Scholar Publishing Group
Machine Learning Theory and Practice
https://doi.org/10.38007/ML.2025.050107
ISSN 2790-0983 Vol. 5, Issue 1: 64-75

Software Engineering Practice of Microservice Architecture in Full Stack Development: From Architecture Design to Performance Optimization

Yuxin Wu[1, a*]

[1]College of Engineering, Carnegie Mellon University, Moffett Field 94035, CA, United States

[a]Email:yuxinwu202507@163.com

**Corresponding author*

Keywords: Single system, microservice architecture, multi feature fusion MicroRefactor、distributed transaction

Abstract： With the rapid development of the Internet and the growth of business requirements, the traditional single system is faced with dual challenges of maintenance and expansion due to its huge code base and highly coupled modules. The existing microservice migration methods rely on subjective judgment or single dimensional analysis, which makes it difficult to fully capture system characteristics. This article proposes a microservice reconstruction scheme based on multi feature fusion, integrating source code and runtime data to construct an inter class dependency undirected weighted graph model, extracting microservice modules through clustering analysis, and supporting database splitting strategies and distributed transactions to ensure data consistency. The core innovation includes a multi feature fusion mechanism that integrates semantic similarity, code structure dependencies, and runtime interactions; Develop the microservice extraction tool MicroRefactor, which significantly improves efficiency and accuracy through a three-stage process; Provide database splitting and consistency assurance solutions. Experimental verification shows that this scheme achieves efficient and accurate module extraction in JPetStore system and medium-sized e-commerce system. After reconstruction, the system functions are complete and performance optimization is significant. The current research is limited by the need to improve code automation processing capabilities and relies on manual design of test cases. In the future, we will explore automated processing technology and test case generation methods to further improve the refactoring plan.

1. Introduction

With the rapid development of the Internet and the growth of business requirements, the traditional single system faces the dual challenges of maintenance and expansion due to its complex and huge code base. Its highly coupled modules lead to any functional modification that may cause a global impact, significantly reducing development efficiency and increasing deployment risk. To address these challenges, microservice architecture has gradually become a mainstream solution. Its core is to break down monolithic applications into a set of small, autonomous services[1-3], each focusing on specific business functions, following the principle of high cohesion and low coupling, supporting multilingual development and independent deployment, thereby enhancing the flexibility and scalability of the system. However, the key to the migration of monolithic systems to microservices lies in efficiently and accurately extracting microservice modules. Existing methods have obvious limitations: Domain Driven Design (DDD) relies on the subjective judgment of domain experts to construct domain models, which is highly subjective; Automatic or semi-automatic extraction techniques are often based on single dimensional analysis of system dependencies such as code analysis and log tracking, making it difficult to fully capture complex characteristics such as code structure and runtime interactions (such as code analysis may ignore runtime dependencies, and system load analysis may not cover the entire code path)[4-5], resulting in a lack of comprehensiveness and objectivity in the extraction results. In response to this, this article proposes a multi feature fusion based single system microservice reconstruction scheme. By integrating system source code and runtime data, an undirected weighted graph model reflecting inter class dependencies is constructed, and microservice modules are extracted using clustering analysis techniques[6-7]; Simultaneously design a supporting database splitting strategy, utilize distributed transactions to ensure data consistency, and implement engineering implementation based on Spring Cloud and Spring Boot. The core contributions of the solution include: a multi feature fusion mechanism that integrates semantic similarity, code structure dependencies, and runtime interaction characteristics to comprehensively and objectively capture system features; Develop MicroRefactor tool to improve extraction efficiency and accuracy through a three-stage process of preprocessing, weighted average calculation of inter class correlation values, and weighted undirected graph clustering[8-9]; Using the commonly used experimental system JPetStore and medium-sized e-commerce system in software engineering as case studies to verify their effectiveness, the reconstructed system has complete functionality and significantly improved performance; Provide database splitting strategies and data consistency assurance solutions to reduce the difficulty of refactoring implementation[10-11].

