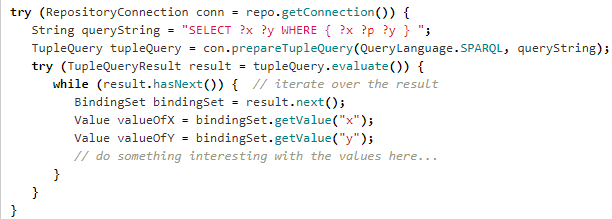
1. 查询仓库

一共有三种查询：元组查询、图查询和布尔查询。

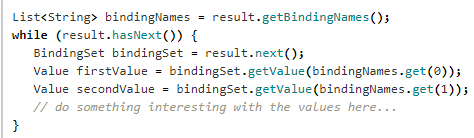
1. 元组查询

元组查询的结果是一系列元组，每一个元组称为一个solution，通常用于从存储的RDF数据中获取特定的值（uri、空白节点、数值），SPARQL的SELECT语句就是元组查询。

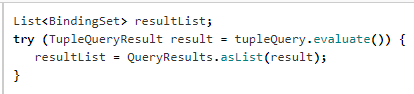
* 1. SELECT查询
     1. 查询返回TupleQueryResult，它由一系列BindingSet对象组成。每一个BindingSet包含一组Binding对象。每个Binding对象是变量名与值联系起来的一对。



进行遍历时，可以按照下面代码写。



还可以将结果全部放进集合，然后再进行遍历



* 1. 单行代码进行元组查询



* 1. 使用TupleQueryResultHandlers

可以将TupleQueryResultHandlers直接用于evaluate()方法来一下获得结果。以下代码是直接将结果全部输出在控制台。



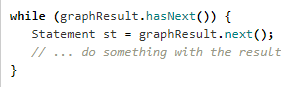
1. 图查询

结果是RDF图或一组语句。从存储的RDF数据库中提取子图很有用，然后继续查询子图——类似于关系型数据库里面的视图。SPARQL的CONSTRUCT和 DESCRIBE就是图查询。

* 1. 图查询代码



* 1. 遍历结果



* 1. 可以快速转换为Model



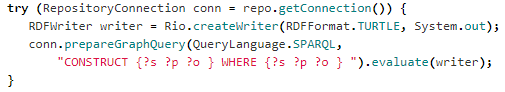
* 1. 单行图查询



1. 布尔查询
   1. 结果是一个简单的布尔值，用于查询是否包含某些特定的信息。SPARQL的ASK就是布尔查询。
2. 使用RDFHandlers

对于图查询，可以使用RDFHandlers与evaluate()结合，然后处理返回的RDF语句。所有的Rio writers（例如：RDFXmlWriter, TurtleWriter, TriXWriter）都继承自RDFHandler这个接口。用来转换各种格式。

