# **GraphFrames用户指南**

本页提供了如何使用GraphFrames进行基本查询，主题查找和一般图形算法的示例。这包括Scala和Python中的代码示例。

* [创建](http://graphframes.github.io/user-guide.html" \l "creating-graphframes)GraphFrames
* [基本图和](http://graphframes.github.io/user-guide.html" \l "basic-graph-and-dataframe-queries)DataFrame查询
* [主题发现](http://graphframes.github.io/user-guide.html" \l "motif-finding)
* [子图](http://graphframes.github.io/user-guide.html" \l "subgraphs)
* [图形算法](http://graphframes.github.io/user-guide.html" \l "graph-algorithms)
  + [宽度搜索（](http://graphframes.github.io/user-guide.html" \l "breadth-first-search-bfs)BFS）
  + [连接组件](http://graphframes.github.io/user-guide.html" \l "connected-components)
    - [强连接组件](http://graphframes.github.io/user-guide.html" \l "strongly-connected-components)
  + [标签传播算法（](http://graphframes.github.io/user-guide.html" \l "label-propagation-algorithm-lpa)LPA）
  + [网页排名](http://graphframes.github.io/user-guide.html" \l "pagerank)
  + [最短路径](http://graphframes.github.io/user-guide.html" \l "shortest-paths)
  + [三角计数](http://graphframes.github.io/user-guide.html" \l "triangle-count)
* [保存并加载](http://graphframes.github.io/user-guide.html" \l "saving-and-loading-graphframes)GraphFrames
* [消息传递通过](http://graphframes.github.io/user-guide.html" \l "message-passing-via-aggregatemessages)AggregateMessages
* [GraphX-GraphFrame](http://graphframes.github.io/user-guide.html" \l "graphx-graphframe-conversions)转换
  + [GraphFrame](http://graphframes.github.io/user-guide.html" \l "graphframe-to-graphx)到GraphX
  + [GraphX](http://graphframes.github.io/user-guide.html" \l "graphx-to-graphframe)到GraphFrame
  + [示例转换](http://graphframes.github.io/user-guide.html" \l "example-conversions)

*注意：大多数示例使用第一个小节中的GraphFrame： [创建](http://graphframes.github.io/user-guide.html" \l "creating-graphframes)GraphFrames。*

# **创建GraphFrames**

用户可以从顶点和边缘的DataFrames创建GraphFrames。

* *顶点DataFrame*：顶点DataFrame应该包含一个名为“id”的特殊列，它为图中的每个顶点指定唯一的ID。
* *Edge DataFrame*：边缘DataFrame应包含两个特殊列：“src”（边缘的源顶点ID）和“dst”（边缘的目标顶点ID）。

两个DataFrames都可以有任意其他列。那些列可以表示顶点和边缘属性。

GraphFrame也可以从包含边缘信息的单个DataFrame构造。顶点将从边缘的来源和目的地推断出来。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_0)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_0)**

以下示例演示如何从顶点和边缘DataFrames创建GraphFrame。

*// Vertex DataFrame*

**val** v **=** sqlContext.createDataFrame(**List**(

("a", "Alice", 34),

("b", "Bob", 36),

("c", "Charlie", 30),

("d", "David", 29),

("e", "Esther", 32),

("f", "Fanny", 36),

("g", "Gabby", 60)

)).toDF("id", "name", "age")

*// Edge DataFrame*

**val** e **=** sqlContext.createDataFrame(**List**(

("a", "b", "friend"),

("b", "c", "follow"),

("c", "b", "follow"),

("f", "c", "follow"),

("e", "f", "follow"),

("e", "d", "friend"),

("d", "a", "friend"),

("a", "e", "friend")

)).toDF("src", "dst", "relationship")

*// Create a GraphFrame*

**val** g **=** **GraphFrame**(v, e)

以上构建的GraphFrame在GraphFrames软件包中可用：

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends

# **基本图和DataFrame查询**

GraphFrames提供了几个简单的图形查询，如节点度。

此外，由于GraphFrames表示图形为顶点和边缘DataFrames，因此可以直接在顶点和边缘DataFrames上进行强大的查询。这些DataFrames是可用的vertices和edges领域的GraphFrame。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_1)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_1)**

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

*// Display the vertex and edge DataFrames*

g.vertices.show()

*// +--+-------+---+*

*// |id| name|age|*

*// +--+-------+---+*

*// | a| Alice| 34|*

*// | b| Bob| 36|*

*// | c|Charlie| 30|*

*// | d| David| 29|*

*// | e| Esther| 32|*

*// | f| Fanny| 36|*

*// | g| Gabby| 60|*

*// +--+-------+---+*

g.edges.show()

