第5項)
- 真偽リテラル-BOOLEAN\_LITERAL, 第2章第1節第6項)
- 文字リテラル-CHARACTER\_LITERAL, 第2章第1節第7項)
- 文字列リテラル-STRING\_LITERAL, 第2章第1節第8項)
- nullリテラル(NULL\_LITERAL, 第2章第1節第9項)

定数リテラルを評価すると org.ashikunep.irenka.dom.CtLiteral<T>型の DOM 要素となり、型引数の T はリテラルの実行時の型(またはそのラップ型)となります。ただし、null リテラルのみは対応する型が存在しないため CtLiteral<?>として表現されます。また、それぞれの DOM 要素が持つプロパティ *literalString* は、Search Query 上に表記した文字列と同一の文字列を保持します。なお、クラスリテラルは利用できません。

定数リテラルが捕捉可能な DOM 要素は、その要素と同値である(同一の値に評価されるリテラル値を表す) DOM 要素です。

## 第2節 修飾子

```
<MODIFIER>
 ::= "public"
    | "protected"
    | "private"
    | "final"
    | "static"
    | "abstract"
    | "native"
    | "synchronized"
    | "volatile"
    | "transient"
    | "strictfp"
```

### 定義 32. 修飾子の種類

修飾子(MODIFIER)はJava言語で規定される修飾子と同様のものが用意されており、それぞれJavaプログラム上に出現する修飾子に対応するDOM要素として評価されます (定義 32)。修飾子进行评估すると、`org.ashikunep.irenka.dom.CtModifier`型のDOM要素となり、また、そのDOM要素が持つプロパティ`kind`は、列挙`org.ashikunep.irenka.dom.ModifierKind`内で定義された定数のうち、即値の表記に対応するものを保持します。

修飾子が捕捉可能な DOM 要素は、その要素と同値である(同一の表記を持つ修飾子を表す)DOM 要素です。

## 第3節 基本型

```

<BASIC_TYPE>
  ::= "void"
    | <PRIMITIVE_TYPE>

<PRIMITIVE_TYPE>
  ::= "int"
    | "long"
    | "float"
    | "double"
    | "byte"
    | "short"
    | "char"
    | "boolean"

```

### 定義 33. 基本型の種類

基本型(BASIC\_TYPE)はJava言語で規定されるvoid型またはプリミティブ型(PRIMITIVE\_TYPE)を表し(定義 33)、それぞれJavaプログラム上に出現する基本型に対応するDOM要素として評価されます。修飾子を評価すると、`org.ashikunep.irenka.dom.CtType<T>`型のDOM要素となり、型引数のTは対応する基本型のラッパ型となります。また、そのDOM要素が持つプロパティ `typeKind`は、列挙 `org.ashikunep.irenka.dom.TypeKind`内で定義された定数のうち、対応する基本型を表現するもの保持します。

基本型が捕捉可能な DOM 要素は、その要素と同値である(同一の表記を持つ基本型を表す)DOM 要素です。

## 第4節 プリミティブ型の配列型

```

<PRIMITIVE_TYPE_ARRAY>
  ::= <PRIMITIVE_TYPE> ( "[" "]" )+

```

### 定義 34. プリミティブ型の配列型の構造

プリミティブ型の配列型は、はJava言語で規定されるプリミティブ型の配列型(JLS3-10)を表し(定義 34)、Javaプログラム上に出現する配列型に対応するDOM要素として評価されます。プリミティブ型の配列型を評価すると、`org.ashikunep.irenka.dom.CtArray<C>`型のDOM要素とな

り、型引数のCは対応する要素型となります。ただし、Cの0次元の要素型はプリミティブ型を取らず、必ずそのラップ型として表現されます<sup>5</sup>。また、それぞれのDOM要素が持つプロパティ *componentType*は、対応する型を第3節または第4節で規定される即値とみなし、それを評価した結果を保持します。

プリミティブ型の配列型が捕捉可能な要素は、その要素と同値である(同一の表記である配列型を表す)DOM要素です。

## 第5節 リンク参照

```

<LINK>
    ::= "{@link" <LINK_TARGET> "}"

<LINK_TARGET>
    ::= <DECLARED_TYPE>
       | <DECLARED_TYPE_ARRAY>
       | <FIELD>
       | <METHOD>

```

### 定義 35. リンクの構造

リンク参照(LINK)はJavaドキュメンテーションコメントの@linkインラインタグを利用して表記され(定義 35)、タグ内で指定された宣言や参照に対応するDOM要素として評価されます。Irenka Search Queryでは@linkインラインタグによって指定可能な要素の種類を拡張し、宣言型、フィールド、メソッドの参照に加え、宣言型の配列型、パラメータ化型、パラメータ化フィールド、パラメータ化メソッドも指定することが可能です。

リンク参照の評価方法は、その参照先によって異なります。個々の評価方法については、それぞれ下記の項を参照してください。

- 宣言型(DECLARED\_TYPE, 第1項)
- 宣言型の配列型(DECLARED\_TYPE\_ARRAY, 第2項)
- フィールド(FIELD, 第3項)
- メソッド(METHOD, 第4項)

