(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 104573098 A
(43)申请公布日 2015.04.29

(21)申请号 201510047803.7

(22)申请日 2015.01.30

(71)申请人 深圳市华傲数据技术有限公司
地址 518057 广东省深圳市南山区高新区中区高新中一道 9 号软件大厦 7 层 713、715、716 室

(72)发明人 王明兴 吴颖徽 马帅 汤南 贾西贝

(74)专利代理机构 深圳市华优知识产权代理事务所（普通合伙）44319
代理人 余薇

(51)Int.Cl.
G06F 17/30(2006.01)

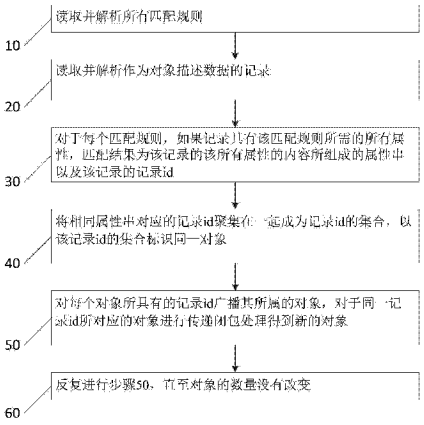
权利要求书2页 说明书9页 附图1页

(54)发明名称

基于 Spark 系统的大规模对象识别方法

(57)摘要

本发明涉及一种基于 Spark 系统的大规模对象识别方法。该方法包括：步骤 10、读取并解析所有匹配规则；步骤 20、读取并解析作为对象描述数据的记录；步骤 30、对于每个匹配规则，如果记录具有该匹配规则所需的所有属性，匹配结果为该记录的该所有属性的内容所组成的属性串以及该记录的记录 id；步骤 40、将相同属性串对应的记录 id 聚集在一起成为记录 id 的集合；步骤 50、对每个对象所具有的记录 id 广播其所属的对象，对于同一记录 id 所对应的对象进行传递闭包处理得到新的对象；步骤 60、反复进行步骤 50，直至对象的数量没有改变。本发明采用大规模并行的策略，解决了面对海量数据的匹配效率问题；通过预定义的匹配规则，规避了数据缺少与错误的问题。



CN 104573098 A

1. 一种基于 Spark 系统的大规模对象识别方法，其特征在于，包括：

步骤 10、读取并解析所有匹配规则；

步骤 20、读取并解析作为对象描述数据的记录；

步骤 30、对于每个匹配规则，如果记录具有该匹配规则所需的所有属性，匹配结果为该记录的该所有属性的内容所组成的属性串以及该记录的记录 id；

步骤 40、将相同属性串对应的记录 id 聚集在一起成为记录 id 的集合，以该记录 id 的集合标识同一对象；

步骤 50、对每个对象所具有的记录 id 广播其所属的对象，对于同一记录 id 所对应的对象进行传递闭包处理得到新的对象；

步骤 60、反复进行步骤 50，直至对象的数量没有改变。

2. 根据权利要求 1 所述的基于 Spark 系统的大规模对象识别方法，其特征在于，步骤 30 还包括：如果记录不匹配任一匹配规则，匹配结果包括特殊值和该记录的记录 id。

3. 根据权利要求 1 所述的基于 Spark 系统的大规模对象识别方法，其特征在于，步骤 50 包括：

步骤 501、对每个对象所具有的记录 id 广播其所属的对象；

步骤 502、收集每个记录 id 所属的对象，如果记录 id 所属的对象只有一个，则标记对应的对象的状态为保留；否则合并所有对象中的记录 id 并去重，生成新的对象并标记该新的对象的状态为新增，标记每个旧的对象的状态为删除；