*// +---+---+------------+*

*// |src|dst|relationship|*

*// +---+---+------------+*

*// | a| b| friend|*

*// | b| c| follow|*

*// | c| b| follow|*

*// | f| c| follow|*

*// | e| f| follow|*

*// | e| d| friend|*

*// | d| a| friend|*

*// | a| e| friend|*

*// +---+---+------------+*

*// import Spark SQL package*

**import** **org.apache.spark.sql.DataFrame**

*// Get a DataFrame with columns "id" and "inDeg" (in-degree)*

**val** vertexInDegrees**:** DataFrame = g.inDegrees

*// Find the youngest user's age in the graph.*

*// This queries the vertex DataFrame.*

g.vertices.groupBy().min("age").show()

*// Count the number of "follows" in the graph.*

*// This queries the edge DataFrame.*

**val** numFollows **=** g.edges.filter("relationship = 'follow'").count()

# **主题发现**

主题查找是指在图中搜索结构模式。

GraphFrame主题查找使用简单的域特定语言（DSL）来表达结构性查询。例如，graph.find("(a)-[e]->(b); (b)-[e2]->(a)")将搜索a,b通过两个方向的边缘连接的顶点对 。它将DataFrame在图中返回所有这样的结构，列中的每个命名元素（顶点或边）在图案中。在这种情况下，返回的列将为“a，b，e，e2”。

用于表达结构模式的DSL：

* 图案的基本单位是边缘。例如，"(a)-[e]->(b)"表示e从顶点a到顶点的边b。注意，顶点用圆括号表示(a)，而边用方括号表示[e]。
* 模式表示为边的并集。边缘图案可以用分号连接。主题"(a)-[e]->(b); (b)-[e2]->(c)"指定从两边a到b到c。
* 在一个模式中，可以将名称分配给顶点和边。例如， "(a)-[e]->(b)"有三个命名元素：顶点a,b和边e。这些名称有两个目的：
  + 这些名称可以识别边缘之间的共同元素。例如， "(a)-[e]->(b); (b)-[e2]->(c)"指定相同的顶点b是边的目的地和边的e源e2。
  + 这些名称在结果中用作列名称DataFrame。如果一个主题包含命名顶点a，则结果DataFrame将包含一列“a”，它与一个 StructType等同于模式（列）的子字段 GraphFrame.vertices。类似地，e图案中的边缘将在结果中产生DataFrame与“模式（列）”相等的子字段 的列“e” GraphFrame.edges。
* 在不需要时，可以省略图案中的顶点或边缘的名称。例如，"(a)-[]->(b)"在顶点之间表达边缘，a,b但不为边缘分配名称。结果中不存在匿名边缘的列DataFrame。类似地，"(a)-[e]->()"表示顶点的外边缘，a但不指定目标顶点。
* 边可以被否定，以表示图中*不*应该出现边。例如，"(a)-[]->(b); !(b)-[]->(a)"从发现的边缘a到b对于其存在*没有* 边缘从b给a。

可以通过对结果应用过滤器来表达更复杂的查询，例如对顶点或边缘属性进行操作的查询DataFrame。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_2)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_2)**

有关API详细信息，请参阅[API](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.GraphFrame)文档。

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

*// Search for pairs of vertices with edges in both directions between them.*

**val** motifs**:** DataFrame = g.find("(a)-[e]->(b); (b)-[e2]->(a)")

motifs.show()

*// More complex queries can be expressed by applying filters.*

motifs.filter("b.age > 30").show()

许多主题查询是无状态的并且简单的表达，如上面的例子。下面的例子展示了更复杂的查询，它沿着主题中的路径传递状态。这些查询可以通过将GraphFrame主题查找与结果中的过滤器相结合来表达，其中过滤器使用序列操作构建一系列DataFrame Columns。

例如，假设一个人希望用一系列函数来确定具有某些属性的4个顶点的链。也就是说，在4个顶点的链中a->b->c->d，标识匹配此复杂过滤器的链的子集：

* 在路径上初始化状态。
* 基于顶点的更新状态a。
* 基于顶点的更新状态b。
* 等等。c和d。
* 如果最终状态匹配某个条件，则该链接被过滤器接受。

下面的代码片段演示了这个过程，其中我们识别4个顶点的链，使得3个边中的至少2个是“朋友”关系。在这个例子中，状态是当前的“朋友”边的数量; 一般来说，它可以是任何的 DataFrame Column。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_3)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_3)**