<sup>5</sup> つまり、CtArray<int[]>という表現はなく、CtArray<Integer[]>となります。これは、CtType<Integer>というint型を表現する型の配列を生成した場合にCtArray<Integer>となり、さらにその配列型がCtArray<Integer[]>となるためです。CtArray<int[]>とCtArray<Integer[]>では型に互換性はありません。

上記の項目はいずれも、@linkインラインタグの中に表記することによって式として取り扱うことができます<sup>6</sup>。

また、実行時にこれらの参照先のDOMを生成できなかった場合、処理系は適切な方法でリンク参照解決に関する例外(第7章第3節第3項)を発生させます。この場合、実際の検索処理は行われません。

## 第1項 宣言型

```
<DECLARED_TYPE>
 ::= <QUALIFIED_NAME> <TYPE_ARG_LIST>?
```

### 定義 36. 宣言型の構造

宣言型(DECLARED\_TYPE)はJava言語仕様のクラス型、インターフェース型、列挙型、注釈型、パラメータ化型のいずれかを表現するもので、Irenka Search Queryではこれらをまとめて単純に宣言型と呼びます。宣言型は単純名または完全限定名のいずれかで名前(QUALIFIED\_NAME)を指定し、それに型引数(TYPE\_ARG\_LIST)を続けることができます(定義36)。QUALIFIED\_NAME, TYPE\_ARG\_LISTの構造については付録Aに詳しい説明があります。このうち、型引数の指定は省略してもよく、指定がないものを単純宣言型、指定があるものをパラメータ化型と呼びます。

単純宣言型は、その名前が指す型のマスタ参照(宣言)を表すDOM要素として評価されます。パラメータ化型は、その名前が指す型に指定された型引数を与えたスレーブ参照(参照)を表すDOM要素として評価されます。いずれも、宣言の種類によって下記のいずれかのDOM要素として表現されることとなります。それぞれの型引数 T は対応する実行時の型となります。

- org.ashikunep.irenka.dom.CtClass<T> (クラス型)
- org.ashikunep.irenka.dom.CtInterface<T> (インターフェース型)
- org.ashikunep.irenka.dom.CtEnum<T> (列挙型)
- org.ashikunep.irenka.dom.CtAnnotation<T> (注釈型)

宣言型の名前に単純名が利用された場合、その参照先の解決方法は Search Query を解釈する環境に影響されます。Search Query の解釈器は単純型を適切な方法で解釈する必要があります。

宣言型が捕捉可能な要素は、定義 37 に定義されます。即値が単純宣言型を表現する場合、捕捉対象の宣言との同一性を考慮するため、対象がパラメータ化型であっても型パラメータをすべて無視して捕捉が行われます。なお、未加工型のみを捕捉する即値を記述することはできません。

<sup>6</sup> 宣言型の表記とプロパティ参照(第5章第5節)の表記とがあいまいであるため、いずれも{@link ~}のように表記する必要があります。宣言型の配列型は少し違和感があります({@link String[]})など。

即値に対応する宣言型を表現する DOM 要素を S、比較対象を評価した結果を T とする。

1. S が単純宣言型を表現する DOM 要素である場合
  - S と T のマスタ参照が同値である(同一の型宣言を表現する)場合のみ即値は対象を捕捉可能である
2. S がパラメータ化型を表現する DOM 要素である場合
  - S と T が同一の型を表現する場合のみ、即値は対象を捕捉可能である
3. それ以外の場合
  - 即値は対象を捕捉可能でない

### 定義 37. 宣言型が捕捉可能な式

## 第2項 宣言型の配列型

```
<DECLARED_TYPE_ARRAY>
 ::= <DECLARED_TYPE> ( "[" "]" )+
```

### 定義 38. 宣言型の配列型の構造

宣言型の配列型(DECLARED\_TYPE\_ARRAY)は、Java言語で規定される宣言型の配列型(JLS3-10)を表し(定義 38)、Javaプログラム上に出現する配列型に対応するDOM要素として評価されます。宣言型の配列型を評価すると、org.ashikunep.irenka.dom.CtArray<C>型のDOM要素となり、型引数のCは対応する要素型となります。また、そのDOM要素が持つプロパティ *componentType* は、対応する型を宣言型(第1項)または宣言型の配列型(第2項)で規定される即値とみなし、それを評価した結果を保持します。

宣言型の配列型が捕捉可能な要素は、その要素と同値である(同一の型を要素型に持つ配列型を表す)DOM要素です。

## 第3項 フィールド

```
<FIELD>
 ::= <DECLARED_TYPE>? "#" <NAME>
```

### 定義 39. フィールドの構造

フィールド(FIELD)はJava言語仕様のフィールド、列挙定数のいずれかを表現するもので、Irenka Search Queryではこれらをまとめて単純にフィールドと呼びます。フィールドはそれを宣言する宣言型(DECLARED\_TYPE)とフィールドの単純名(NAME)の間に区切り子"#"(¥u0023)を挟んだ形式で(定義 39)、指定の宣言型の中に指定の単純名で定義されたフィールドを参照します。宣言型の指定が省略された場合、その宣言型の解決方法はSearch Queryを解釈する環境に