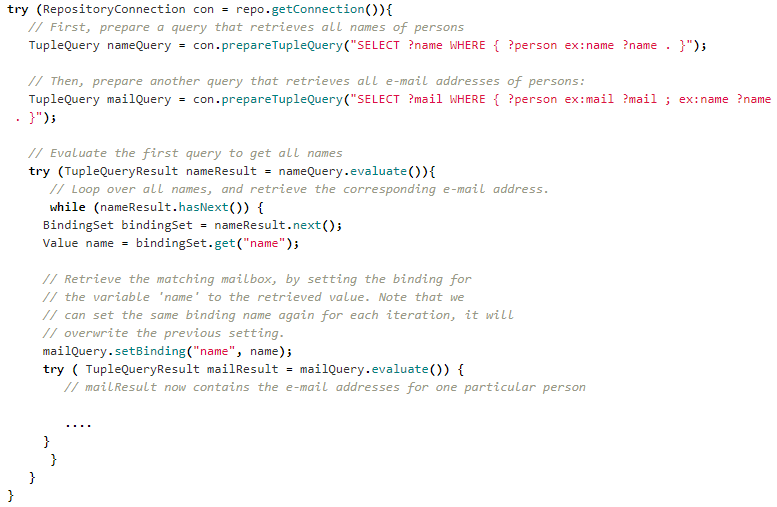
1. 将图查询的结果以Turtle格式写出来



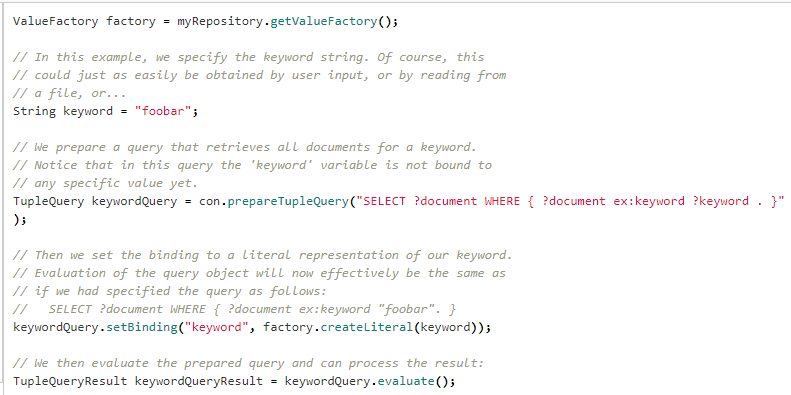
1. 填充查询

联合查询或定值查询。

1. 以下例子是先查询出有哪些人，然后再进行一次查询，查出这些人对应的邮箱。问题的重点是：查询到一个人对应的邮箱，主要使用setBinding()方法。



1. 给定值查询

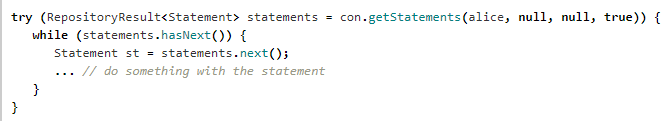


1. 创建、检索和删除单个语句
2. RepositoryConnection可以用于添加、检索、删除或以其他方式操作statement或statements。为了能够添加新的statements，可以使用ValueFactory来创建statements所包含的值。

例如：

除了上述方式，通常的场景是需要将另一个查询的结果作为一部分条件重新查询，因此可以使用ModelBuilde快速地创建语句集合。

1. 快速返回符合要求的statements



在getStatements()调用中，四个参数的前三个表示应该检索的RDF语句的主语、谓语和宾语。null值表示通配符，因此上面的方法调用检索所有以Alice为主题的语句，以及任何谓词和对象。可选的第四个参数指示是否应该包含推断语句(可以省略这个参数，默认为true )。

1. 删除statements

con.remove(alice, name, alicesName);

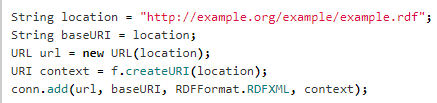
con.remove(alice, null, null);

1. 使用命名图和上下文

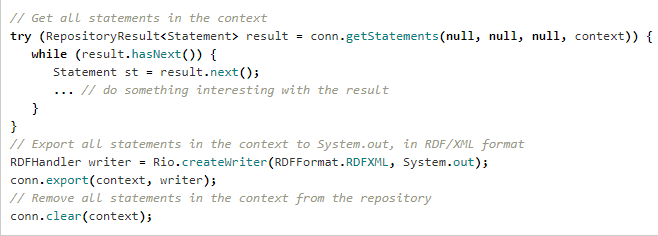
使用上下文的一种非常典型的方法是跟踪存储库中语句的出处，即这些语句来自哪个文件。在将来自不同文件的RDF数据添加到存储库中，更新其中一个文件，然后只想替换单个文件中的数据，为了做到这一点，您需要一种方法来确定需要删除哪些语句。上下文机制提供了一种方法。

另一个典型的用例是支持命名图，相当于视图。在SPARQL查询语言中，可以将命名图作为计算查询的数据集的子集进行查询。在rdf4j中，命名图是通过上下文机制实现的。这意味着，如果将数据放在rdf4j上下文中，就可以在SPARQL中以命名图的形式查询该上下文中的数据。