**import** **org.apache.spark.sql.Column**

**import** **org.apache.spark.sql.functions.**{col, when}

**import** **org.graphframes.examples**

**val** g **=** examples.**Graphs**.friends *// get example graph*

*// Find chains of 4 vertices.*

**val** chain4 **=** g.find("(a)-[ab]->(b); (b)-[bc]->(c); (c)-[cd]->(d)")

*// Query on sequence, with state (cnt)*

*// (a) Define method for updating state given the next element of the motif.*

**def** sumFriends(cnt**:** Column, relationship**:** Column)**:** Column = {

when(relationship === "friend", cnt + 1).otherwise(cnt)

}

*// (b) Use sequence operation to apply method to sequence of elements in motif.*

*// In this case, the elements are the 3 edges.*

**val** condition **=** **Seq**("ab", "bc", "cd").

foldLeft(lit(0))((cnt, e) **=>** sumFriends(cnt, col(e)("relationship")))

*// (c) Apply filter to DataFrame.*

**val** chainWith2Friends2 **=** chain4.where(condition >= 2)

chainWith2Friends2.show()

上述例子证明了固定长度链的有状态图案。目前，为了搜索可变长度的图案，用户需要为每个可能的长度运行一个查询。然而，上述查询模式允许用户为每个长度重新使用相同的代码，唯一的变化是更新主题元素（“ab”，“bc”，“cd”）的序列。

# **子图**

在GraphX中，该subgraph()方法采用边缘三元组（边缘，顶点和顶点，加上属性），并允许用户根据三元组和顶点过滤器选择子图。

GraphFrames提供了一种更为强大的方式，可以基于主题查找和DataFrame过滤器的组合来选择子图。

**简单子图：顶点和边缘过滤器**：以下示例显示了如何根据顶点和边缘过滤器选择子图。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_4)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_4)**

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends

*// Select subgraph of users older than 30, and edges of type "friend"*

**val** v2 **=** g.vertices.filter("age > 30")

**val** e2 **=** g.edges.filter("relationship = 'friend'")

**val** g2 **=** **GraphFrame**(v2, e2)

**复数子图：三元组滤波器**：以下示例显示了如何根据在边缘上操作的三元组滤波器及其src和dst顶点来选择子图。这个例子可以通过使用更复杂的图案来扩展到超越三元组。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_5)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_5)**

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

*// Select subgraph based on edges "e" of type "follow"*

*// pointing from a younger user "a" to an older user "b".*

**val** paths **=** g.find("(a)-[e]->(b)")

.filter("e.relationship = 'follow'")

.filter("a.age < b.age")

*// "paths" contains vertex info. Extract the edges.*

**val** e2 **=** paths.select("e.src", "e.dst", "e.relationship")

*// In Spark 1.5+, the user may simplify this call:*

*// val e2 = paths.select("e.\*")*

*// Construct the subgraph*

**val** g2 **=** **GraphFrame**(g.vertices, e2)

# **图形算法**

GraphFrames提供与GraphX相同的标准图算法套件，还有一些新的。我们在下面提供简要说明和代码段。有关详细信息，请参阅[API](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.lib)文档。

一些算法目前是围绕GraphX实现的包装器，因此它们可能不比GraphX更具可扩展性。将来会有更多的算法迁移到本地的GraphFrames实现。

## **宽度搜索（BFS）**

宽度优先搜索（BFS）从一个顶点（或一组顶点）到另一个顶点（或一组顶点）找到最短路径。开始和结束顶点被指定为Spark DataFrame表达式。

有关更多背景，请参阅[BFS上的维基百科](https://en.wikipedia.org/wiki/Breadth-first_search)。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_6)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_6)**

以下代码片段搜索连接到用户“Bob”的人。

有关API详细信息，请参阅[API](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.lib.BFS)文档。

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

*// Search from "Esther" for users of age <= 32.*

**val** paths**:** DataFrame = g.bfs.fromExpr("name = 'Esther'").toExpr("age < 32").run()

paths.show()

*// Specify edge filters or max path lengths.*

g.bfs.fromExpr("name = 'Esther'").toExpr("age < 32")

.edgeFilter("relationship != 'friend'")

.maxPathLength(3)

.run()

## **连接组件**

计算每个顶点的连接组件成员资格，并返回一个图，每个顶点分配一个组件ID。

查看[维基百科](https://en.wikipedia.org/wiki/Connected_component_(graph_theory))的背景。

注意：使用GraphFrames 0.3.0及更高版本，默认的连接组件算法需要设置Spark检查点目录。用户可以使用旧的算法恢复.setAlgorithm("graphx")。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_7)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_7)**