影響されます。Search Queryの解釈器はこのような省略された宣言型を適切な方法で解釈する必要があります。

フィールドは、宣言型で指定された型が単純宣言型であるかパラメータ化型のいずれかであるかによってその評価方法が異なります。宣言型が単純宣言型として解決されるフィールドを単純フィールド、パラメータ化型として解決されるフィールドをパラメータ化フィールドと呼びます。

単純フィールドは、参照先のフィールドまたは列挙定数のマスタ参照(宣言)を表すDOM要素として評価されます。パラメータ化フィールドは、対象のフィールドに総称化コンテキストを与えたスレーブ参照(参照)を表すDOM要素として評価されます<sup>7</sup>。いずれも、フィールドの種類によって下記のいずれかのDOM要素として表現されることになります。それぞれの型引数Tは対応するフィールドを評価した際の型となります。

- org.ashikunep.irenka.dom.CtField<T> (フィールド)
- org.ashikunep.irenka.dom.CtEnumConstant<T> (列挙定数)

フィールドが捕捉可能な要素は、定義 40に定義されます。

即値に対応するフィールドを表現する DOM 要素を S、比較対象を評価した結果を T とする。

1. T がフィールドまたは列挙定数を表現する DOM 要素である場合
  1. 即値が単純フィールドを表現する場合
    - SとTのマスタ参照が同値である(同一のフィールド宣言を表現する)場合のみ即値は対象を捕捉可能である
  2. 即値がパラメータ化フィールドを表現する場合
    - SとTが同値である(同一のフィールド宣言と同一の総称化コンテキストを有する)場合のみ即値は対象を捕捉可能である
2. そうでない場合
  1. 即値は対象を捕捉可能でない

#### 定義 40. フィールドが捕捉可能な式

<sup>7</sup> class A<T> { T field; } という宣言に対して{@link A<String>#field}と指定すると、そのフィールド要素の型はStringと解釈することができます。これは、指定のフィールド要素が[T:=String]という総称化コンテキストを与えられている状態で、型変数TはStringに置換されて利用されます。

## 第4項 メソッド

```

<METHOD>
    ::= <DECLARED_TYPE>? "#" <TYPE_ARG_LIST>? <NAME> <PARAM_LIST>

<PARAM_LIST>
    ::= "(" ")"
    | "(" <PARAMETER> ( "," <PARAMETER> )* ")"

<PARAMETER>
    ::= <TYPE> <NAME>?
  
```

### 定義 41. メソッドの構造

メソッド(METHOD)は Java 言語仕様のメソッド、コンストラクタ、注釈要素のいずれかを表現するもので、Irenka Search Query ではこれらをまとめて単純にメソッドと呼びます。

メソッドは切り子"#"(¥u0023)の左側にそれを宣言する宣言型(DECLARED\_TYPE)を表記し、その右側に型引数リスト(TYPE\_ARG\_LIST)、単純名(NAME)、引数リスト(PARAM\_LIST)を表記した形式で(定義 41)、指定の宣言型の中で定義された指定の単純名と引数宣言リストを持つメソッドを参照します。宣言型の指定が省略された場合、その宣言型の解決方法はSearch Queryを解釈する環境に影響されます。Search Queryの解釈器はこのような省略された宣言型を適切な方法で解釈する必要があります。

メソッドは、その表記に型引数リストを含むかどうかによってその評価方法が異なります。宣言型が単純宣言型として解決され、かつ区切り子と単純名の間に型引数リストを取らないようなメソッドを単純メソッドと呼びます。逆にパラメータ化型を宣言型にとるものや、区切り子と単純名の間に型引数リストの指定があるものはパラメータ化メソッドと呼びます。

単純メソッドは、参照先のメソッド、コンストラクタ、注釈要素いずれかのマスタ参照(宣言)を表すDOM要素として評価されます。パラメータ化メソッドは、対象のメソッドまたはコンストラクタ<sup>8</sup>に総称化コンテキストを与えたスレーブ参照(参照)を表すDOM要素として評価されます。いずれも、メソッドの種類によって下記のいずれかのDOM要素として表現されることとなります。それぞれの型引数Tは対応する要素の戻り値型となります。

- org.ashikunep.irenka.dom.CtMethod<T> (メソッド)
- org.ashikunep.irenka.dom.CtConstructor<T> (コンストラクタ)
- org.ashikunep.irenka.dom.CtAnnotationElement<T> (注釈要素)

<sup>8</sup> 注釈要素は、その宣言型である注釈型にも要素の宣言にも型引数を含めることができません。

メソッドが捕捉可能な要素は、定義 42に定義されます。

即値に対応するメソッドを表現する DOM 要素を S、比較対象を評価した結果を Tとする。

1. T がメソッド、コンストラクタ、注釈要素のいずれかを表現する DOM 要素である場合
  1. 即値が単純メソッドを表現する場合
    - SとTのマスタ参照が同値である(同一のメソッド宣言を表現する)場合のみ即値は対象を捕捉可能である
  2. 即値がパラメータ化メソッドを表現する場合
    - SとTが同値である(同一のメソッド宣言と同一の総称化コンテキストを有する)場合のみ即値は対象を捕捉可能である
2. そうでない場合
  1. 即値は対象を捕捉可能でない