2. Correlation Theory

2.1. Monolithic vs. Microservice: Comparative Analysis and Scenario-Based Trade-off

In the field of software engineering, monolithic architecture and microservice architecture are two core application design patterns, each with distinct characteristics and applicable scenarios.The monolithic architecture (as shown in Figure 1) treats the entire application as a single entity, integrating all functional modules internally, typically deployed on a web server in the form of a web application archive (WAR) file, and distributing traffic through a load balancing server[12-13].

Its advantages lie in its easy implementation and maintenance, especially suitable for small projects with low initial development costs and short iteration cycles. However, as the project scale expands and business complexity increases, the limitations of monolithic architecture gradually become apparent: the high coupling of modules leads to a decrease in code maintainability and an increase in update difficulty; The scalability of the system is limited, and when dealing with high concurrency scenarios, it can only be addressed by increasing server resources, which is costly and has limited effectiveness, ultimately affecting the system's response speed and stability.

The microservice architecture (as shown in Figure 2) adopts a distributed and modular design, decomposing complex systems into multiple small, autonomous service units, each running independently within a process and interacting through lightweight communication mechanisms such as RESTful APIs and message queues.

Compared to monolithic architecture, its core advantages are reflected in: 1) Scalability: Service units can independently and horizontally expand, enabling on-demand resource allocation and improving system performance; 2) Flexibility: The loose coupling feature supports independent development, deployment, and upgrade of services, reducing the impact on other services or systems; 3) Maintainability: Code modularization facilitates understanding, modification, and unit testing, improving manageability; 4) Technical diversity: Different services can choose the most suitable technology stack and programming language based on functional requirements, enhancing the flexibility of system construction and maintenance[14-15].

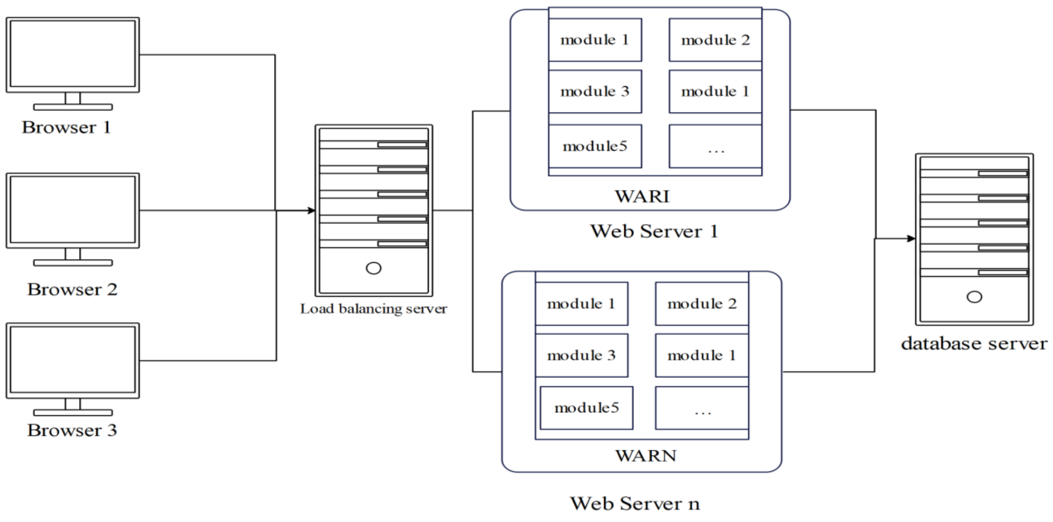

Figure 1. Web Application Architecture of Multi Server Centralized Database with Load Balancing