步骤 503、合并每个对象的状态信息，如果状态内包含新增，此对象需保留；如果状态内包含删除，此对象需删除；否则，此对象需保留；

步骤 504、输出所有需要保留的对象。

4. 根据权利要求 1 所述的基于 Spark 系统的大规模对象识别方法，其特征在于，该属性串由连接符串联该所有属性的内容组成。

5. 根据权利要求 1 所述的基于 Spark 系统的大规模对象识别方法，其特征在于，步骤 10 包括：

读取匹配规则的记录文件；

获取每个规则包含的属性列。

6. 根据权利要求 1 所述的基于 Spark 系统的大规模对象识别方法，其特征在于，步骤 20 包括：

Spark 系统读取源文件；

解析源文件中的记录数据，以分割符对每行数据进行拆分。

7. 根据权利要求 1 所述的基于 Spark 系统的大规模对象识别方法，其特征在于，该匹配规则包括：

匹配规则的数据格式包括规则 id 及待比较的属性列的列表；

该匹配规则的含义为，对于任意两条记录，如果待比较的属性都不为空且相等，则称该两条记录匹配规则成功。

8. 根据权利要求 1 所述的基于 Spark 系统的大规模对象识别方法，其特征在于，对于多条匹配规则，任意两条记录满足任一条规则即称该两条记录匹配规则成功。

9. 根据权利要求 1 所述的基于 Spark 系统的大规模对象识别方法，其特征在于，如果第

一规则判定第一记录和第二记录为同一对象，第二规则判定该第二记录和第三记录为同一对象，则该第一记录、第二记录和第三记录为同一对象。

基于 Spark 系统的大规模对象识别方法

技术领域

[0001] 本发明涉及数据处理技术领域，尤其涉及一种基于 Spark 系统的大规模对象识别方法。

背景技术

[0002] 网络技术飞速发展的今天，大量网络应用和产品的使用产生了海量的数据，当我们需要对数据进行清洗、集成时，就需要识别出这些数据中哪些记录是描述同一现实对象的。举个例子：各个电商销售商品时通常会记录消费者本身的信息（姓名、性别、年龄、电话、邮箱、住址等）以及商品的信息（如商品名称、类别、单价、数量等），当需要分析消费者的消费行为时，首要的事情时根据记录中消费者的信息来识别哪些记录是隶属于同一现实消费者，而通常不同的电商记录的消费者信息内容会有所不同，或者同一现实消费者在各电商网站注册的信息有差异，部分数据会缺少甚至错误，因此不能通过简单的去重来识别同一消费者。

[0003] 对象识别又称记录匹配，其目的是从（不可靠的）各种数据源中识别出表示同一现实对象的记录。对象识别在数据清洗、数据集成、数据分析等应用中具有重要作用。在实际应用中，一个对象的信息通常需要与其他数据源的信息进行关联。然而，其他数据源中表示同一对象的信息可能存在错误或具有不同的表示形式。因此，对象识别并不简单，特别是在互联网技术的迅猛发展的今天，数据在急剧膨胀，采用传统的方法从海量数据中识别出哪些对象是相同（或相似的）几乎不可行，相关问题亟需解决。其中包含两个关键问题：一是针对数据缺少与错误的情况如何识别同一对象；二是面对海量的数据如何解决匹配效率问题，传统的策略面对海量数据时已无能为力。

[0004] 另一方面，Spark 系统是一个开源的通用并行分布式计算框架，由加州大学伯克利分校的 AMP 实验室开发，适合各种迭代算法和交互式数据分析，能够提升大数据处理的实时性和准确性，现已逐渐获得很多企业的支持。Spark 是一种与 Hadoop 相似的开源集群计算环境，但是 Spark 启用了内存分布数据集，中间输出结果可以保存在内存中，从而不再需要读写 HDFS，缩短访问延迟，除了能够提供交互式查询外，还可以优化迭代工作负载。因此 Spark 系统能更好地适用于数据挖掘与机器学习等需要迭代的 MapReduce 的算法。

