注解（Annotations）

注解（Annotations）

注释一览表（Overview of annotations）

- abstract（抽象）
- cached （缓存）
- deprecated（已废弃）
- external（外部）
- tranisient（暂时）
- final（最终）
- libarary(库)
- override（重写）
- private（私有）
- query（查询）

编译相关注释（Compiler pragmas）

- Inlining（内联）
  - pragmas[inline]
  - pragmas[noinline]
  - pragmas[nomagic]
  - pragmas[noopt]

语言相关注解

- language[monotonicAggregates]

绑定设置(Binding sets)

- bindingset[...]

注解（Annotations）

注释是可以直接在QL实体或名称的声明之前放置的字符串。
例如，要将模块声明`M`为私有模块，可以使用：

```
1 private module M {
```

```
    ...
}
```

请注意，某些注释作用于实体本身，而另一些注释作用于实体的特定*名称*：

- **作用于实体**：`abstract`，`cached`，`external`，`transient`，`final`，`override`，`pragma`，`language`，和`bindingset`
- **作用于名称**：`deprecated`，`library`，`private`，和`query`

例如，如果您用private注释实体，则只有该特定名称是private。您仍然可以使用其他名称（使用别名）访问该实体。另一方面，如果使用缓存对实体进行注释，则实体本身也会被缓存。
这是一个明确的示例：

```
module M {
  private int foo() { result = 1 }
  predicate bar = foo/0;
}
```

在这个case中 select M::foo()是会抛编译错误，在这种情况下，因为名称foo是私有的。查询select M :: bar（）是有效的（给出结果1），因为名称栏是可见的，并且它是谓词foo的别名。
您可以将缓存应用于foo，但不能应用于bar，因为foo是实体的声明。

注释一览表（Overview of annotations）

本节描述了不同的注释的作用以及何时使用它们

abstract（抽象）

适用于:类(class)、成员谓词(member predicates)
抽象注释用于定义抽象实体。抽象谓词是没有主体的成员谓词。它们可以在任何类上定义，并且应在非抽象子类型中覆盖。
这是一个使用抽象谓词的示例。在QL中编写数据流分析时，常见的模式是定义配置类。除其他事项外，这种配置必须描述其跟踪的数据源。所有此类配置的超型可能看起来像这样

```
abstract class Configuration extends string {
  ...
  /** Holds if `source` is a relevant data flow source. */
  abstract predicate isSource(Node source);
  ...
```

```
6 }
```

然后您可以定义Configuration的子类型，该子类型继承谓词isSource，以描述特定的配置。任何非抽象子类型都必须（直接或间接）覆盖它，以描述它们各自跟踪的数据源。

换句话说，所有扩展Configuration的非抽象类都必须在其自身体内重写isSource，或者它们必须从另一个重写isSource的类继承：

```
class ConfigA extends Configuration {
  ...
  // provides a concrete definition of `isSource`
  override predicate isSource(Node source) { ... }
}
class ConfigB extends ConfigA {
  ...
  // doesn't need to override `isSource`, because it inherits it f
rom ConfigA
}
```

cached （缓存）

适用于:类(class)、成员谓词(member predicates)、代数数据类型（algebraic datatypes）、特征谓词（characteristic predicates）、非成员谓词（non-member predicates）、模块（modules）

所述`cached`注释指示的实体应在其整体进行评估，并存储在评估高速缓冲存储器。以后对该实体的所有引用将使用已经计算的数据。这会影响其他查询以及当前查询的引用。
例如，缓存需要很长时间才能评估并在许多地方重用的谓词可能会有所帮助。
您应该`cached`小心使用，因为这可能会导致意想不到的后果。例如，缓存的谓词可能会占用大量存储空间，并且可能阻止QL编译器基于上下文在每个使用谓词的地方优化谓词。但是，这可能是一个合理的权衡，因为只需要计算一次谓词即可。
如果使用注释类或模块`cached`，则还必须使用注释其主体中的所有非私有实体`cached`，否则会报告编译器错误。

deprecated（已废弃）

适用于：类(class)、成员谓词(member predicates)、代数数据类型（algebraic datatypes）、非成员谓词（non-member predicates）、模块（modules）、字段（fields）、别名（aliases）

不建议使用的注释将应用于已过时且计划在将来的QL版本中删除的名称。如果您的任何QL文件使用了不建议使用的名称，则应考虑重写它们以使用更新的替代名称。通常，不推荐使用的名称具有QLDoc注释，该注释告诉用户应使用哪个更新的元素。

例如，不赞成使用名称DataFlowNode并具有以下QLDoc注释：

```
/**
 * DEPRECATED: Use `DataFlow::Node` instead.
 *
 * An expression or function/class declaration,
 * viewed as a node in a data flow graph.
 */
deprecated class DataFlowNode extends @dataflownode {
  ...
}
```

当您在QL编辑器中使用名称DataFlowNode时，将出现此QLDoc注释。

external（外部）

适用于：非成员谓词（non-member predicates）
该`external`注释用于在谓词，定义一个外部的“模板”谓词。这类似于数据库谓词

tranisient（暂时）

适用于：非成员谓词（non-member predicates）
该`transient`注释将应用于也用注释的非成员谓词`external`，以指示在评估期间不应将它们缓存到磁盘。请注意，如果您尝试`transient`不使用`external`，则编译器将报告错误。

final（最终）

适用于：类(class)、成员谓词(member predicates)、字段（fields）
该`final`注释被施加到不能被重写或扩展的实体。换句话说，最终类不能用作任何其他类型的基类型，并且最终谓词或字段不能在子类中被覆盖。
如果您不希望子类更改特定实体的含义，则这很有用。
例如，如果元素具有名称，则谓词成立。它使用谓词检查此内容，并且子类更改此定义没有任何意义。在这种情况下，应该是最终的：`hasName(string name)` `name` `getName()` `hasName`

```
class Element ... {
  string getName() { result = ... }
  final predicate hasName(string name) { name = this.getName() }
}
```

libarary(库)

