分析Java中的数据流(Analyzing data flow in Java)

您可以使用CodeQL跟踪通过Java程序使用的数据流。

关于本文（About this article）

本文介绍了如何在Java的CodeQL库中实现数据流分析，并包括一些示例来帮助您编写自己的数据流查询。 以下各节描述如何将库用于本地数据流，全局数据流和污点跟踪。

本地数据流（Local data flow）

本地数据流是单个方法或可调用方法内的数据流。 本地数据流通常比全局数据流更容易，更快和更精确，并且足以用于许多查询。

使用本地数据流（Using local data flow）

58安全应急响应中心

本地数据流库位于模块DataFlow中，该模块定义了Node类，该类表示数据可以流经的任何元素。 节点分为表达式节点（ExprNode）和参数节点（ParameterNode）。 您可以使用成员谓词asExpr和asParameter在数据流节点和asExpr/asParameter之间进行映射：

```
class Node {
  /** Gets the expression corresponding to this node, if any. */
  Expr asExpr() { ... }

  /** Gets the parameter corresponding to this node, if any. */
  Parameter asParameter() { ... }

  ...
}
```

或使用谓词exprNode和parameterNode:

```
/**
 * Gets the node corresponding to expression `e`.
 */
ExprNode exprNode(Expr e) { ... }

/**
 * Gets the node corresponding to the value of parameter `p` at fu
nction entry.
 */
ParameterNode parameterNode(Parameter p) { ... }
```

谓词localFlowStep（Node nodeFrom，Node nodeTo）保存是否存在从节点nodeFrom到节点nodeTo的直接数据流边缘。您可以通过使用+和*运算符或通过使用预定义的递归谓词localFlow来递归地应用谓词，它等效于localFlowStep *。
例如，您可以在零个或多个本地步骤中找到从参数source到表达式sink的流：

使用本地污点跟踪(Using local taint tracking)

本地污点跟踪通过包括非保留值的流程步骤来扩展本地数据流。 例如：

```
String temp = x;
String y = temp + ", " + temp;
```

58安全应急响应中心

如果x是受污染的字符串，则y也将受污染。
本地污染跟踪库位于TaintTracking模块中。 像本地数据流一样，谓词localTaintStep（DataFlow :: Node nodeFrom，DataFlow :: Node nodeTo）在从节点nodeFrom到节点nodeTo的污点传播边缘立即存在的情况下成立。您可以通过使用+和*运算符，或通过使用预定义的递归谓词localTaint（与localTaintStep *等效）来递归地应用谓词。
例如，您可以在零个或多个本地步骤中找到从参数源到表达式接收器的异味传播：

```
TaintTracking::localTaint(DataFlow::parameterNode(source), DataFlow::exprNode(sink))
```

例子（Examples）

此查询查找传递给新FileReader（..）的文件名。

```
import java

from Constructor fileReader, Call call
where
  fileReader.getDeclaringType().hasQualifiedName("java.io", "FileReader") and
  call.getCallee() = fileReader
select call.getArgument(0)
```

不幸的是，这仅在参数中给出表达式，而不是可以传递给它的值。 因此，我们使用本地数据流查找流入该参数的所有表达式：

```
import java
import semmle.code.java.dataflow.DataFlow

from Constructor fileReader, Call call, Expr src
where
  fileReader.getDeclaringType().hasQualifiedName("java.io", "FileReader") and
  call.getCallee() = fileReader and
  DataFlow::localFlow(DataFlow::exprNode(src), DataFlow::exprNode(call.getArgument(0)))
select src
```

58安全应急响应中心

然后，我们可以使源更加具体，例如对公共参数的访问。 此查询查找将公共参数传递到新FileReader（..）的位置：

```
import java
import semmle.code.java.dataflow.DataFlow

from Constructor fileReader, Call call, Parameter p
where
  fileReader.getDeclaringType().hasQualifiedName("java.io", "FileReader") and
  call.getCallee() = fileReader and
  DataFlow::localFlow(DataFlow::parameterNode(p), DataFlow::exprNode(call.getArgument(0)))
select p
```

此查询查找对格式字符串未进行硬编码的格式化函数的调用。

```
import java
import semmle.code.java.dataflow.DataFlow
import semmle.code.java.StringFormat

from StringFormatMethod format, MethodAccess call, Expr formatString
where
  call.getMethod() = format and
  call.getArgument(format.getFormatStringIndex()) = formatString and
  not exists(DataFlow::Node source, DataFlow::Node sink |
    DataFlow::localFlow(source, sink) and
    source.asExpr() instanceof StringLiteral and
    sink.asExpr() = formatString
  )
select call, "Argument to String format method isn't hard-coded."
```

练习（Exercises）

练习1：编写一个查询，使用本地数据流查找所有用于创建java.net.URL的硬编码字符串。

58安全应急响应中心

全局数据流（Global data flow）

全局数据流跟踪整个程序中的数据流，因此比本地数据流更强大。 但是，全局数据流不如本地数据流精确，并且分析通常需要更多的时间和内存来执行。

使用全局数据流（Using global data flow）

您可以通过扩展类DataFlow :: Configuration来使用全局数据流库：

```
import semmle.code.java.dataflow.DataFlow

class MyDataFlowConfiguration extends DataFlow::Configuration {
  MyDataFlowConfiguration() { this = "MyDataFlowConfiguration" }

  override predicate isSource(DataFlow::Node source) {
    ...
  }

  override predicate isSink(DataFlow::Node sink) {
    ...
  }
}
```