发明内容

[0005] 本发明的目的在于提供一种基于 Spark 系统的大规模对象识别方法，能够提高面对海量数据的匹配效率。

[0006] 为实现上述目的，本发明提供一种基于 Spark 系统的大规模对象识别方法，包括：

[0007] 步骤 10、读取并解析所有匹配规则；

[0008] 步骤 20、读取并解析作为对象描述数据的记录；

[0009] 步骤 30、对于每个匹配规则，如果记录具有该匹配规则所需的所有属性，匹配结果为该记录的该所有属性的内容所组成的属性串以及该记录的记录 id；

[0010] 步骤 40、将相同属性串对应的记录 id 聚集在一起成为记录 id 的集合，以该记录 id 的集合标识同一对象；

[0011] 步骤 50、对每个对象所具有的记录 id 广播其所属的对象，对于同一记录 id 所对应的对象进行传递闭包处理得到新的对象；

[0012] 步骤 60、反复进行步骤 50，直至对象的数量没有改变。

[0013] 其中，步骤 30 还包括：如果记录不匹配任一匹配规则，匹配结果包括特殊值和该记录的记录 id。

[0014] 其中，步骤 50 包括：

[0015] 步骤 501、对每个对象所具有的记录 id 广播其所属的对象；

[0016] 步骤 502、收集每个记录 id 所属的对象，如果记录 id 所属的对象只有一个，则标记对应的对象的状态为保留；否则合并所有对象中的记录 id 并去重，生成新的对象并标记该新的对象的状态为新增，标记每个旧的对象的状态为删除；

[0017] 步骤 503、合并每个对象的状态信息，如果状态内包含新增，此对象需保留；如果状态内包含删除，此对象需删除；否则，此对象需保留；

[0018] 步骤 504、输出所有需要保留的对象。

[0019] 其中，该属性串由连接符串联该所有属性的内容组成。

[0020] 其中，步骤 10 包括：

[0021] 读取匹配规则的记录文件；

[0022] 获取每个规则包含的属性列。

[0023] 其中，步骤 20 包括：

[0024] Spark 系统读取源文件；

[0025] 解析源文件中的记录数据，以分割符对每行数据进行拆分。

[0026] 其中，该匹配规则包括：

[0027] 匹配规则的数据格式包括规则 id 及待比较的属性列的列表；

[0028] 该匹配规则的含义为，对于任意两条记录，如果待比较的属性都不为空且相等，则称该两条记录匹配规则成功。

[0029] 其中，对于多条匹配规则，任意两条记录满足任一条规则即称该两条记录匹配规则成功。

[0030] 其中，如果第一规则判定第一记录和第二记录为同一对象，第二规则判定该第二记录和第三记录为同一对象，则该第一记录、第二记录和第三记录为同一对象。

[0031] 综上所述，本发明通过采用大规模并行的策略，解决了面对海量数据的匹配效率问题；通过预定义的匹配规则，规避了数据缺少与错误的问题。

附图说明

[0032] 图 1 为本发明基于 Spark 系统的大规模对象识别方法一较佳实施例的流程图。

具体实施方式

[0033] 下面结合附图，通过对本发明的具体实施方式详细描述，将使本发明的技术方案及其有益效果显而易见。

[0034] 针对数据缺少与错误的情况如何识别同一对象的问题，本发明预先制定出几个关键的匹配规则，当两个消费者记录信息满足某一匹配规则时就认为他们是同一消费者，例如，本发明可设定消费者姓名与电话号码相同时就可认为是同一消费者，通过这个方法可以很好的规避数据缺少与错误的问题。为了解决面对海量的数据的匹配效率问题，本发明采用大规模并行的策略，利用多台机器并行处理，具体采用了基于内存计算 Spark 系统并行处理策略来解决这个问题，处理对象识别比 Hadoop 框架更快速。

[0035] 下面详细介绍本发明的处理细节。

[0036] ●概念定义

[0037] 不失一般性，本发明一较佳实施例使用如下通用的对象描述数据格式：

[0038]

id	姓名	性别	就职企业	…
1	王明兴	男	华傲数据	

[0039] 记录——本发明中称一行对象描述数据为一条记录，其中数据第一列“id”为记录的唯一标识，第二列以及随后的列为描述记录的属性。

[0040] 对象——本发明中称现实中相同的实体为对象。例如，同一消费者、同一某物品等。

[0041] 一个对象可能存在多条记录信息，也可能只存在一条。例如，某一消费者在不同的电商网站都有消费记录，则会存在多条记录信息；如果只在某一网站有消费，则只会有一条记录信息。