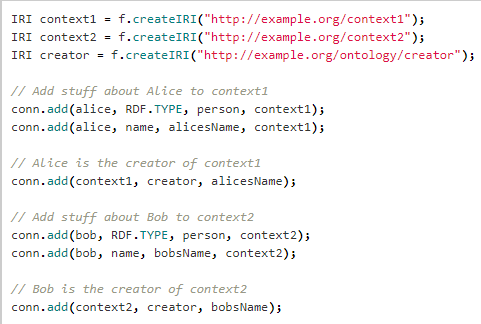
1. context简单使用，相当于使用context控制一个范围，只在这个范围内进行操作。同时可以指定多个context，可以对不同的context进行组合。
   1. 定义context



* 1. 然后使用上下文机制在存储库中专门处理这些语句，以便检索和删除操作。



1. 创建两个上下文例子
   1. 在下面的示例中，添加了关于Bob和Alice的信息，但这次每个人都有自己的上下文。还创建了一个名为“creator”的新属性，它的值是特定上下文的创建者的名字。然而，关于上下文创造者的信息不添加到任何特定的上下文中。



* 1. 之后可以获取context1与context2的statements



上面代码返回是不包含没有加进上下文的creator信息的。

* 1. 只查询非上下文的statements



上面代码只以creator信息为范围进行查询，因为这是唯一没有指定上下文的statements。

* 1. 查询指定上下文和非上下文

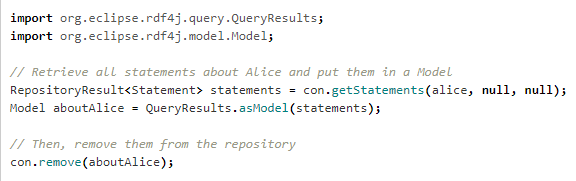


因此，可以自由的组合上下文与非上下文。

1. Models, Collections and Iterations

不同于之前都是对单个语句进行处理，也有处理集合的方法。Repository API提供了Connections等类，其中包含一些有用的实用程序函数，专门用于从存储库检索RDF集合。

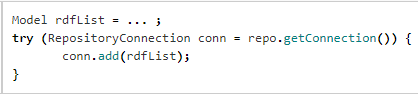
1. 下面的例子查询到一个语句集合，然后直接删掉。



1. 或者用下面：



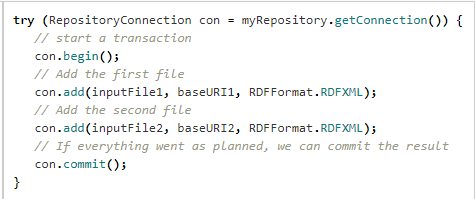
1. 另外，还可以直接使用add()增加。



1. 查询出所有语句，放入一个model



1. Transactions
2. 以上都是一些独立的操作，增加、删除等。事务也是支持的。

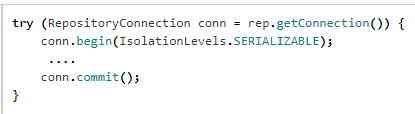


上述代码虽然没有con.rollback()，但是是有这个功能的。

1. 事务级别

默认是SNAPSHOT\_READ

人为指定方式：



1. 自动事务处理



以上两段代码效果是一样的。但是当想要改变事务级别时候还是要显式调用。

1. 线程安不安全

Repository API支持对存储的多线程访问，多个并发线程可以获得到存储库的连接，并同时对其进行查询和执行操作。

Repository是线程安全的，可以在多个线程之间安全地共享和重用(实现这一点的一个好方法是通过RepositoryProvider)。

但是RepositoryConnection不是线程安全的。因此不应该尝试在多个线程上共享一个储存库连接。相反，确保同一时间只有一个线程通过RepositoryConnection连接一个Repository。但是可以使用事务来控制线程之间的操作。