有关API详细信息，请参阅[API](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.lib.ConnectedComponents)文档。

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

**val** result **=** g.connectedComponents.run()

result.select("id", "component").orderBy("component").show()

### **强连接组件**

计算每个顶点的强连接组件（SCC），并返回一个图，每个顶点分配给包含该顶点的SCC。

查看[维基百科](https://en.wikipedia.org/wiki/Strongly_connected_component)的背景。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_8)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_8)**

有关API详细信息，请参阅[API](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.lib.StronglyConnectedComponents)文档。

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

**val** result **=** g.stronglyConnectedComponents.maxIter(10).run()

result.select("id", "component").orderBy("component").show()

## **标签传播算法（LPA）**

运行静态标签传播算法，用于检测网络中的社区。

网络中的每个节点最初分配给自己的社区。在每个超级节点上，节点将社区隶属关系发送给所有邻居，并将其状态更新为传入消息的模式社区归属。

LPA是图形的标准社区检测算法。虽然（1）不能保证收敛，但是（2）可以用简单的解决方案（所有节点都被识别为一个单一的社区），这是非常便宜的。

查看[维基百科](https://en.wikipedia.org/wiki/Label_Propagation_Algorithm)的背景。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_9)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_9)**

有关API详细信息，请参阅[API](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.lib.LabelPropagation)文档。

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

**val** result **=** g.labelPropagation.maxIter(5).run()

result.select("id", "label").show()

## **网页排名**

有两个PageRank的实现。

* 第一个实现使用独立GraphFrame接口并运行PageRank来进行固定次数的迭代。这可以通过设置来运行maxIter。
* 第二个实现使用org.apache.spark.graphx.Pregel接口并运行PageRank直到收敛。这可以通过设置来运行tol。

这两种实现都支持非个性化和个性化的PageRank，其中设置sourceId 个性化该顶点的结果。

查看[维基百科](https://en.wikipedia.org/wiki/PageRank)的背景。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_10)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_10)**

有关API详细信息，请参阅[API](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.lib.PageRank)文档。

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

*// Run PageRank until convergence to tolerance "tol".*

**val** results **=** g.pageRank.resetProbability(0.15).tol(0.01).run()

*// Display resulting pageranks and final edge weights*

*// Note that the displayed pagerank may be truncated, e.g., missing the E notation.*

*// In Spark 1.5+, you can use show(truncate=false) to avoid truncation.*

results.vertices.select("id", "pagerank").show()

results.edges.select("src", "dst", "weight").show()

*// Run PageRank for a fixed number of iterations.*

**val** results2 **=** g.pageRank.resetProbability(0.15).maxIter(10).run()

*// Run PageRank personalized for vertex "a"*

**val** results3 **=** g.pageRank.resetProbability(0.15).maxIter(10).sourceId("a").run()

## **最短路径**

计算从每个顶点到给定的一组地标顶点的最短路径，其中地标由顶点ID指定。请注意，这将考虑边缘方向。

查看[维基百科](https://en.wikipedia.org/wiki/Shortest_path_problem)的背景。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_11)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_11)**

有关API详细信息，请参阅[API](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.lib.ShortestPaths)文档。

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

**val** results **=** g.shortestPaths.landmarks(**Seq**("a", "d")).run()

results.select("id", "distances").show()

## **三角计数**

计算通过每个顶点的三角形数量。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_12)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_12)**

有关API详细信息，请参阅[API](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.lib.TriangleCount)文档。

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

**val** results **=** g.triangleCount.run()

results.select("id", "count").show()

# **保存并加载GraphFrames**

由于GraphFrames是围绕DataFrames构建的，因此它们自动支持从同一组数据源中保存和加载数据。有关详细信息，请参阅[数据源上](http://spark.apache.org/docs/latest/sql-programming-guide.html" \l "data-sources)的“ [Spark SQL](http://spark.apache.org/docs/latest/sql-programming-guide.html" \l "data-sources)用户指南” 。

以下示例显示如何保存并加载图形。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_13)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_13)**

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

*// Save vertices and edges as Parquet to some location.*

g.vertices.write.parquet("hdfs://myLocation/vertices")

g.edges.write.parquet("hdfs://myLocation/edges")