[0042] 匹配规则——本发明一较佳实施例中定义配规则如下：

[0043] 规则 id：待比较的属性列的列表。

[0044] 例如：rule1:2,3。

[0045] 该规则的含义为：任意两条记录 r1 和 r2，如果第二、第三列的属性都不为空且两条记录之间相等，则称记录 r1、r2 匹配规则成功，即记录 r1、r2 为同一对象。

[0046] 对于多条匹配规则，只要记录 r1 和 r2 满足任一条规则即称匹配规则成功。

[0047] 记录匹配的传递性——如果规则 a 判定记录 r1 和 r2 为同一对象，规则 b 判定记录 r2 和 r3 为同一对象，则有记录 r1、r2、r3 为同一对象。

[0048] ●制定匹配规则

[0049] 对象识别的准备工作为针对不同的业务数据、不同的需求制定合理的匹配规则，例如针对上面消费者的例子，本发明可预先制定如下规则（假设数据中第 2 列内容为姓名，第 3 列为电话，第 4 列为邮箱）：

[0050] rule1:2,3

[0051] rule2:2,4

[0052] rule3:3,4

[0053] 即如果两个消费者姓名和电话相同，或者姓名和邮箱相同，或者电话和邮箱相同即认为这两个消费者为同一消费者。

[0054] 下面结合图 1 所示的本发明基于 Spark 系统的大规模对象识别方法一较佳实施例的流程图及伪代码，具体举例说明本发明的详细步骤。

[0055] ●识别同一对象

[0056] 制定好匹配规则后，下一步就是利用规则来识别同一对象。本发明采用基于内存计算 Spark 系统并行处理策略来应付海量数据。

[0057] 步骤 10、读取并解析所有匹配规则。本发明首先处理匹配规则。

[0058] 先读取匹配规则文件并解析，处理过程如下：

[0059] a. 读取匹配规则的记录文件：

[0060] val ruleData = SparkContex.textFile(“ruleFileName”)

[0061] b. 解析，忽略规则 id，获取每个匹配规则包含的属性列（规则列）：

[0062] val rules = ruleData.map(_.split(":")(1).split(",").map(_.toInt)).collect()

[0063] 步骤 20、读取并解析作为对象描述数据的记录。接下来处理记录数据。不失一般性，本发明假定数据文件存储在文本文件中，一条记录存储为一行，各列属性以逗号分隔。

[0064] a.Spark 系统读取源文件：

[0065] val orgData = SparkContex.textFile(“dataFileName”)

[0066] b. 解析源文件中的记录数据，以逗号对每行数据进行拆分：

[0067] val recorders = orgData.map(_.split(","))

[0068] 通过步骤 20 输入作为对象描述数据的记录，记录的数据格式包括记录 id 及相应的属性。解析后，可得到记录 id，以及各列属性值，例如：

[0069]

1	Attr1	Attr2	Attr3	Attr4

[0070] 步骤 30、对于每个匹配规则，如果记录具有该匹配规则所需的所有属性，匹配结果为该记录的该所有属性的内容所组成的属性串以及该记录的记录 id。步骤 30 通过使用匹配规则来匹配记录数据来识别对象。首先计算出每一个规则能识别出哪些记录是代表同一对象的。

[0071] 使用匹配规则匹配数据：

[0072]

```
val matchData = recorders.flatMap(recorder => {
    // 第一个元素为记录 id
    val recId = recorder(0)
    // 用于保存所有匹配规则的匹配结果数据
    val result = new ArrayBuffer[(String, String)]
    //处理每个匹配规则
    rules.foreach(rule => {
      //判断记录属性 attrs 中 rule 所包含的属性是否为空，只要有一个为空则返回空字符串，表明此记录不适合使用规则 rule；否则将所有属性值用逗号串联起来并返回
      val ruleAttr = dealOneRule(rule, recorder)
      // 返回结果不为空，说明此记录可使用规则 rule
      if (!ruleAttr.isEmpty) {
        result +=(ruleAttr, recId)
      }
    })
    //如果结果为空，说明此记录不匹配任何一条规则，防止该记录丢失，需在结果中增加一个特殊值
    if (result.isEmpty) {
      result +=(recId, recId)
    }
    result
  })
```