**定義 42. メソッドが捕捉可能な式**

## 第7章 検索

---

この章では、ここまで規定した Irenka Search Query の構造を利用して、実際に検索が行われる際の処理について規定します。なお、Irenka Search Query を利用したクエリを検索するための機構を Irenka Search Engine、または文脈によっては単純に処理系と呼びます。

### 第1節 検索対象

検索対象とは Irenka Search Engine が利用する可能性のある全ての値の集合です。これらの値は計算中にプレースホルダに格納され、検索結果に出現する可能性があります。

Irenka Search Engine は Irenka DOM 全体を検索対象とするのではなく、まず、Irenka DOM に含まれるプログラムの一部を初期検索対象(第 1 項)として指定します。そして、そこから到達する全ての値や、Search Query から潜在的に到達可能な全ての値を検索対象として自動的に算出します。

### 第1項 初期検索対象の算出

初期検索対象とはユーザが指定する検索範囲のことで、Irenka DOM 上に存在する任意の DOM 要素を指定することができます。初期検索対象は Irenka Search Engine が実際に取り扱う値である検索対象を計算するための集合で、初期検索対象には即値やプロパティの値など、式の評価によって得られる値を含める必要はありません。

初期検索対象は 2 種類の方法で指定することができます。以下はその方法の概要です。

1. DOM 要素の集合を直接指定する
2. Irenka DOM に含まれる部分木を指定する

1は初期検索対象に含むDOM要素を直接指定し、特に加工等を行わない方法です。2は Irenka DOM がツリー構造を有することを利用し、指定したDOM要素を根とした部分木を初期検索対象に含める方法です。この方法では、定義 43 のアルゴリズムに従って計算が行われ、部分木に含まれるスレーブ参照とその子要素以外のすべての要素が初期検索対象に含まれます。

1. 要素がマスタ参照である場合
  - 自身を検索対象に加え、さらに自身の全ての子要素に対して再帰的にこのアルゴリズムを適用する
2. 要素がスレーブ参照である場合
  - 何も行わない
3. それ以外の要素である場合
  - 自身を検索対象に加え、さらに自身の全ての子要素に対して再帰的にこのアルゴリズムを適用する

#### 定義 43. 初期検索対象に部分木を指定した場合の解釈

## 第2項 検索対象の算出

検索対象はIrenka Search Engineが一度の検索を行っている最中に潜在的に出現するすべての値の集合を指します。これは定義 44の方法を元に、初期検索対象、即値(第 5 章第 4 節)、プロパティ参照(第 5 章第 5 節)、リスト構築(第 5 章第 6 節)などを考慮して算出されます。

1. 初期検索対象集合に含まれるすべての要素
2. Search Queryに出現する即値の評価結果<sup>9</sup>
3. Search Query に出現するリスト構築の評価結果
4. 検索対象にプロパティ参照を適用した評価結果<sup>10</sup>、および結果の一次元リストに含まれるすべてのDOM要素

#### 定義 44. 検索対象に含まれる要素

検索対象が無限集合となる場合、Irenka Search Engineはそれらを適切に取り扱い、有限時間内で計算可能である等価な問題へと変換するか、または検索を行わずに例外(第 3 節)を発生させる必要があります。検索対象は通常、Irenka DOMのモデルに含まれる要素と即値の評価によって得られる要素、およびそれらを要素に含むリストに限定されますが、特殊なプロパティの参照によってそれ以外の要素を生成する場合があります。プロパティ参照は参照透明性を保証する

<sup>9</sup> 即値の評価には捕捉変換が適用される場合がありますが、捕捉変換によって得られるDOM要素は必ず検索対象に含まれる要素と同一です。つまり、捕捉変換が適用されない場合の評価結果のみが検索対象に含まれていない可能性があり、捕捉変換を適用した結果はほかの方法で算出される検索対象の集合に含まれていることが保証されます。

<sup>10</sup> この操作は再帰的に行われます。つまり、プロパティ参照を適用した評価結果に対して、さらにプロパティ参照を適用した評価結果もまた検索対象に含まれます。

必要はありませんが、それらのプロパティを利用する場合には何らかの方法で無限集合の生成を検出しなければなりません。

## 第2節 検索処理

### 第1項 プレースホルダ間の依存

各プレースホルダ間には依存関係というものが存在し、定義 45に示す方法で算出されます。同一制約の直接の対象となるプレースホルダは依存先を持つプレースホルダとして取り扱われ、その依存先のプレースホルダの値が確定するまで依存元のプレースホルダを含む式の計算が行われません。ただし、別名と判断されたプレースホルダ間には依存関係はなく、同一である 1 つのプレースホルダとして利用されます。

