计算机科学
COMPUTER SCIENCE

基于数据库表的微服务拆分方法

黄志成, 柳先辉

引用本文

黄志成, 柳先辉. 基于数据库表的微服务拆分方法[J]. 计算机科学, 2023, 50(11A): 230200102-7.

HUANG Zhicheng, LIU Xianhui. Microservice Splitting Approach Based on Database Table[J]. Computer Science, 2023, 50(11A): 230200102-7.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230200102

基于数据库表的微服务拆分方法

黄志成　柳先辉
同济大学电子与信息工程学院　上海 201800
(540955198@qq.com)

摘　要　微服务架构和云平台容器化部署是当前软件工程实践中一个比较热门的话题,很多研究报告表明越来越多的软件开发者正在将单体架构向微服务架构转型。在将单体架构应用拆分成微服务架构应用的过程中,实施者面临着一个重要的挑战,即缺乏一个明确的方法将单体应用高效准确的进行拆分。针对这个问题,提出了一种基于数据库表的微服务拆分方法并实现了一个拆分工具。该方法通过收集项目中的所有SQL语句,并结合数据库表之间的主外键关系,生成表关联矩阵,根据这个表关联矩阵初步划分出一部分微服务。然后根据测试案例收集所有的交易链路,再结合交易链路分析表和微服务之间的关系,计算出独立的表和微服务的关联度矩阵,根据这个独立的表和微服务的关联度矩阵来完成最终的微服务数据库表的划分,最后按照提出的规则进行微服务代码的拆分。实验结果表明,所提方法可以帮助软件开发者高效准确地进行微服务拆分。

关键词:微服务;拆分;数据库表
中图法分类号　TP311

Microservice Splitting Approach Based on Database Table

HUANG Zhicheng and LIU Xianhui
School of Electronics and Information Engineering,Tongji University,Shanghai 201800,China

Abstract　Microservice architecture and cloud platform container deployment are a hot topic in current software engineering practice. Many research reports show that more and more software developers are transforming single architecture to microservice architecture. In the process of splitting a single architecture application into a microservice architecture application,the implementer faces an important challenge,that is,the lack of a clear method to effectively and accurately split the single application. To solve this problem,a micro-service splitting method based on database tables is proposed and a splitting tool is implemented. This method generates a table association matrix by collecting all the SQL statements in the project and combining the primary and foreign key relationships between the database tables. According to this table association matrix,a part of microservices is initially divided. Then collect all the transaction links according to the test cases,and combine the relationship between the transaction link analysis table and the micro-service to calculate the association matrix of the independent table and the micro-service. According to the association matrix of the independent table and the micro-service,complete the division of the final micro-service database table,and finally split the micro-service code according to the proposed rules. The experiment shows that this method can help software developers effectively and accurately split microservices.

Keywords　Microservices,Splitting,Database table

1　引言

微服务架构与单体架构相比有着一些优势,但是微服务的划分依然是一个棘手的问题。在将单体应用拆分成微服务应用的过程中,会涉及一系列复杂的决策,而通过手工的方式进行微服务的划分比较依赖开发人员的技术和经验,但是人工完成这些操作可能存在一定的主观性误差,效率不高。在现有的研究中,半自动化和自动化的微服务划分方法,在一定程度上缓解了手工划分方法的缺陷,提高了效率。但是微服务划分不仅仅是将单体应用的代码拆分为多个微服务的代码,还包括将一个数据库拆分为多个库,即每个微服务均使用属于自己的数据库。当某一个微服务需要访问属于它自己的数据库时,它可以直接进行访问,当想要获取其他微服务数据库中的数据时,通过远程调用其他微服务提供的接口来获取这些数据。现有的研究很少关注到这一方面。如何利用表的拆分信息辅助微服务的拆分是一个巨大的挑战。本文将研究一种基于数据库表的微服务拆分方法,用于辅助微服务的拆分。

在微服务拆分领域,前人已经进行不少探索。Ahmadvand等[1]提出了一种基于软件的安全性与可拓展性需求进行微服务化拆分的方法。该方法强调,在进行微服务划分时应该考虑系统的安全性和可扩展性等因素,但该方法只考虑

基金项目:国家重点研发计划项目(2022YFB330570);上海市科技创新行动计划项目(21511104302)
This work was supported by the National Key Research and Development Program of China(2022YFB330570) and Science and Technology Innovation Action Plan Program of Shanghai(21511104302).
通信作者:柳先辉(xianhui_l@163.com)