[0073] 每个规则对每条记录的匹配方法如下：

[0074]

```
//判断记录 recorder 是否匹配 rule，成功则返回匹配值，否则返回空值
  def dealOneRule(rule: Array[Int], recorder: Array[String]): String = {
    val ruleAttr = ""
    rule.foreach(r => {
      if (r >= attrs.length || attrs(r).isEmpty) {
        return ""
      }
```

[0075]

```
            ruleAttr += attrs(r) + ","
        })
        ruleAttr
    }
```

[0076] 步骤 30 中，对于每个匹配规则 rule，读取规则所包含列的所有内容，如果某列内容为空，则忽略此规则；否则称该条记录匹配规则 rule。例如，对应上述的记录数据，假设此规则包含 2 个列，分别为第二列与第四列，则需判断第二列和第四列内容是否为空，如果任一列内容为空，则忽略此规则，进行下一规则判断；此处第二列和第四列内容分别为“Attr1”，“Attr3”，都不为空，输出的属性串为“Attr1，Attr3”以及记录 id “1”。

[0077] 此外，步骤 30 还可以包括：如果该记录不匹配任一规则，则需要输出特殊的内容以防止该记录丢失，例如，输出的属性串可以为记录的 id 值，通过记录 id 与规则所包含的各列属性进行区分。

[0078] 步骤 40、将相同属性串对应的记录 id 聚集在一起成为记录 id 的集合，以该记录 id 的集合标识同一对象。使用规则匹配数据后，相同属性串对应的记录为同一对象，因此需将相同属性串对应的记录 id 聚集在一起，并去重，可得到初步的同一对象结果：

[0079] var sameObject = matchData.groupByKey().map(x =>x._2.toSet)

[0080] 在步骤 40 中，记录 id 的集合即对象的形式可以是：将所有的记录 id 用逗号串联起来，使用文本的方式，一个对象保存为一行，如“1,3,4”。

[0081] 通过上述步骤，本发明能够并行计算得到每个匹配规则能识别哪些记录是代表同一对象的，如规则 1 识别出记录 1、3、4 为同一对象，规则 2 识别出 2、4 为同一对象，通过传递可知道，记录 1、2、3、4 都表示同一对象，因此需要将规则匹配的结果再处理一下，本发明称此步骤为传递闭包，执行过程参见步骤 50 和 60。因为对象之间可能存在多次传递，本发明具体采用迭代过程来解决。

[0082] 步骤50、对每个对象所具有的记录id广播其所属的对象，对于同一记录id所对应的对象进行传递闭包处理得到新的对象。

[0083] 具体可以包括：

[0084] 步骤 501、对每个对象所具有的记录 id 广播其所属的对象；

[0085] 步骤502、收集每个记录id所属的对象，如果记录id所属的对象只有一个，则标记对应的对象的状态为保留；否则合并所有对象中的记录 id 并去重，生成新的对象并标记该新的对象的状态为新增，标记每个旧的对象的状态为删除；

[0086] 步骤 503、合并每个对象的状态信息，如果状态内包含新增，此对象需保留；如果状态内包含删除，此对象需删除；否则，此对象需保留；

[0087] 步骤 504、输出所有需要保留的对象。

[0088] 步骤 50 的输入为步骤 40 的输出或上一次迭代也就是步骤 504 的输出，可以采用文本输入格式，每行内容为一个对象，也就是标识同一对象的记录 id 的集合。例如对象为“1,3,4”时将输出 3 组内容，分别为“1”/ “1,3,4”、“3”/ “1,3,4”以及“4”/ “1,3,4”。此过程的目的是广播每个记录 id 分别属于哪些对象。