あるプレースホルダ  $a$  と、あるプレースホルダ  $b$  は、依存性に関して次の関係を持つ。

ただし、 $p(X)$ は任意の式  $X$  に出現する全てのプレースホルダの集合とする。

1. 制約が  $a = b$  の形式を持つ同一制約で表わされる場合、 $a$  は  $b$  の別名であり、 $a$  と  $b$  は同一のプレースホルダとして取り扱われる
2.  $X$  はプレースホルダのみからなる式ではなく、かつ制約が  $a = X$  または  $X = a$  の形式を持つ同一制約で表わされる場合、 $b \in p(X)$ となる全ての  $b$  について  $a$  は  $b$  に依存する
3. 制約が  $a \text{ in } X$  の形式を持つ包括制約で表わされる場合、 $b \in p(X)$ となる全ての  $b$  について  $a$  は  $b$  に依存する
4.  $a$  が任意のプレースホルダ  $x$  に依存し、かつ  $x$  が  $b$  に依存する場合、 $a$  は  $b$  に依存する

#### 定義 45. プレースホルダ間の依存関係

Irenka Search Queryで記述されたクエリは、循環依存を持つプレースホルダを含めることができません。循環依存関係を持つプレースホルダとは直接または間接的に自分自身に依存するプレースホルダのことで、これは検索対象の算出時に無限集合を生成する可能性があります。このようなクエリに対して、処理系は適切に循環依存(第 3 節第 2 項)に関する例外を発生させなければなりません。

### 第2項 プレースホルダの初期化と伝搬

各プレースホルダの値は、Irenka Search Engine によって次のいずれかで決定されます。

1. 定数による初期化が行われる
2. 初期検索対象による初期化が行われる
3. 依存先のプレースホルダから伝搬される

最初の定数式とは式(第5章)の中でも特殊なもので、式中にプレースホルダが出現しないものを指します。より厳密には、即値(第5章第4節)、プロパティ参照(第5章第5節)、リスト構築(第5章第6節)、リスト参照(第5章第7節)のみで構成される式が定数式に該当します。依存先を持たないプレースホルダが定数式と同一制約の関係にある場合、このプレースホルダは定数による初期化が行われます(定義46)。定数式による初期化は、まず初期検索対象集合に含まれるすべての要素と定数式の同一性が評価され、同一であるものが存在すればその値によってプレースホルダが初期化されます。そうでない場合、定数式を評価した値によってプレースホルダが初期化されます。このようなプレースホルダを定数プレースホルダと呼びます。

次のいずれかの条件を満たすプレースホルダ  $p$  は定数による初期化が行われる対象となる

ただし、 $C$  はプレースホルダが出現しない定数式を表す。

1.  $p$  は依存先を持たず、かつ  $p = C$  または  $C = p$  の形式を持つ同一制約がクエリ中に存在する
2.  $p$  は依存先を持たず、かつ  $p \in C$  の形式を持つ包括制約がクエリ中に存在する

#### 定義 46. 定数による初期化が行われる対象

次の初期検索対象による初期化が行われる対象は、定数による初期化がおこなれておらずかつ依存先を持たないプレースホルダが対象となります。このようなプレースホルダは開放プレースホルダと呼ばれ、初期検索対象による初期化の対象となります。これらのプレースホルダは、その格納できる値が暗黙のうちに初期検索対象集合に含まれる値した値に制限されます。この暗黙の制約を初期化制約と呼びます。

すべてのプレースホルダは定数プレースホルダ、開放プレースホルダ、依存先を持つプレースホルダの3種類に分類されます。すべての定数プレースホルダおよび開放プレースホルダは初期化が行われ、その値が依存関係によって依存元へと同一制約を用いて伝搬されることによって、クエリ上に存在するすべてのプレースホルダの値を確定することができます。その際、プレースホルダに値を伝搬させる場合には、常に常に制約を満足させながら行う必要があります。

### 第3項 検索結果の作成

全ての制約を同時に満足するプレースホルダの値の組み合わせを発見した場合、処理系はその組み合わせに対するスナップショットを生成します。このスナップショットは各プレースホルダの名前と、そこに格納された値を保持しており、ユーザはスナップショットに含まれるプレースホルダのうち必要なものを抜粋して検索結果として利用します。

全ての制約を同時に満足するプレースホルダの値の組み合わせは1つの検索対象に1種類とは限らず、2通り以上の検索結果が得られる場合があります。この場合、処理系は何らかの方法をもって、全ての可能性に対する検索結果を提供する方法を有する必要があります。

## 第3節 検索例外

検索例外は Irenka Search Engine が実際の検索処理を行う前に発生させる例外で、検索が正常に行われないうちにユーザへの通知のために発生させます。これは、下記の 2 種類に分類されます。

1. 検索処理を行うことができない場合
2. 検索処理は行えるが、確実に検索結果が存在しないような自明な矛盾を持つ場合

前者は処理系が必ず実装しなければならない処理で、構文例外(第 1 項)、循環依存例外(第 2 項)、リンク参照解決(第 3 項)の 3 種類が該当します。後者は処理系が選択して実装できる処理で、衝突制約例外(第 4 項)、プロパティ解決(第 5 項)が該当します。後者を実装しなかった場合には単純に検索結果が存在しないのみとなります。