了非功能性需求，而没有考虑到系统的功能性需求。Gysel等[2]提出了一种名为Service Cutter的基于耦合标准的微服务划分方法。该方法通过提取耦合信息生成无向加权图，并应用聚类算法进行微服务划分。然而，这种方法需要人为定义聚类的标准，无法从系统本身提取必要的信息。Levcovitz等[3]通过人工的方式将数据库表分类，并根据静态代码依赖图进行微服务划分。然而，这种方法在考虑微服务特性、系统内部之间的关联以及业务场景等方面并不够充分，导致微服务划分出现了一定的不合理性。Baresi等[4]提出了一种基于OpenAPI规范的计算语义相似度的方法，利用参考词汇表对潜在的候选微服务进行内聚操作。然而，这种方法仅适用于符合OpenAPI规范的系统，并且参考词汇的判断依赖于人的主观判断。Chen等[5]提出了一种基于数据流驱动的微服务划分算法，该算法包括3个步骤。首先进行需求分析，构建一个详细的数据流图，然后将相同类型输出数据的相同操作组合成一个虚拟抽象数据流，最后从虚拟抽象数据流中提取“操作及其输出数据”的各个模块，用于表示已识别的微服务候选者。Mazlamid等[6]提出了一种基于微服务提取模型的划分方法。该方法通过逻辑耦合、语义耦合、参与者耦合策略生成无向图，最后利用图聚类算法进行微服务划分。然而，这种方法仅对类进行划分，并没有考虑数据库的影响因素。Eski等[7]提出了一种使用代码仓库将现有应用程序转换为微服务的方法。该方法利用动态与静态代码耦合信息和图聚类算法自动提取微服务。然而，这种方法未考虑数据库方面的影响，并且可能会使微服务拆分的粒度过细。Kamimura等[8]提出了一种基于源代码中微服务划分的方法。该方法定义了“程序组”与“数据”的关系，并使用软件聚类算法进行微服务划分。然而，这种方法严重依赖于源代码中注解的使用。Rademacher等[9]提出了一种模型驱动的开发方法。它通过引入中间模型，来明确微服务接口和部署细节。但是，这种方法需要结合大量的专家经验，而且在考虑微服务特性方面不够充分。Ding等[10]使用场景驱动的方式，通过动态分析获取单体遗留系统运行时的方法调用和数据库操作信息，并基于数据表之间的关联关系分析生成数据库拆分方案。Li等[11]提出了一种优化的微服务化拆分方法。该方法通过动态分析和静态分析相结合的方式，实现了更加高效的数据流信息自动化收集，并采用两阶段聚类算法来取代完全基于自定义规则的微服务化拆分算法。然而，该方法未考虑系统之间SQL语句的关联，仅从场景分析无法充分利用系统中表的内在关系。

国内外的研究表明，微服务的划分可以基于系统的业务能力、业务场景的需求和业务模型的需要，但是每种方法都有其独特的使用场景和局限性。例如，需要大量专家经验来判断，并且无法获得全面的系统信息。现有的方法大多都未实现微服务架构提倡的Database per Service原则，数据库的拆分是微服务拆分过程中的重要环节，因此探索一种实用高效的数据库拆分方法具有重要意义。此外，在微服务拆分过程中，将数据库表、SQL语句与业务场景之间进行交互缺乏相应的研究。

2 基于数据库表的微服务拆分方法

本文提出的基于数据库表的微服务拆分方法一共分为6步，具体流程如图1所示。

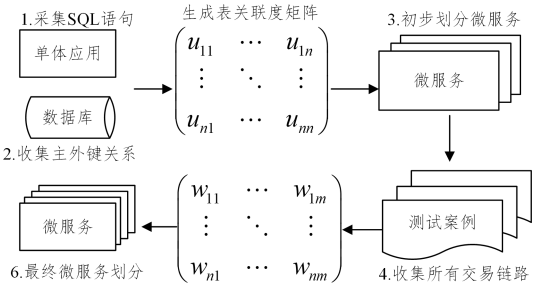

图1 基于数据库表的微服务拆分方法的流程图

Fig. 1 Flow chart of microservice splitting method based on database table

2.1 系统中SQL语句的采集

采集系统中的SQL语句，并对所有SQL语句进行分析。定义如式(1)所示的数据库表关联矩阵，U_{ij}代表第i张数据库表与第j张数据库表的关联情况，如果两张数据库表有关联，则$U_{ij}=U_{ji}=1$，否则$U_{ij}=U_{ji}=0$。初始状态下，关联矩阵的所有元素均为0。

$$U=\begin{pmatrix} u_{11} & \cdots & u_{1n} \\ \vdots & \ddots & \vdots \\ u_{n1} & \cdots & u_{nn} \end{pmatrix} \tag{1}$$