[0089] 因为对象的每个记录 id 都将给该对象增加一个状态信息，且状态信息可能不一

致，如对于对象“1，3，4”，“1”只属于此对象，因此它将给该对象增加状态“保留”，而“4”属于多个对象，表明“1，3，4”需与其他对象合并后删除，保留那个新增的对象，因此它将给该对象增加状态“删除”。故此需要合并对象的所有状态信息，并确定对象的最终状态。例如：第一步可能得到的结果为“1，2”，“2，3”，“3，4”，经分析可得记录“1，2，3，4”都表示同一个对象，而经过一轮传递闭包计算后得到“1，2，3”和“2，3，4”，需再做一次传递闭包才得最终的结果“1，2，3，4”。也就是执行步骤 60，反复进行步骤 50，直至对象的数量没有改变。

[0090] 步骤 50 和 60 具体如下：

[0091]

```
//num2 记录当前对象的数量
var num2 = sameObject.count()
//上一步对象的数量，初始值为当前数量加 1
var num1 = num2 + 1
//迭代条件，当对象数没有更新时终止
while (num1 > num2) {
  //记录广播，告之每个记录所属的对象
  val recBroadcast = sameObject.flatMap(obj => obj.map(o => (o, obj)))
  //聚集，收集每个记录及其所属的对象，并合并
  val groups = recBroadcast.groupByKey().flatMap(x => {
    val obs = x._2
    val res = new ArrayBuffer[(Set[String], Int)]()
    if (obs.size == 1) {
```

[0092]

```
            //如果所属的对象数为 1，说明没有传递，原对象保留,增加状态值 0
              res +=(obs(0), 0)
            } else {
              //否则，将所有对象内的记录合并、去重后得到新的对象，并将原
对象删除,状态值为 1
              val newObs = obs.reduce(_ ++ _)
              //旧对象标记状态为 1（删除）
              obs.foreach(x => res +=(x, 1))
              //新对象标记状态为 2（新增）
              res +=(newObs, 0)
            }
            res
          })
          //合并每个对象的所有状态并处理
          sameObject = groups.groupByKey().map(x => (x._1,
newState(x._2))).filter(_._2 == 0)
          //更新对象数量
          num1 = num2
          num2 = sameObject.count()
        }
```

其中多状态处理方法如下：

```
      //合并对象的各个状态
        def newState(stats: Iterable[Int]): Int = {
          //若状态内包含新增状态，则返回 1，表明此对象需保留
          if (stats.exists(i == 2)) {
            return 1
          }
          //否则若状态内包含删除状态，则返回 0，表明此对象需删除
          if (stats.exists(i == 1)) {
            return 0
          }
          //否则此对象需要保留
          1
```

[0093]

```
        }
```

[0094] 至此，本发明基于 Spark 系统的大规模对象识别方法执行完成。

[0095] 综上所述，本发明基于 Spark 系统的大规模对象识别方法采用大规模并行的策略，解决了面对海量数据的匹配效率问题；通过预定义的匹配规则，规避了数据缺少与错误的问题；众所周知，数据的价值是 1+1>>2 的，本发明将原本孤立但却高度相关的数据联系起来，其价值要远大于本身价值之和。