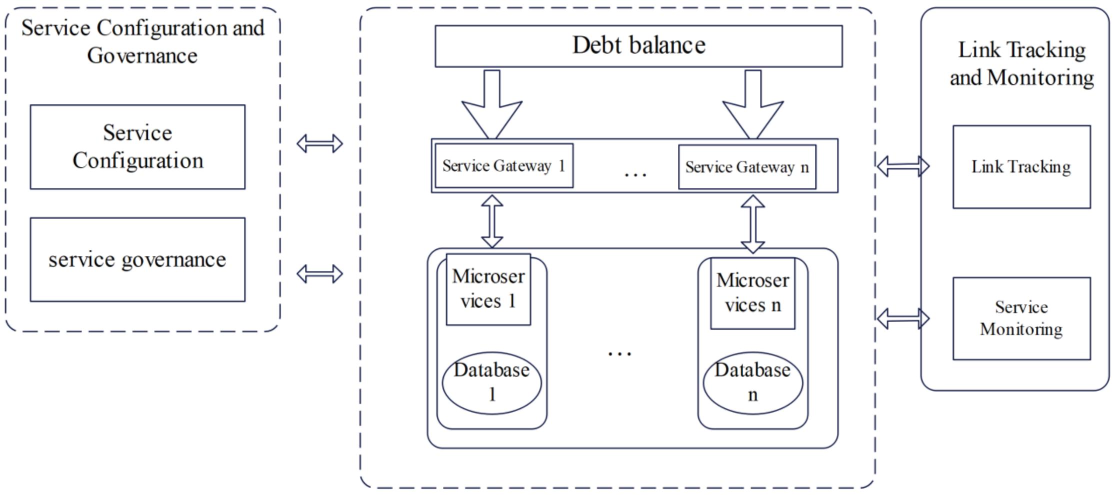

Figure 2. Microservices Architecture Diagram

2.2. Collaborative application of Microservices using Spring Boot, Spring Cloud, and Maven POM

Spring Boot, as an open-source Java development framework based on the Spring framework, aims to optimize application lifecycle management (startup, development, configuration, deployment, running, monitoring). It significantly reduces code volume through annotation driven development, integrates traditional XML/Properties configuration into a concise YML format, and improves configuration management efficiency; Built in Tomcat/Jetty/Undertow servers, supporting a lightweight runtime mode that does not require WAR package deployment, simplifying the deployment process; Integrating the Actor package to implement application monitoring functionality and simplify package management dependency configuration has become the preferred tool for developing individual microservices in microservice architecture[16-18].

Spring Cloud is built on Spring Boot and is a key framework for microservice architecture governance. By integrating components such as Eureka (service registration and discovery), Zuul (API gateway), Hystrix (fuse), etc., it provides service governance tools required for distributed systems (such as load balancing and configuration management) and supports efficient deployment of cloud native applications on Docker/PaaS platforms. It is deeply bound with Spring Boot and relies on the latter to achieve rapid development and independent deployment of service instances, jointly forming the technical stack of Java microservice development [19-20].

The Maven Project Object Model (POM), as a Java build automation tool, standardizes project build, reporting, and document management processes with POM as the core. It achieves automated build through lifecycle management and plugin mechanisms, following the principle of "convention over configuration" to reduce configuration complexity. At the same time, it extends functionality through a rich plugin ecosystem (compatible with Spring Boot, JUnit, Git, etc.), reducing developers' investment in building tool configuration and making it more focused on business logic development.

The synergistic effect of the three: Spring Boot simplifies microservice unit development, Spring Cloud provides distributed governance capabilities, Maven POM unifies construction and dependency management, jointly building an efficient and scalable Java microservice development system. In terms of limitations, Spring Cloud strictly relies on Spring Boot and cannot be used independently; Maven's principle of 'agreement over configuration' may limit flexibility in complex project scenarios.