按规则1和规则2更新数据库表关联矩阵。

规则1 若两张数据库表在同一SQL语句中涉及关联查询，则将这两张数据库表标记为有关联。

规则2 若多张数据库表在同一SQL语句中涉及关联查询，则将这些数据库表互相标记为有关联。

2.2 数据库表之间的主外键关系收集

收集数据库表之间的主外键关系，按规则3更新式(1)中所定义的表关联矩阵U。

规则3 若数据库表i的某个字段是数据库表j的物理外键或逻辑外键，则将这两张数据库表标记为有关联。

2.3 微服务初步划分

在前两步完成之后，我们将获得一个表关联矩阵$\boldsymbol{U}$。接下来，根据该表关联矩阵进行微服务的初步划分。本文采用并查集相关算法对数据库表进行合并，进而初步划分微服务，算法流程如算法1所示。其中，$find(i)$函数的作用为获取第i张数据库表所属集合的根结点编号，$union(i,j)$函数的作用为将第i张数据库表与第j张数据库表合并。

算法1 并查集算法合并相关联数据库表

输入：数据库表关联矩阵 **U**

输出：微服务数据库表集合A，独立的表集合B

```
1. n=len(U),A=List<List<Integer>>,B=List<Integer> /* 初始定义矩阵 W 的维度，集合 A,B */
2. for ( i=0;i < n;i++) do
3.     for ( j=i+1;j < n;j++) do
4.         if (U[i][j]== 1) then/* 如果第 i 张表与第 j 张表相关联 */
5.             union(i,j) /* 将第 i 张表与第 j 张表合并 */
6.         end if
7.     end for
8. end for
9. for ( i=0;i < n;i++) do
10.    for ( j=i+1;j < n;j++) do
```

```
11.      if (find(i) == find(j)) then /* 第 i 张表与第 j 张表在同一组 */
12.      A[find(i)].add(i)
13.        A[find(i)].add(j) /* 将第 i 张表与第 j 张表加到同一个数据库表集合中 */
14.      end if
15.    end for
16. end for
17. for ( i=0;i< n;i++) do
18.    if (i not in A) then /* 如果第 i 张表不存在与集合 A 中 */
19.      B.add(i) /* 将第 i 张表加入集合 B */
20.    end if
21. end for
22. return A,B
```

算法 1 将所有相关联的数据库表进行合并，依据数据表划分微服务，形成微服务相关的数据库表集合与独立数据库表集合。其中，微服务相关的数据库表集合中包含若干子集，每一个子集属于一个微服务，该子集内包含属于这个微服务的所有数据库表；另外还剩一些不与任何表相关联的数据库表，称为独立的表。此时，完成微服务的初步划分，如图 2 所示。

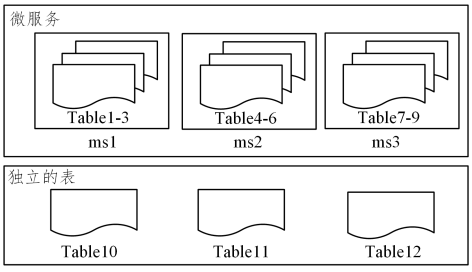

图 2　微服务初步划分示意图

Fig. 2　Preliminary division diagram of microservices

2.4　交易链路构建

根据项目中的全功能测试案例，构建所有的交易链路。如图 3 所示，交易链路指在一个测试案例中，用户发起请求后，经由系统内部的一系列类进行处理，每个类中调用相应方法，并执行 SQL 语句以访问数据库中的表格，这些元素的整合形成了一个交易路径。

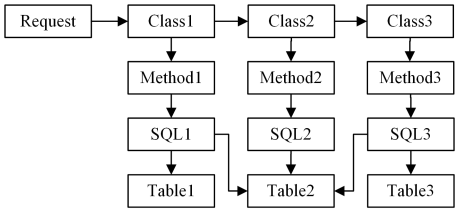

图 3　交易链路示意图

Fig. 3　Transaction link diagram

遍历项目中所有的测试案例，每一个测试案例都可以得到一个交易链路，遍历完成之后得到一个交易链路集合。

2.5　独立的表与微服务关联度计算

在第三步完成之后，我们将获得若干微服务及其对应的数据库表，以及一些独立的表。第五步的目的是根据第四步生成的交易链路集合，计算独立的表与微服务之间的关联度，以期望将独立表划分到最适合的微服务中。