[0096] 以上所述仅为本发明的较佳实施例，并不用以限制本发明，凡在本发明的精神和原则之内所作的任何修改、等同替换和改进等，均应包含在本发明的保护范围之内。

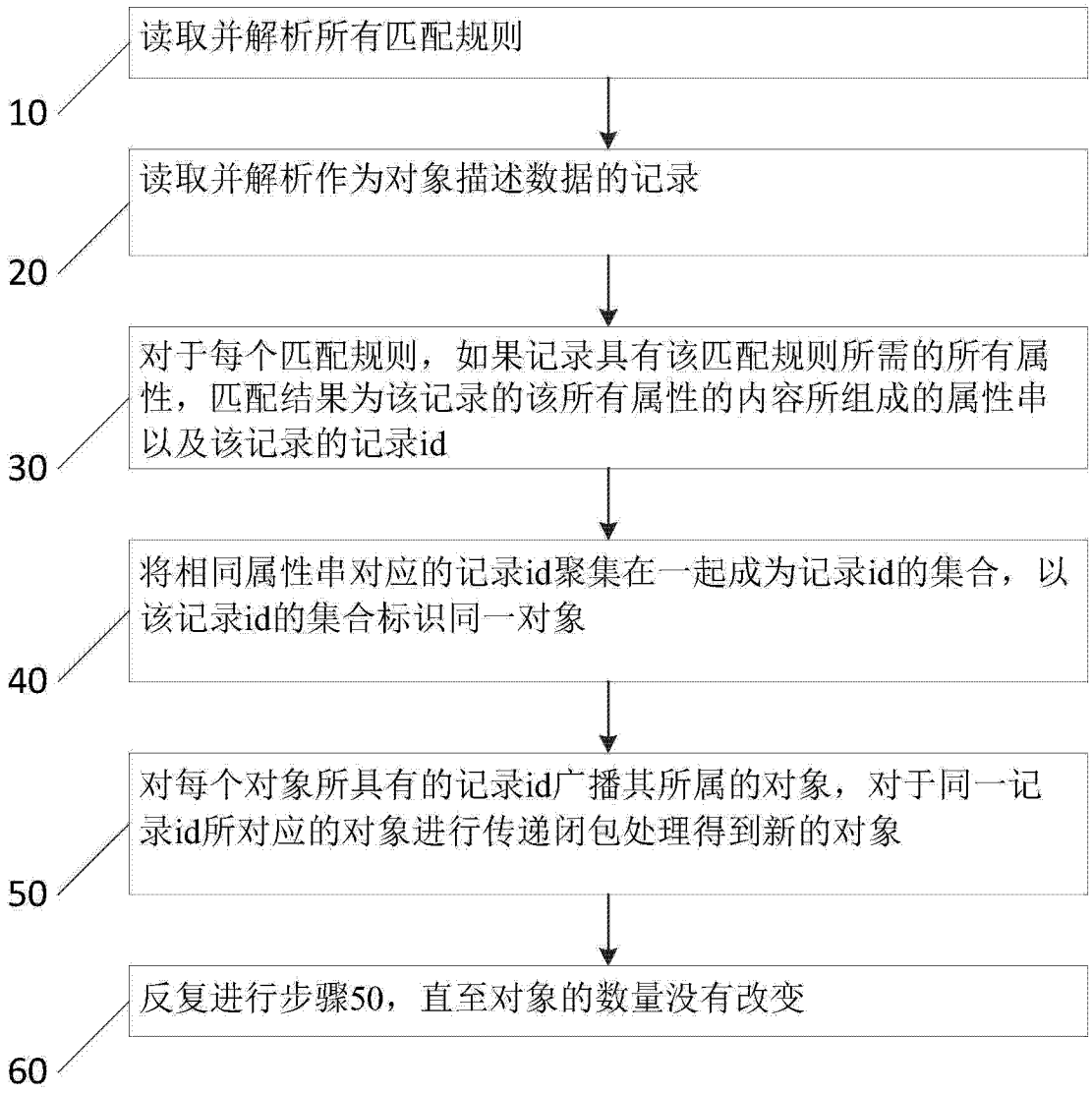

读取并解析所有匹配规则
10
读取并解析作为对象描述数据的记录
20
对于每个匹配规则，如果记录具有该匹配规则所需的所有属性，匹配结果为该记录的该所有属性的内容所组成的属性串以及该记录的记录id
30
将相同属性串对应的记录id聚集在一起成为记录id的集合，以该记录id的集合标识同一对象
40
对每个对象所具有的记录id广播其所属的对象，对于同一记录id所对应的对象进行传递闭包处理得到新的对象
50
反复进行步骤50，直至对象的数量没有改变
60

图 1