3. Research Method

3.1. Research on Microservice Refactoring Scheme for Single System Based on Multi Feature Fusion

The current research on microservice refactoring faces two core challenges: Domain Driven Design (DDD) relies on subjective judgments of domain experts, which can lead to excessive abstraction of business concepts and affect the accuracy of microservice partitioning; Although automated microservice identification technology reduces subjective dependencies, it often focuses on a single dimension (such as code structure or runtime interaction), making it difficult to fully capture system features, resulting in poor performance or weak adaptability of the architecture after refactoring. Some tools, such as ServiceButton, require a complete product in the software design phase and have high method complexity, which limits their applicability in monolithic system refactoring. In response to the above issues, this article proposes a single system microservice reconstruction scheme based on multi feature fusion, as shown in its overall architecture. The solution constructs a comprehensive system feature model through multidimensional analysis of individual systems, reduces subjectivity, and can be operated with only source code, making it more applicable. The refactoring process is divided into five stages: first, parse the source code of the monolithic system, construct a set of classes (abstracting each class as a vertex of the graph), and formally represent the class relationship graph through the triplet G=(C, R, W) (Figure 3).

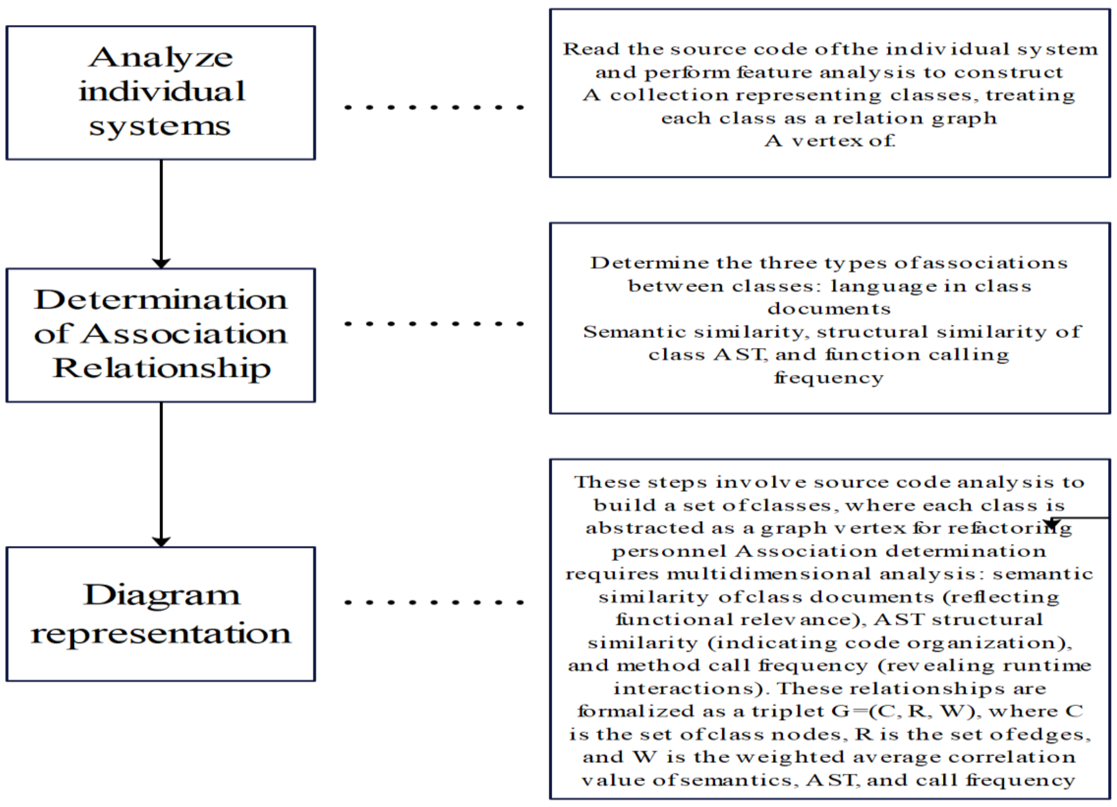
Analyze individual systems
Read the source code ofthe individual system and perform feature analysis to construct A collection representing classes, treating each class as a relation graph A vertex of
Determination of Association Relationship
Determine the three types ofassociations between classes: language in class documents Semantic similarity, structural similarity of class AST, and function calling frequency
Diagram representation
These steps involve source code analysis to build a set ofclasses, where each class is abstracted as a graph vertex for refactoring personnel Association determination requires multidimensional analysis: semantic similarity ofclass documents (reflecting functional relevance), AST structural similarity (indicating code organization), and method call frequency (revealing runtime interactions). These relationships are formalized as a triplet G=(C, R, W), where C is the set ofclass nodes, R is the set ofedges, and W is the weighted average correlation value ofsemantics, AST, and call frequency