定义一个独立的表与微服务关联度矩阵，如式(2)所示，其中W_{ij}代表第 i 个独立的表与第 j 个微服务之间的关联度。初始时关联度矩阵所有值均为 0，横坐标表示所有独立的表，纵坐标表示所有的微服务。

$$W=\begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix} \tag{2}$$

按规则 4 更新式(2)定义的关联度矩阵 $\boldsymbol{W}$。

规则 4　如果在某个交易链路中同时涉及到第 i 个独立的表和第 j 个微服务，那么将第 i 个独立的表与第 j 个微服务的关联度加 1。

规则 4 可以通过算法 2 来实现，在执行完所有的交易链路之后，可以统计出每一个独立的表与所有微服务的关联度。

算法 2　计算独立的表与微服务的关联度

输入：交易链路图集合 G，微服务数据库表集合 A，独立的表集合 B

输出：独立的表与微服务关联度矩阵 **W**

```
1. n=len(G),W[B.length][A.length] /* 初始定义交易链路大小，关联度矩阵 */
2. for ( i=0;i < n;i++) do
3.    for ( j=0;i < A.length;j++) do
4.       for ( k=0;k < B.length;k++) do
5.          if (A[j]和 B[k]存在于集合 G(i)中)then /* 如果第 i 个交易链路同时涉及了第 k 个独立的表和第 j 个微服务 */
6.             W[k][j]++ /* 将第 k 个独立的表与第 j 个微服务的关联度加 1 */
7.          end if
8.       end for
9.       end for
10.    end for
11. return A
```

2.6　微服务最终划分

根据独立的表与微服务的关联度矩阵完成最终微服务数据库表的划分，划分规则如规则 5 所示。

规则 5　根据独立的表与微服务的关联度矩阵 $\boldsymbol{W}$，将每个独立的表添加到与之关联度最大的微服务数据库表集合中。

划分的流程如图 4 所示，将每个独立的表添加到与之关联度最大的微服务中。在本图中，假设独立的表 table10－table12 都与第 3 个微服务关联度最大。

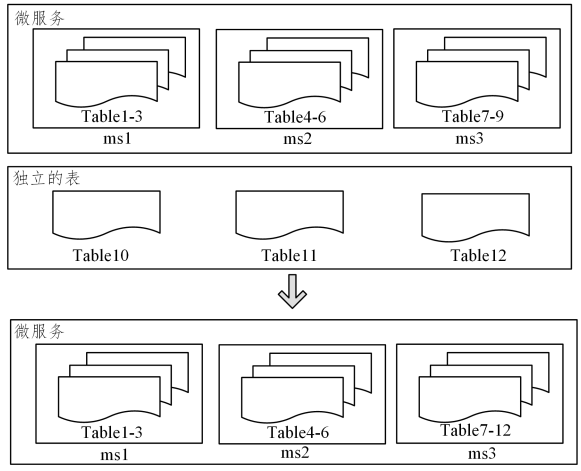

图 4　最终微服务数据库表划分图

Fig. 4　Final micro-service database table partition diagram

数据库表整体划分完成之后，原先的单体应用中的数据库表会被拆分到多个微服务中。完成微服务数据库表的拆分

后，对于项目代码的修改，按以下规则完成。

规则 6 为每个微服务新建一个项目并新建一个数据库，将属于该微服务的表结构和表数据迁移到新库中。

规则 7 为每个微服务新编写 API 接口，以便将其库中的数据提供给其他微服务使用。

规则 8 将原先单体应用中的所有业务代码根据功能迁移至最适合的微服务项目中。迁移后，在代码中访问属于自己的数据库时采用直接访问方式；而在访问其他微服务中的数据库表数据时，改为调用其他微服务提供的 API 接口。

经过以上 6 个步骤，即可完成微服务数据库表与代码的划分。

3 模拟拆分

本文先通过对 Jpetstore[1)] 应用进行模拟拆分来详细介绍本文拆分算法的具体流程。Jpetstore 是基于 Spring 和 Mybatis 框架实现的小型宠物商店系统。该应用主要包含以下功能：登录、注册、查询账户信息、修改账户信息、浏览商品信息、加入购物车、移除购物车、更新购物车、下单、查看订单等。

此应用的数据库表如表 1 所列。在该应用中，购物车数据存储于内存中。然而，在实际项目中，购物车数据通常存储在数据库中。因此，在本案例中，我们将购物车数据也视为一张表。

表 1 Jpetstore 数据库

Table 1 Jpetstore database

编号	表名	编号	表名
T1	Account	T8	Category
T2	Profile	T9	Supplier
T3	Bannerdata	T10	Orders
T4	Signon	T11	Orderstatus
T5	Product	T12	Lineitem
T6	Item	T13	Sequence
T7	Inventory	T14	Cart

第 1 步 在应用中采集所有 SQL 语句，并筛选出包含关联查询的 SQL 语句。具体筛选结果如表 2 所列。

表 2 涉及关联查询的 SQL 语句

Table 2 SQL statement table involving associated query

SQL ID	涉及的表编号
getItemListByProduct	T5，T6
getAccountByUsername	T1，T2，T4
getAccountByUsernameAndPassword	T1，T2，T3，T4
getItem	T5，T6，T7
getOrder	T10，T11