适用于：类（class）

```
重要的
不建议使用此注释。不用使用库注释名称，而是将其放在私有（或私有导入）模块中。
```

该`library`注释被应用到名字，你只能从内部参考 `.qll`文件。如果尝试从不具有`.qll`扩展名的文件中引用该名称，则QL编译器将返回错误。

override（重写）

适用于：成员谓词(member predicates)、字段（fields）

该`override`注释被用于指示一个定义覆盖从基类型的成员谓词或字段。
如果您在没有注释的情况下覆盖谓词或字段，则QL编译器会发出警告。

private（私有）

适用于：类(class)、成员谓词(member predicates)、代数数据类型（algebraic datatypes）、非成员谓词（non-member predicates）、模块（modules）、字段（fields）、别名（aliases）、导入（imports）
private注释用于定义私有的名称，无法被导出
如果名称具有注释`private`，或者通过带有注释的import语句访问`private`名称，则只能从当前模块的命名空间中引用该名称。

query（查询）

适用于：非成员谓词（non-member predicates）、别名（aliases）
的`query`注释用于打开一个谓词（或谓词别名）成查询。这意味着它是QL程序输出的一部分。

编译相关注释（Compiler pragmas）

适用于：特征谓词（characteristic predicates）、成员谓词(member predicates)、非成员谓词（non-member predicates）

以下编译器杂项会影响查询的编译和优化。除非遇到严重的性能问题，否则应避免使用这些注释。

Inlining（内联）

对于简单谓词，QL优化器有时会用谓词主体本身替换对谓词的调用。这称为内联。
例如，假设您有一个定义谓词one（int i）{i = 1}和对该谓词的调用... one（y）...。QL优化器可以将谓词内联到... y = 1 ....
您可以使用以下编译器杂注注释来控制QL优化器内联谓词的方式。

pragmas[inline]

该`pragma[inline]`注解告诉QL优化器总是内联注释谓词进入它被调用的地方。当谓词主体的整体计算非常昂贵时，这将很有用，因为它可以确保谓词在调用它的地方与其他上下文信息一起进行评估。

pragmas[noinline]

该`pragma[noinline]`注解是用来防止谓语被联到那里它调用的地方。实际上，当您已经将某些变量组合到一个“辅助”谓词中时，此注释很有用，以确保对关系进行一次评估。这可以帮助提高性能。QL优化器的内联可能会撤消辅助谓词的工作，因此最好使用进行注释`pragma[noinline]`。

pragmas[nomagic]

该`pragma[nomagic]`注解用来防止QL优化从谓语表演“魔术套”的优化。
这种优化涉及从谓词调用的上下文中获取信息，并将其推入谓词主体。这通常是有益的，因此`pragma[nomagic]`除非GitHub建议，否则不应该使用注释。
注意这`nomagic`意味着`noinline`。

pragmas[noopt]

该`pragma[noopt]`注解用来防止QL优化程序优化断言，除非它是编制和评估工作绝对必要的。
这很少有必要，并且`pragma[noopt]`除非GitHub推荐这样做，否则不应该使用注释，例如，以帮助解决性能问题。
使用此注解时，请注意以下问题：

1. QL优化器以有效的方式自动对复杂公式的连词进行排序。在`noopt`谓词中，将按您编写它们的顺序来精确评估它们。
2. QL优化器会自动创建中间合取词，以将某些公式“转换”为更简单的公式的结合。在`noopt`谓词中，必须显式地编写这些连接。特别是，您不能链接谓词调用或在cast上调用谓词 。您必须将它们编写为多

个连接，并对其进行显式排序。
例如，假设您具有以下定义：

```
class Small extends int {
  Small() { this in [1 .. 10] }
  Small getSucc() { result = this + 1}
}

predicate p(int i) {
  i.(Small).getSucc() = 2
}

predicate q(Small s) {
  s.getSucc().getSucc() = 3
}
```

如果添加`noopt`杂项，则必须重写谓词。例如：

```
pragma[noopt]
predicate p(int i) {
  exists(Small s | s = i and s.getSucc() = 2)
}

pragma[noopt]
predicate q(Small s) {
  exists(Small succ |
    succ = s.getSucc() and
    succ.getSucc() = 3
  )
}
```

语言相关注解

适用于:类(class)、特征谓词（characteristic predicates）、成员谓词(member predicates)、非成员谓词（non-member predicates）

language[monotonicAggregates]

此批注允许您使用**单调聚合**而不是标准QL 聚合。

绑定设置(Binding sets)

bindingset[...]

您可以使用此批注明确声明谓词的绑定集。绑定集是谓词参数的子集，因此，如果这些参数被约束为一组有限的值，那么谓词本身就是有限的（即，它求值为一组有限的元组）。
该`bindingset`批注采用逗号分隔的变量列表。每个变量必须是谓词的参数，可能包括`this`（对于特征谓词和成员谓词）和`result`（对于返回结果的谓词）。