Figure 3. Class Relationship Diagram Construction Process

among them, C is the set of class nodes, R is the set of inter class relationships, and W is the comprehensive correlation value (calculated by weighting semantic similarity, AST structure similarity, and function call frequency). Next, perform multi feature correlation value calculation, integrating semantic similarity (measuring the tightness of business logic), AST structure similarity (reflecting code structure correlation), and function call frequency (revealing runtime interaction activity) to obtain a comprehensive correlation value through weighted averaging. The higher the comprehensive correlation value, the closer the inter class relationship, and the more it should be classified into the same microservice. Then, the Chinese Whisper algorithm is used to perform clustering analysis on the class relationship graph, with the input being a triplet G=(C, R, W). Through iterative optimization of clustering labels, a microservice extraction scheme (clustering distribution of class nodes) is output to improve cohesion and reduce coupling. Then enter the database splitting stage, splitting the tables corresponding to entity classes in the original monolithic database into different microservices; Release foreign key constraints or table connection operations across microservices and retrieve data through API calls; For non entity class tables (such as log tables), build a "non entity class table access microservice" to achieve interaction; Introduce distributed transactions to ensure data consistency. Finally, a microservice governance architecture is built based on Spring Cloud (integrating Eureka service registration, Zuul gateway, Hystrix fuses, and other components). The extracted class code is encapsulated using Spring Boot, and service calls are implemented through Feign. Hystrix handles call exceptions.

3.2. Multi feature fusion class relationship modeling and microservice reconstruction method

This section proposes a multi feature fusion method for calculating the comprehensive correlation value of class relationships, which is used to construct the class relationship graph required for microservice reconstruction. This method integrates three types of association features: semantic similarity of class documents, structural similarity of abstract syntax tree (AST), and dynamic interaction frequency during system runtime, and uses a weighted average strategy to generate comprehensive inter class association values. In microservice refactoring, class documents serve as the core carrier of functional logic, and their semantic similarity directly reflects the functional correlation between classes. This study uses the BERT model to calculate the deep semantic similarity between class documents. The specific process is as follows: first, unify the encoding format of class documents to UTF-8, extract core elements such as class name, method name, variable name, etc., and generate feature files in the form of "class name. info"; Then use the BERT tokenizer to perform fine-grained segmentation on the feature file, taking the hidden state of the last layer of the model as the class document vector representation; Finally, the cosine similarity is used to measure the degree of semantic correlation between vectors, and the calculation formula is

$$R_{c_{semantic}}(C_i, C_j) = \frac{classVector_i \cdot classVector_j}{\|classVector_i\| \times \|classVector_j\|} \quad (1)$$

the range of values is [0,1], with larger values indicating tighter semantic associations. The similarity of class AST structure reflects the cohesion of code. High similarity classes usually have inheritance or interface implementation relationships. This study proposes an AST similarity algorithm based on Levenshtein distance. The steps are as follows: convert the class AST into a string representation and initialize the (m+1) × (n+1) dimensional group D (m, n is the string length), fill in an increasing sequence from 0 to m/n, and calculate the minimum editing distance through dynamic programming

$$\mathrm{D[i][j]} = \min(\mathrm{D[i-1][j]} + 1, \mathrm{D[i][j-1]} + 1, \mathrm{D[i-1][j-1]} + \mathrm{cost}) \quad (1)$$

(where cost is the character matching judgment value, 0 for the same and 1 for different), and finally normalized to obtain similarity.The algorithm flow is shown in Figure 4.