根据表 2 的结果，可以将 T5 和 T6 标记为有关联关系；T1，T2，T4 之间标记为有关联关系；T1，T2，T3，T4 之间标记为有关联关系；T5，T6，T7 之间标记为有关联关系；将 T10 和 T11 标记为有关联关系。

第 2 步 收集数据库表之间的主外键关系。若某表中的某字段是另一张表的外键，则将这两张表标记为具有关联。由于在建表过程中应用了外键约束，因此会将所有涉及外键约束的两张表标记为具有关联。例如，T5 商品表和 T8 商品类别表具有关联，T6 商品项表和 T9 供应商表具有关联。

完成前两步后，可得到如图 5 所示的数据库表关联矩阵，其中矩阵的横纵坐标均按照表 1 中 T1－T14 的顺序表示每个表。

$$\boldsymbol{U}=\begin{pmatrix}
0&1&1&1&0&0&0&0&0&0&0&0&0&0\\
1&0&1&1&0&0&0&0&0&0&0&0&0&0\\
1&1&0&1&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&0&0&0&0&0&0\\
0&0&0&0&1&0&1&0&1&0&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

图 5 数据库表关联矩阵图

Fig. 5 Database table association matrix

第 3 步 使用算法 1 对所有具有关联关系的数据库表进行合并。最终形成 3 个微服务和 3 个独立表，分别为用户服务、订单服务和商品服务。初步划分微服务的结果如表 3 所列。

表 3 微服务初步划分结果

Table 3 Preliminary division results of microservices

划分结果	涉及的表
微服务 1(用户服务)	Signon，Account，Profile，Bannerdata
微服务 2(订单服务)	Orders，Orderstatus
微服务 3(商品服务)	Product，Item，Inventory，Category，Supplier
独立的表	Lineitem，Sequence，Cart

第 4 步 根据所有测试案例，构建所有的交易链路。本应用的所有交易链路介绍如表 4 所列。表 4 中，使用代码中的 SQL 语句的 ID 来替代 SQL 语句，使用表 1 中的编号来替代数据库表。

表 4 Jpetstore 交易链路

Table 4 Jpetstore transaction link

编号	说明	涉及的 SQL 语句	涉及的数据库表编号
U1	注册	insertAccount，insertProfile，insertSignon，getAccountByUsername，getProductListByCategory	T1，T2，T3，T4
U2	修改账户信息	updateAccount，updateProfile，getAccountByUsername，getProductListByCategory	T1，T2，T3，T4，T5
U3	登录	getAccountByUsernameAndPassword，getProductListByCategory	T1，T2，T3，T4，T5
U4	退出登录	无	无
U5	添加商品项到购物车	incrementQuantity，getInventoryQuantity，getItem，addItem	T5，T6，T7，T14
U6	从购物车中移除商品项	removeItemById	T14
U7	更新购物车库存	getAllCartItems，setQuantityByItemId	T14
U8	浏览商品类别	getProductListByCategory，getCategory	T5，T8
U9	浏览商品	getItemListByProduct，getProduct	T5，T6

1) https://github.com/mybatis/jpetstore-6

（续表）

编号	说明	涉及的 SQL 语句	涉及的数据库表编号
U10	浏览商品项	getItem	T5，T6，T7
U11	搜索商品	searchProductList	T5
U12	查询所有订单	getOrdersByUsername	T10，T11
U13	下单	getSequence，updateSequence，updateInventoryQuantity，insertOrder，insertOrderStatus，insertLineItem	T7，T10，T11，T12，T13，T14
U14	查看订单	getOrder，getLineItemsByOrderId，getItem，getInventoryQuantity	T5，T6，T7，T10，T11，T12

第 5 步　使用算法 2 计算独立的表与微服务之间的关联度，之后生成如图 6 所示的关联度矩阵，其中横坐标从上到下分别代表 Lineitem，Sequence，Cart 3 张表，纵坐标从左至右分别代表微服务 1（用户服务）、微服务 2（订单服务）、微服务 3（商品服务）。

$$\boldsymbol{W}=\begin{pmatrix}0&0&0\\1&1&1\\0&0&0\end{pmatrix}$$

图 6　独立的表与微服务之间的关联度矩阵图

Fig. 6　Correlation matrix between independent tables and microservices

第 6 步　根据独立表与微服务之间的关联度矩阵图，可以发现剩余的 3 张表与订单微服务关联性最大。因此，将这 3 张表都归划在订单微服务中。最终的微服务划分方案如表 5 所列。