### 第1項 構文例外

Irenka Search Engineが第 2 章で規定される文法外のクエリを処理しようとした際に、処理系は構文例外を発生させなければなりません。

### 第2項 循環依存例外

第 2 節第 1 項に説明のあるプレースホルダ間の依存が存在する場合、処理系は循環依存例外を発生させます。

### 第3項 リンク参照解決例外

第 6 章第 5 節に説明のあるリンク参照で、その参照先を Irenka DOM 上から検出できなかった場合、処理系はリンク参照解決例外を発生させます。

### 第4項 衝突制約例外

衝突する 2 つ以上の制約によってクエリ全体に矛盾が存在する場合、処理系は矛盾制約例外を発生させることができます。下記は、矛盾を持つクエリの例です。

1. 自明な矛盾("1 = 2"など)が存在する
2. プレースホルダの型推論によって値をとらないプレースホルダが存在することが判明した
3. 要素を持たないリスト構築に対して包括制約を評価する場合

### 第5項 プロパティ解決例外

第 5 章第 5 節に説明のあるプロパティ参照で、その参照先が常に対象のプロパティを有しないことが判明した場合、処理系はプロパティ解決例外を発生させることができます。下記は、解決できないプロパティを持つ例です。

1. プロパティの名前が存在しないプロパティを表現している
2. 型推論の結果、プロパティの参照先が指定の名前のプロパティを有しない場合

## 付録A. Irenka Search Query の BNF

```

<QUERY>
  ::= "@when"? <CONSTRAINT>*

<CONSTRAINT>
  ::= <RELATIONAL_CONSTRAINT>
     | <ELEMENT_TYPE_CONSTRAINT>

<RELATIONAL_CONSTRAINT>
  ::= <EXPRESSION> <OPERATOR> <EXPRESSION>

<ELEMENT_TYPE_CONSTRAINT>
  ::= <EXPRESSION> ":" <ELEMENT_TYPES>

<EXPRESSION>
  ::= <TERM>
     | <LITERAL>
     | <MODIFIER>
     | <BASIC_TYPE>
     | <PRIMITIVE_TYPE_ARRAY>

<TERM>
  ::= <PLACEHOLDER>
     | <LINK>
     | <LIST_CONSTRUCTION>
     | <PROPERTY>
     | <LIST_ACCESS>

<PLACEHOLDER>
  ::= <NAME>

<LINK>
  ::= "{@link" <LINK_TARGET> "}"

<LIST_CONSTRUCTION>
  ::= "(" ")"
     | "(" <EXPRESSION> ( "," <EXPRESSION> )* ")"

```

```

<PROPERTY>
    ::= <TERM> "." <NAME>

<LIST_ACCESS>
    ::= <TERM> "[" <EXPRESSION> "]"

<ELEMENT_TYPES>
    ::= <DECLARED_TYPE_LINK> ( "," <DECLARED_TYPE_LINK> )*

<DECLARED_TYPE_LINK>
    ::= "{@link" <DECLARED_TYPE> "}"

<LINK_TARGET>
    ::= <DECLARED_TYPE>
    | <DECLARED_TYPE_ARRAY>
    | <FIELD>
    | <METHOD>

<FIELD>
    ::= <DECLARED_TYPE>? "#" <NAME>

<METHOD>
    ::= <DECLARED_TYPE>? "#" <TYPE_ARG_LIST>? <NAME> <PARAM_LIST>

<PARAM_LIST>
    ::= "(" ")"
    | "(" <PARAMETER> ( "," <PARAMETER> )* ")"

<PARAMETER>
    ::= <TYPE> <NAME>?

<TYPE_ARG_LIST>
    ::= "<" <TYPE_ARGUMENT> ( "," <TYPE_ARGUMENT> )* ">"

<PRIMITIVE_TYPE_ARRAY>
    ::= <PRIMITIVE_TYPE> ( "[" "]" )+

<DECLARED_TYPE>
    ::= <QUALIFIED_NAME> <TYPE_ARG_LIST>?

```

```

<DECLARED_TYPE_ARRAY>
    ::= <DECLARED_TYPE> ( "[" "]" )+

<TYPE>
    ::= <PRIMITIVE_TYPE>
       | <REFERENCE_TYPE>

<REFERENCE_TYPE>
    ::= <PRIMITIVE_TYPE_ARRAY>
       | <DECLARED_TYPE>
       | <DECLARED_TYPE_ARRAY>

<TYPE_ARGUMENT>
    ::= <TYPE>
       | <WILDCARD_TYPE>

<WILDCARD_TYPE>
    ::= "?"
       | "?" "extends" <REFERENCE_TYPE>
       | "?" "super" <REFERENCE_TYPE>

<OPERATOR>
    ::= "="
       | "in"
       | "=="
       | "!="
       | "<"
       | ">"
       | "<="
       | ">="
       | "<:"
       | ":>"
       | "=~"
       | "!~"

<LITERAL>
    ::= <INTEGER_LITERAL>
       | <REAL_LITERAL>
       | <BOOLEAN_LITERAL>
       | <CHARACTER_LITERAL>