The frequency of inter class interactions during system runtime reflects the strength of functional coupling. In this study, a self-developed AOP link tracing tool was used to record the function call behavior under test case coverage

This indicator is implemented through non-invasive AOP technology, accurately reflecting the degree of correlation between classes during actual runtime. To balance the three types of correlation features, the weighted average method is used to generate the comprehensive correlation value, and the calculation formula is

$$\mathrm{W} = \mathrm{W}_1 \times R_{c_{semantic}} + W_2 \times R_{C_{AST}} + W_2 \times R_{C_{functionCall}} \quad (1)$$

Among them, the weight coefficients meet the requirements. Table 1 provides three preset weight schemes (semantic similarity priority: 0.5, 0.25, 0.25; structural similarity priority: 0.25, 0.5, 0.25; dynamic interaction priority: 0.25, 0.25, 0.5), which support flexible configuration according to reconstruction needs.

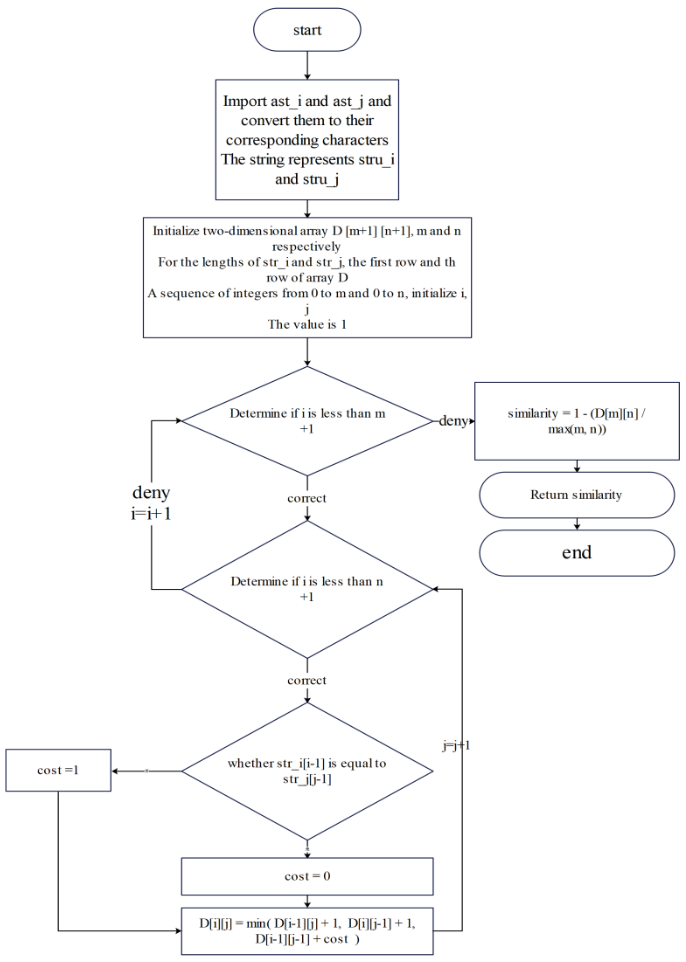
start
Import ast_i and ast_j and convert them to their corresponding characters The string represents stru_i and stru_j
Initialize two-dimensional array D [m+1] [n+1], m and n respectively For the lengths of str_i and str_j, the first row and th row of array D A sequence of integers from 0 to m and 0 to n, initialize i, j The value is 1
Determine if i is less than m +1
deny
similarity = 1 - (D[m][n] / max(m, n))
Return similarity
end
deny
i=i+1
correct
Determine if i is less than n +1
correct
j=j+1
cost =1
whether str_i[i-1] is equal to str_j[j-1]
cost = 0
D[i][j] = min(D[i-1][j] + 1, D[i][j-1] + 1, D[i-1][j-1] + cost)

Figure 4. AST Similarity Calculation Process Diagram Between Classes

Table 1. Weight Allocation Table for Multi-Feature Fusion

Focus	W_1 (Semantic)	W_2 (AST)	W_3 (Interaction)
Semantic Priority	0.5	0.25	0.25
Structural Priority	0.25	0.5	0.25
Dynamic Priority	0.25	0.25	0.5