表 5　最终微服务划分情况

Table 5　Final micro-service division

划分结果	涉及的表
微服务 1（用户服务）	Signon，Account，Profile，Bannerdata
微服务 2（订单服务）	Orders，Orderstatus，Lineitem，Sequence，Cart
微服务 3（商品服务）	Product，Item，Inventory，Category，Supplier

4　实验设计

本文采用 Java 语言实现了一个微服务拆分工具，其核心思路与部分代码已在第 2 节详细介绍，工具的具体流程已在第 3 节详细介绍。拆分工具所需使用的数据包括涉及关联查询数据库表、涉及主外键关系数据库表、所有交易链路等，通过预先整理以配置文件的形式给出。为验证本文拆分算法的有效性和效率性，我们将其与其他工具进行了对比实验。

4.1　实验应用选择

本文选择两个 Java 应用进行评估，分别是 Jpetstore 和 Plants[1)]。选择它们的原因是，这些应用是开源的并且功能完善的单体应用，且在微服务拆分领域已被多次使用。

4.2　Baseline 选择

本文实验结果与以下 5 个 Baseline 进行比较：FoSCI[12]，CoGCN[13]，Bunch[14]，MEM[15]，M2M[16]。

1）FoSCI，使用分层聚类方法创建功能原子，然后使用遗传算法合并原子以计算分区建议。

2）CoGCN，提出了一种通过最小化可能存在于其他类的嵌入中的异常类的影响，来划分单片应用程序的方法。

3）Bunch，使用一个外部模块依赖图作为其输入来生成分区，使用了 Saeidi 等提出的最近爬山算法（NAHC）[17]。

4）MEM，使用 Kruskal 算法计算最小生成树，考虑它们的逻辑和语义耦合策略来生成分区。

5）M2M，执行时空分解，利用定义良好的业务用例和运行时调用关系来创建应用程序类的功能内聚分区。

4.3　评价指标

使用以下 5 个指标来评价实验的有效性。

1）SMQ[12]，是从结构角度衡量模块化质量的指标，scoh 衡量服务的结构内聚性，scop 衡量服务间耦合度。它们的计算式如式（3）－式（5）所示。

$$SMQ=\frac{1}{N}\sum_{i=1}^{N}scoh_i-\frac{2}{N(N-1)}\sum_{i\neq j}^{N}scop_{i,j} \tag{3}$$

$$scoh_i=\frac{\mu_i}{N_i^2} \tag{4}$$

$$scop_{i,j}=\frac{\sigma_{i,j}}{2*(N_i*N_j)} \tag{5}$$

其中，μ_i 表示微服务 i 内部边的数量，$\sigma_{i,j}$ 表示微服务 i 与微服务 j 之间边的数量，N_i 或 N_j 代表服务 i 或 j 内实体的数量。如果两个实体之间存在调用依赖关系，则存在一条边。SMQ 值越高，服务的模块化程度就越好。

2）ICP[16]，衡量了在两个微服务 i 和 j 之间发生的运行时调用的百分比，计算式如式（6）所示。

$$icp_{i,j}=\frac{c_{i,j}}{\sum_{i=1,j=1,i\neq j}^{M}c_{i,j}} \tag{6}$$

其中，$c_{i,j}$ 代表微服务 i 与微服务 j 之间的调用次数。ICP 值越低越好。

3）BCP[16]，衡量每个分块的业务用例的平均熵。分块的用例由与其成员类关联的所有用例标签组成。如果分块实现了一小组用例，则认为其功能是紧密的。BCP 的计算式如式（7）所示。

$$BCP=\frac{1}{M}\sum_{i=1}^{M}bcp_i \tag{7}$$

其中，bcp_i 的计算式如式（8）所示。

$$bcp_i=-\sum_{j=1}^{m_i}\frac{1}{|m_i|}\log\left(\frac{1}{|m_i|}\right) \tag{8}$$

其中，$1/|m_i|$ 是大小为 m_i 的向量，m_i 表示微服务 i 的用例数量。例如，给定一个有 3 个用例的微服务，$1/|m_i|$ 表示为[1/3，1/3，1/3]。BCP 值越低越好。

4）IFN[12]，衡量微服务中的接口数量。IFN 的计算式如式（9）所示。

$$IFN=\frac{1}{N}\sum_{i=1}^{N}ifn_i \tag{9}$$

其中，ifn_i 为微服务中的接口数量，N 为微服务的总数。IFN 值越低越好。

5）NED[13]，衡量微服务的大小的均衡性。NED 的计算式如式（10）所示。

$$NED=1-\frac{\sum_{i=1,knotextreme}^{K}n_k}{|N|} \tag{10}$$

1) https://github.com/WASdev/sample.mono-to-ms.pbw-monolith

其中，k 在{5,20} 范围内[18]。NED 值越低越好。NED 最初是由 Wu 等[19]提出的，用于评估微服务分布的极端性。

5 实验结果分析

5.1 有效性

为了评估本文方法的有效性，本文对 Jpetstore 和 Plants 案例使用拆分工具进行划分，并与 M2M，FoSCI，CoGCN，Bunch 和 MEM 在 5 个指标上进行对比。实验结果如表 6、表 7 所列。