```

```
| <STRING_LITERAL>
| <NULL_LITERAL>

<MODIFIER>
 ::= "public"
 | "protected"
 | "private"
 | "final"
 | "static"
 | "abstract"
 | "native"
 | "synchronized"
 | "volatile"
 | "transient"
 | "strictfp"

<BASIC_TYPE>
 ::= "void"
 | <PRIMITIVE_TYPE>

<PRIMITIVE_TYPE>
 ::= "int"
 | "long"
 | "float"
 | "double"
 | "byte"
 | "short"
 | "char"
 | "boolean"

<NAME>
 ::= (識別子: 第2章第1節第3項)

<INTEGER_LITERAL>
 ::= (整数リテラル: 第2章第1節第4項)

<REAL_LITERAL>
 ::= (浮動小数点数リテラル: 第2章第1節第5項)

<BOOLEAN_LITERAL>
```

::= (真偽リテラル: 第2章第1節第6項)

<CHARACTER\_LITERAL>

::= (文字リテラル: 第2章第1節第7項)

<STRING\_LITERAL>

::= (文字列リテラル 第2章第1節第8項)

<NULL\_LITERAL>

::= (nullリテラル: 第2章第1節第9項)

## 付録B. 利用可能なプロパティ

プロパティ名	DOM 要素	概要
annotations	CtReferece	付与された注釈の一覧
arguments	CtInvocation	実引数一覧
array	CtArrayAccess	参照先の配列を表す式
	CtArrayLength	
assertion	CtAssert	表明式
body	CtCatch	catch 節本体
	CtForEach	拡張 for 文のループ
	CtInitializer	初期化子のブロック
	CtInvocable	メソッド等のブロック
	CtJavadoc	Javadoc の内容
	CtSwitch	switch 文のブロック
	CtSynchronized	synchronized 文のブロック
bound	CtWildcard	ワイルドカード境界
boundKind	CtWildcard	ワイルドカード境界の種類
bounds	CtTypeParameter	型引数の境界型一覧
catches	CtTry	catch 節の一覧
componentType	CtArray	要素型
condition	CtConditional	条件式
	CtDo	
	CtFor	
	CtIf	
	CtWhile	
constants	CtEnum	宣言された列挙定数の一覧
constructors	CtClass	宣言されたコンストラクタの一覧
default	CtAnnotationElement	注釈要素の規定値
dimensions	CtNewArray	次元ごとの初期要素数一覧
elements	CtAnnotation	注釈要素一覧
	CtAnnotationInstance	注釈要素一覧
else	CtConditional	条件不成立時に実行される式
	CtIf	else 節
expression	CtCase	case ラベル
	CtCast	キャスト対象の式
	CtInstanceOf	比較されるオブジェクトを表す式
exists	CtElement	要素の存在(true/false)
fields	CtDeclaredType	宣言されたフィールドの一覧
finally	CtTry	finally 節
index	CtArrayAccess	添え字を表す式
initialElements	CtNewArray	初期要素を表す式の一覧
initializer	CtEnumConstant	列挙定数の初期化ブロック

プロパティ名	DOM 要素	概要
	CtField	フィールド宣言の初期化子
	CtLocalVariable	ローカル変数の初期化式
initializers	CtFor	for 文の初期化文一覧
iterable	CtForEach	拡張 for 文の反復対象式
javadoc	CtReference	付与された Javadoc
kind	CtEvent	イベントの種類を表す文字列
	CtModifier	修飾子の種類
labels	CtStatement	付与されたラベルの一覧
leftHandSide	CtAssignment	代入文の左辺
leftOperand	CtInfix	二項演算式の左項
lock	CtSynchronized	synchronized 文のロック式
master	CtReference	マスタ参照
members	CtDeclaredType	宣言されたメンバー一覧
	CtPackage	サブパッケージ、宣言型の一覧
message	CtAssert	assert 文のメッセージ
methods	CtDeclaredType	宣言されたメソッド一覧
modifiers	CtReference	付与された修飾子一覧
operand	CtUnary	単項演算の項
operator	CtAssignment	代入演算の演算子
	CtInfix	二項演算の演算子
	CtUnary	単項演算の演算子
parameter	CtCatch	catch 節の例外引数宣言
	CtForEach	拡張 for 文の変数宣言
parameters	CtInvocable	宣言された仮引数一覧
parent	CtElement	Irenka DOM 上での親要素
phase	HackEvent	イベントの発生時期
qualifier	CtDeclaredType	外部クラス
	CtInvocation	メソッド起動のレシーバオブジェクト
	CtVariableAccess	フィールドの所有者オブジェクト
result	CtReturn	return 文の戻り値
returnType	CtInvocable	メソッドの戻り値型
rightHandSide	CtAssignment	代入演算の右辺
rightOperand	CtInfix	二項演算の右項
selector	CtSwitch	switch 文のセレクト式
simpleName	CtNamedReference	名前付き宣言の単純名
statements	CtBlock	ブロックに含まれる式の一覧
	CtCase	case ラベルに続く式の一覧
superClass	CtType	親クラス
superInterfaces	CtType	親インターフェース一覧
target	CtInvocation	起動対象のメソッド/コンストラクタなど
targetLabel	CtJump	ジャンプ先のラベル
then	CtCondition	条件成立時に実行される式
	CtIf	then 節