*// Load the vertices and edges back.*

**val** sameV **=** sqlContext.read.parquet("hdfs://myLocation/vertices")

**val** sameE **=** sqlContext.read.parquet("hdfs://myLocation/edges")

*// Create an identical GraphFrame.*

**val** sameG **=** **GraphFrame**(sameV, sameE)

# **消息传递通过AggregateMessages**

像GraphX一样，GraphFrames提供用于开发图形算法的原语。两个关键组成部分是：

* aggregateMessages：在顶点之间发送消息，并为每个顶点聚合消息。GraphFrames提供aggregateMessages使用DataFrame操作实现的本机方法。这可以类似于GraphX API使用。
* 连接：使用原始图形加入消息聚合。GraphFrames依靠DataFrame联接，它提供了GraphX连接的完整功能。

以下代码片段显示了如何使用aggregateMessages来计算相邻用户年龄的总和。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_14)**
* **[蟒蛇](http://graphframes.github.io/user-guide.html" \l "tab_python_14)**

有关API详细信息，请参阅[API](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.lib.AggregateMessages)文档。

**import** **org.graphframes.examples**

**import** **org.graphframes.lib.AggregateMessages**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

*// We will use AggregateMessages utilities later, so name it "AM" for short.*

**val** **AM** **=** **AggregateMessages**

*// For each user, sum the ages of the adjacent users.*

**val** msgToSrc **=** **AM**.dst("age")

**val** msgToDst **=** **AM**.src("age")

**val** agg **=** g.aggregateMessages

.sendToSrc(msgToSrc) *// send destination user's age to source*

.sendToDst(msgToDst) *// send source user's age to destination*

.agg(sum(**AM**.msg).as("summedAges")) *// sum up ages, stored in AM.msg column*

agg.show()

对于一个更复杂的例子，看看用于实现[信仰传播示例](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.examples.BeliefPropagation$)的代码 。

# **GraphX-GraphFrame转换**

我们提供用于在GraphFrame和GraphX图之间进行转换的实用程序。有关[GraphX](http://spark.apache.org/docs/latest/graphx-programming-guide.html) 的详细信息，请参阅[GraphX用户指南](http://spark.apache.org/docs/latest/graphx-programming-guide.html)。

## **GraphFrame到GraphX**

转换为GraphX会创建一个GraphX Graph，它具有Long顶点ID和类型的属性Row。

顶点和边的属性是在原始的行vertices和edges，分别。

请注意，顶点（和边缘）属性包括顶点ID（和源，目标ID），以支持非长顶点ID。如果顶点ID不能转换为Long值，则为了生成相应的Long顶点ID（这是一个昂贵的操作），这些值被索引。

返回的Graph顶点和边缘属性的列排序分别由GraphFrame.vertexColumns和指定 GraphFrame.edgeColumns。

## **GraphX到GraphFrame**

GraphFrame提供了两种转换方法。第一个采用任何GraphX图形，并使用模式推理将顶点和边缘RDDs转换为DataFrames。那些DataFrames然后被用来创建一个GraphFrame。

第二种转换方法更复杂，对于现有GraphX代码的用户是有用的。其主要目的是支持以下形式的工作流程：（1）将GraphFrame转换为GraphX，（2）运行GraphX代码以增加具有新顶点或边缘属性的GraphX图，（3）将新属性合并回原始GraphFrame。

例如，给出：

* GraphFrame originalGraph
* GraphX Graph[String, Int] graph与一个String顶点属性我们想调用“category”和一个Int边缘属性我们想调用“count”

我们可以调用fromGraphX(originalGraph, graph, Seq("category"), Seq("count"))生成一个新的GraphFrame。新的GraphFrame将是一个增强版本originalGraph，新的GraphFrame.vertices列“category”和新的GraphFrame.edges列“count”被添加。

例如使用，看看用于实现[信仰传播示例](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.examples.BeliefPropagation$)的代码 。

## **示例转换**

以下示例演示了简单的GraphFrame-GraphX转换。

* **[斯卡拉](http://graphframes.github.io/user-guide.html" \l "tab_scala_15)**

有关API详细信息，请参阅API文档：

* [toGraphX](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.GraphFrame)
* [fromGraphX](http://graphframes.github.io/api/scala/index.html" \l "org.graphframes.GraphFrame$)

**import** **org.apache.spark.graphx.Graph**

**import** **org.apache.spark.Row**

**import** **org.graphframes.examples**

**val** g**:** GraphFrame = examples.**Graphs**.friends *// get example graph*

*// Convert to GraphX*

**val** gx**:** Graph[Row, Row] **=** g.toGraphX

*// Convert back to GraphFrame.*

*// Note that the schema is changed because of constraints in the GraphX API.*

**val** g2**:** GraphFrame = **GraphFrame**.fromGraphX(gx)

这些转换仅在Scala中受支持，因为GraphX没有Python API。