表 6 Jpetstore 案例中本文方法与 M2M，FoSCI，CoGCN，Bunch，MEM 的实验结果

Table 6 Experimental results of the proposed method, M2M, FoSCI, CoGCN, Bunch and MEM in Jpetstore

Jpetstore	本文方法	M2M	FoSCI	CoGCN	Bunch	MEM
BCP	1.436	1.625	2.181	1.905	2.433	2.496
ICP	0.392	0.333	0.478	0.582	0.477	0.434
SM	0.061	0.054	0.044	0.091	—	0.124
IFN	2.234	1.857	3.750	2.533	7.948	3.429
NED	0.356	0.257	0.516	0.392	0.667	1.000

表 7 Plants 案例中本文方法与 M2M，FoSCI，CoGCN，Bunch，MEM 的实验结果

Table 7 Experimental results of the proposed method, M2M, FoSCI, CoGCN, Bunch and MEM in Plants

Plants	本文方法	M2M	FoSCI	CoGCN	Bunch	MEM
BCP	1.834	1.690	2.593	2.338	2.902	1.902
ICP	0.279	0.381	0.682	0.571	0.501	0.320
SM	0.833	0.078	0.135	0.133	0.155	0.210
IFN	3.452	6.000	4.875	4.875	6.357	4.750
NED	0.246	0.038	0.538	0.500	0.346	0.231

从表 6、表 7 中可以看出，在业务链路紧密度(BCP)方面，本文方法获得了相对较好的成果，在 Jpetstore 案例中表现最佳，而在 Plants 案例中表现为第二佳。这是因为本研究在划分过程中考虑了 SQL 语句及业务链路，从而使得微服务业务用例划分更为紧密。在内部调用比例(ICP)方面，本文方法在 Jpetstore 案例中表现为第二佳，而在 Plants 案例中表现最佳，这表明提出的拆分方法在微服务拆分过程中相对较好，各微服务之间的调用次数较少。在模块度(SM)方面，MEM 和 M2M 分别在 Jpetstore 案例和 Plants 案例中获得了最佳表现，而本文方法的表现处于中等水平。这意味着在模块度方面，本文方法的性能适中，既不是最优也不是最差。在接口分离度(IFN)方面，本文方法在 Jpetstore 案例中获得了第二佳的结果，在 Plants 案例中获得了最佳的结果。这表明本文方法在代码接口划分方面更为有效，有助于提高微服务之间的独立性。在节点分布均匀度(NED)方面，M2M 方法明显优于其他方法，表明 M2M 生成的大多数微服务包含 5～20 个类。这是由于 M2M 采用了基于层次聚类的方法，而其他算法则使用了多目标优化和 K 均值算法。本文方法在这一方面的表现也处于中等水平。

5.2 效率性

表 8 列出了每种方法划分为服务所需的时间。本文比较了各种方法，发现本文方法划分微服务的时间明显短于 FoSCI，CoGCN 和 MEM，与 M2M 的时间差不多。Bunch 使用爬山方法花费的时间最短。此外，发现使用遗传算法的 FoSCI 花费的时间最长，其次是使用神经网络方法的 CoGCN，以及使用最小生成树方法的 MEM。这主要归因于本文算法在划分过程中没有涉及复杂的分类算法，而且大部分计算工作在数据预处理阶段已经完成，从而使得所提方法在微服务划分的执行效率上具有优势。

表 8 各方法划分微服务的时间

Table 8 Microservice time divides by each method

(单位：s)

应用	本文方法	M2M	FoSCI	CoGCN	Bunch	MEM
Jpetstore	1.24	0.10	210.9	14.41	0.03	1.60
Plants	0.89	0.05	87.24	11.01	0.04	3.50

结束语 本文提出了一种基于数据库表的微服务拆分方法，并使用 Java 语言实现了一个拆分工具，该方法通过采集项目中的所有 SQL 语句，结合数据库表之间的主外键关系进行初步分析之后划分一部分微服务，然后结合交易链路分析表与微服务之间的关联关系，最后通过计算独立表与微服务的关联度完成最后微服务数据库表的拆分。还通过一个小型的商城项目 Jpetstore 对这种方法进行模拟拆分，以对详细流程进行介绍。最后与主流的拆分工具进行对比实验，对算法的有效性与效率性进行了验证。