プロパティ名	DOM 要素	概要
throwable	CtThrow	throw 文のスローする例外
throws	CtInvocable	throws 節
try	CtTry	try 節
type	CtAnnotationInstance	注釈の型
	CtCast	キャスト式のキャスト対象型
	CtInstanceof	instanceof 式の比較対象型
	CtNewArray	生成する配列の型
	CtThis	this が表現する型
	CtTypedReference	型付き宣言の型
typeArguments	CtGenericReference	適用される実型引数の一覧
typeParameters	CtGenericReference	宣言された型引数の一覧
updaters	CtFor	for 文の更新文一覧
value	CtAnnotationInstanceElement	注釈要素の持つ値
	CtClassLiteral	クラスリテラルが表す型
	CtLiteral	リテラルの表す値
variable	CtVariableAccess	参照対象の変数

## 参考文献

---

- Gosling, J., Joy, B., Steele, G., & Bracha, G. (2005). *The Java(TM) Language Specification* (Third ed.). Prentice Hall PTR.
- Irenka DOM Specification
  - <http://irenka.ashikunep.org/documents/spec/dom.pdf>
- Irenka End User API References
  - <http://irenka.ashikunep.org/documents/api>

## 用語索引

<b>@</b>		構造展開..... 10
@link..... 27		構文例外..... 37
<b>D</b>		コメント..... 2
DOM..... 7		コンストラクタ..... 31
DOM 要素..... 1, 7, 14, 16, 17, 20, 21, 22, 33		コンパイル例外..... 34
一次元リスト..... 14, 16, 17, 20, 21, 22		
<b>I</b>		<b>し</b>
Irenka DOM..... 1, 33		式 7, 8, 16, 22, 24, 36
Irenka Search Engine..... 33		評価..... 7, 8, 14, 16
Irenka Search Query..... 1		識別子..... 3
<b>N</b>		修飾子..... 25
null リテラル..... 5, 24		評価..... 25
<b>き</b>		捕捉..... 25
キーワード..... 3		循環依存例外..... 37
基本型..... 26		衝突制約例外..... 37
評価..... 26		初期検索対象..... 33, 34
捕捉..... 26		初期検索対象集合..... 36
<b>く</b>		真偽リテラル..... 4, 24
空白文字..... 2		
クエリ..... 7		<b>せ</b>
区切り子..... 6, 14, 15, 19, 22, 29, 31		整数リテラル..... 4, 24
<b>け</b>		制約..... 7, 8
検索対象..... 7, 33, 34		違反..... 8, 16, 17, 18
<b>こ</b>		関係制約..... 8
項 17, 19, 22		事前条件..... 8
		順序比較..... 12
		照合..... 14
		初期化..... 36
		同一..... 9, 19, 35, 36
		同値比較..... 11
		評価..... 8, 19
		評価に失敗..... 8
		包括..... 9, 22, 37
		満足..... 7, 8, 17, 18, 36

要素型制約.....	8, 14
制約演算子.....	7, 8
順序比較.....	12
照合.....	14
同一.....	9
同値比較.....	11
包括.....	9
セパレータ.....	6
宣言型.....	28
配列型.....	27, 29
捕捉.....	28

---

## そ

即値.....	24, 34
捕捉可能である.....	19

---

## た

単純宣言型.....	28, 30, 31
単純フィールド.....	30
単純メソッド.....	31

---

## ち

注釈要素.....	31
-----------	----

---

## て

定数式.....	36
定数リテラル.....	24
評価.....	24
捕捉.....	24

---

## は

配列型	
評価.....	26, 29
捕捉.....	27, 29
パラメータ化型.....	27, 28, 30, 31
パラメータ化フィールド.....	27, 30
パラメータ化メソッド.....	31

パラメータ化メソッド.....	27
-----------------	----

---

## ふ

フィールド.....	29
捕捉.....	30
不定.....	17
浮動小数点数リテラル.....	4, 24
プリミティブ型.....	26
配列型.....	26
ブレースホルダ.....	7, 17, 33
依存.....	35, 37
開放.....	36
循環依存.....	35
制約を課す.....	17, 18
定数.....	36
伝搬.....	36
同名.....	18
名前.....	17
評価.....	17
別名.....	35
プロパティ.....	19
解決.....	19
参照.....	19
参照先.....	17, 19, 22, 23, 38
名前.....	19, 38
評価.....	21
プロパティ解決例外.....	37
プロパティ参照.....	19, 34, 37

---

## ほ

捕捉変換.....	9, 11, 19, 22
-----------	---------------

---

## め

メソッド.....	31
捕捉.....	32

---

**も**

文字リテラル ..... 5, 24  
 文字列リテラル ..... 5, 24

---

**り**

リスト  
   構築 ..... 22  
   参照 ..... 22  
   参照先 ..... 17, 22, 23  
   添え字 ..... 22, 23

リスト構築 ..... 22, 34, 37  
   評価 ..... 22  
   要素式 ..... 22  
 リスト参照 ..... 22  
   評価 ..... 23  
 リンク参照 ..... 27, 37  
   評価 ..... 27  
 リンク参照解決例外 ..... 37

---

**れ**

列挙定数 ..... 29