这些谓词在配置中定义：

- isSource–定义数据可能从何处流入
- isSink–定义数据可能流向的汇聚点
- isBarrier–可选，限制数据流
- isAdditionalFlowStep–可选，添加其他流程步骤

特征谓词MyDataFlowConfiguration（）定义配置的名称，因此“ MyDataFlowConfiguration”应该是唯一的名称，例如，类的名称。

使用谓词hasFlow（DataFlow :: Node source，DataFlow :: Node sink）进行数据流分析：

```
from MyDataFlowConfiguration dataflow, DataFlow::Node source, Data
Flow::Node sink
where dataflow.hasFlow(source, sink)
select source, "Data flow to $@.", sink, sink.toString()
```

58安全应急响应中心

使用全局污点跟踪(Using global taint tracking)

全局污点跟踪是针对全局数据流，就像本地污点跟踪是针对本地数据流一样。 也就是说，全局污点跟踪通过附加的非保留值步骤扩展了全局数据流。 您可以通过扩展TaintTracking :: Configuration类来使用全局污染跟踪库：

```
import semmle.code.java.dataflow.TaintTracking

class MyTaintTrackingConfiguration extends TaintTracking::Configu
ration {
  MyTaintTrackingConfiguration() { this = "MyTaintTrackingConfigu
ration" }

  override predicate isSource(DataFlow::Node source) {
    ...
  }

  override predicate isSink(DataFlow::Node sink) {
    ...
  }
}
```

这些谓词在配置中定义：

- isSource–定义数据可能从何处流入
- isSink–定义数据可能流向的汇聚点
- isSanitizer–可选，限制数据流
- isAdditionalFlowStep–可选，添加其他流程步骤

与全局数据流类似，特征谓词MyTaintTrackingConfiguration（）定义配置的唯一名称。
使用谓词hasFlow（DataFlow :: Node source，DataFlow :: Node sink）进行污点跟踪分析。

数据流（Flow Sources）

数据流库包含一些预定义的流源。 类RemoteFlowSource（在semmle.code.java.dataflow.FlowSources中定义）表示可由远程用户控制的数据流源，这对于发现安全性问题很有用。

例子（Examples）

此查询显示使用远程用户输入作为数据源的污点跟踪配置。

360安全应急响应中心

```
import java
import semmle.code.java.dataflow.FlowSources

class MyTaintTrackingConfiguration extends TaintTracking::Configu
ration {
  MyTaintTrackingConfiguration() {
    this = "..."
  }

  override predicate isSource(DataFlow::Node source) {
    source instanceof RemoteFlowSource
  }

  ...
}
```

练习（Exercises）

练习2：编写查询，使用全局数据流查找用于创建java.net.URL的所有硬编码字符串

练习3：编写一个表示来自java.lang.System.getenv（..）的流源的类

练习4：使用2和3的答案，编写一个查询，查找从getenv到java.net.URL的所有全局数据流。

练习答案（Answers）

练习一

```
import semmle.code.java.dataflow.DataFlow

from Constructor url, Call call, StringLiteral src
where
  url.getDeclaringType().hasQualifiedName("java.net", "URL") and
  call.getCallee() = url and
  DataFlow::localFlow(DataFlow::exprNode(src), DataFlow::exprNode(
call.getArgument(0)))
select src
```

58安全应急响应中心

练习二

```
import semmle.code.java.dataflow.DataFlow

class Configuration extends DataFlow::Configuration {
  Configuration() {
    this = "LiteralToURL Configuration"
  }

  override predicate isSource(DataFlow::Node source) {
    source.asExpr() instanceof StringLiteral
  }

  override predicate isSink(DataFlow::Node sink) {
    exists(Call call |
      sink.asExpr() = call.getArgument(0) and
      call.getCallee().(Constructor).getDeclaringType().hasQualifiedName("java.net", "URL")
    )
  }
}

from DataFlow::Node src, DataFlow::Node sink, Configuration config
where config.hasFlow(src, sink)
select src, "This string constructs a URL $@.", sink, "here"
```

练习三

```
import java

class GetenvSource extends MethodAccess {
  GetenvSource() {
    exists(Method m | m = this.getMethod() |
      m.hasName("getenv") and
      m.getDeclaringType() instanceof TypeSystem
    )
```

58安全应急响应中心

```
  }
}
```

练习四

```
import semmle.code.java.dataflow.DataFlow

class GetenvSource extends DataFlow::ExprNode {
  GetenvSource() {
    exists(Method m | m = this.asExpr().(MethodAccess).getMethod(
) |
      m.hasName("getenv") and
      m.getDeclaringType() instanceof TypeSystem
    )
  }
}

class GetenvToURLConfiguration extends DataFlow::Configuration {
  GetenvToURLConfiguration() {
    this = "GetenvToURLConfiguration"
  }

  override predicate isSource(DataFlow::Node source) {
    source instanceof GetenvSource
  }

  override predicate isSink(DataFlow::Node sink) {
    exists(Call call |
      sink.asExpr() = call.getArgument(0) and
      call.getCallee().(Constructor).getDeclaringType().hasQualif
iedName("java.net", "URL")
    )
  }
}

from DataFlow::Node src, DataFlow::Node sink, GetenvToURLConfigur
ation config
where config.hasFlow(src, sink)
```

58安全应急响应中心

```
31 select src, "This environment variable constructs a URL $@.", sin
   k, "here"
```

58安全应急响应中心