本文后续的研究方向如下：

1)微服务拆分工具的优化。本文的拆分工具在运行过程中所依赖的数据(涉及关联查询表、涉及主外键关系表、交易链路等)在现阶段主要靠人工整理以配置文件的形式交给程序。后续考虑引入动态分析的技术对这些数据进行高效的收集。

2)企业级应用的评估。本文算法在中小型案例中的拆分效果不错，但是在企业级应用中，通常数据库表的规模都较大，还需要继续评估此算法在企业级应用中的表现。

参 考 文 献

[1] AHMADVAND M, IBRAHIM A. Requirements reconciliation for scalable and secure microservice (de) composition[C]// Proceedings of the 2016 IEEE 24th International Requirements Engineering Conference Workshops(REW). 2016.

[2] GYSEL M, KÖLBENER L, GIERSCHE W, et al. Service cutter: A systematic approach to service decomposition[C]// Proceedings of the European Conference on Service-Oriented and Cloud Computing. 2016.

[3] LEVCOVITZ A, TERRA R, VALENTE M T. Towards a technique for extracting microservices from monolithic enterprise systems [J]. arXiv: 160503175, 2016.

[4] BARESI L, GARRIGA M, RENZIS A D. Microservices identification through interface analysis[C]// Proceedings of the European Conference on Service-Oriented and Cloud Computing. 2017.

[5] CHEN R, LI S, LI Z. From monolith to microservices: A dataflow-driven approach[C]// Proceedings of the 2017 24th Asia-Pacific Software Engineering Conference (APSEC). 2017.

[6] MAZLAMI G, CITO J, LEITNER P. Extraction of microservices from monolithic software architectures[C]// Proceedings of the 2017 IEEE International Conference on Web Services(ICWS). 2017.

[7] ESKI S, BUZLUCA F. An automatic extraction approach: Transition to microservices architecture from monolithic application

[C]// Proceedings of the Proceedings of the 19th International Conference on Agile Software Development: Companion. 2018.

[8] KAMIMURA M, YANO K, HATANO T, et al. Extracting candidates of microservices from monolithic application code[C]// Proceedings of the 2018 25th Asia-Pacific Software Engineering Conference(APSEC). 2018.

[9] RADEMACHER F, SORGALLA J, SACHWEH S. Challenges of domain-driven microservice design: A model-driven perspective [J]. IEEE Software, 2018, 35(3): 36-43.

[10] DING D, PENG X, GUO X F, et al. Scenario-Driven and Bottom-Up Microservice Decomposition for Monolithic Systems [J]. Ruan Jian Xue Bao/Journal of Software, 2020, 31(11): 3461-3480.

[11] LI S S, RONG G P, GAO Q Y, et al. Optimized dataflow-driven approach for microservices-oriented decomposition[J]. Ruan Jian Xue Bao/Journal of Software, 2021, 32(5): 1284-1301.

[12] JIN W, LIU T, CAI Y, et al. Service Candidate Identification from Monolithic Systems based on Execution Traces [J]. IEEE Transactions on Software Engineering, 2019, 47(5): 987-1007.

[13] DESAI U, BANDYOPADHYAY S, TAMILSELVAM S. Graph Neural Network to Dilute Outliers for Refactoring Monolith Application[C]// Proceedings of the AAAI Conference on Artificial Intelligence. 2021.

[14] MITCHELL B S, MANCORIDIS S. On the Automatic Modularization of Software Systems Using the Bunch Tool [J]. IEEE Transaction on Software Engineering, 2006, 32(3): 193-208.

[15] MAZLAMI G, CITO J, LEITNER P. Extraction of Microservices from Monolithic Software Architectures[C]// Proceedings of the IEEE International Conference on Web Services. 2017.

[16] KALIA A K, JIN X, CHEN L, et al. Mono2Micro: an AI-based toolchain for evolving monolithic enterprise applications to a microservice architecture[C]// Proceedings of the Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020.

[17] SAEIDI A M, HAGE J, KHADKA R, et al. A search-based approach to multi-view clustering of software systems[C]// Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). 2015.

[18] SCANNIELLO G, D'AMICO A, D'AMICO C, et al. An approach for architectural layer recovery[C]// Proceedings of the Acm Symposium on Applied Computing. 2010.

[19] WU J, HASSAN A E, HOLT R C. Comparison of clustering algorithms in the context of software evolution[C]// Proceedings of the IEEE International Conference on Software Maintenance. 2005.

HUANG Zhicheng, born in 1995, master candidate. His main research interests include micro service, cloud native.

LIU Xianhui, born in 1979, Ph.D, associate researcher. His main research interests include machine learning, data mining andbig data, networked manufacturing.