3.3. Mechanism Microservice extraction and database splitting methods

After completing the comprehensive correlation value calculation of class relationships, microservice extraction can be achieved through unsupervised graph clustering. This article adopts the Chinese Whisper algorithm, which classifies nodes with similar association relationships into the same cluster through label propagation, automatically determines the number of clusters, and is suitable for processing large-scale sparse graph data and robust to noise. The specific process is as follows: import a class relationship graph G containing n nodes (node numbers 0 to n-1, initial labels are numbers), set the number of iterations and association threshold; In each iteration, the edge weights of all categories of neighboring nodes are accumulated for each node, and the category with the highest weight is selected (if there are multiple maximum values, the smallest number is selected). If the weight of the category exceeds the threshold, the node is classified into this category. Otherwise, the original category is maintained; Repeat until the preset number of iterations is reached, and the final label of the output node completes clustering. Threshold selection needs to balance stability and flexibility, usually initialized to 0.1 or 0.3 and adjusted according to data.Database splitting should follow the principle of single responsibility, with each microservice corresponding to an independent database. According to whether the data table has corresponding entity classes, the splitting strategy can be divided into two categories: for tables with entity classes, they are assigned to the corresponding microservice database, the database foreign key constraints are lifted, and the foreign key related operations are encapsulated as APIs through the ORM framework. Cross table logic is processed at the application layer (such as after splitting the order and customer tables, the order service calls the customer service through the API to verify the customer ID); For tables without entity classes (such as region tables), entity classes are generated through ORM reverse engineering, and a data access layer (DAO), business logic layer (Service), and control layer (Controller) are built, encapsulated as independent microservices (such as "table access services without entity classes"), providing CRUD interfaces for other services to call, avoiding the complexity caused by database triggers or middleware.Distributed transactions are the core of ensuring data consistency after database splitting. Comparing two-stage commit (2PC), three-stage commit (3PC), and transaction compensation mode (TCC), this article chooses the TCC scheme because it has a small impact on system performance and can ensure final consistency, despite complex business logic and increased reconstruction costs. TCC is divided into three stages: Try stage to reserve resources, Confirm stage to confirm operations and use reserved resources, Cancel stage to cancel operations and release resources. During implementation, it is necessary to handle exceptions such as null rollback (Confirm/Cancel not executed), idempotency (repeated operations have no impact), and hanging (Cancel not executed after Try failure), and ensure consistency by recording transaction status and checking transaction IDs.The Spring Cloud framework is chosen for microservice implementation, and its mature ecosystem, flexible component selection, and extensive community support are key advantages. The architecture design includes core components: Eureka is responsible for service registration and discovery, Zuul serves as a gateway to receive HTTP requests and implement load balancing distribution through Ribbon, Feign or RestTemplate is used for inter service communication, Spring Boot Admin integrates Eureka information to monitor service running status, Hystrix fuses provide fault tolerance mechanisms to avoid microservice avalanche effects.Microservice encapsulation relies on the Spring Boot framework, and the process includes: creating a Spring Boot project and defining service interfaces and implementation classes, using Spring annotations to mark externally callable service methods, configuring basic information such as service names and ports in the configuration file, and finally registering the service to the Eureka service center through the startup class. In terms of service invocation, Feign simplifies the invocation side encoding through declarative interfaces, while RestTemplate requires manual construction of HTTP requests, making it more suitable for complex business logic. In response to Feign call exceptions, combined with Hystrix's processing flow, the following steps are taken: defining Feign client interface to declare service endpoints and methods, writing a service downgrade handling class to trigger downgrade logic when the call fails, enabling Feign and Hystrix support in the configuration file, and finally implementing service calls and exception handling by injecting Feign client.

4. Results and Discussion

4.1. Design and experimental verification of MicroRefactor microservice automatic extraction tool

MicroRefactor is a single system microservice automatic extraction tool based on multi feature fusion. Its core functions are divided into three modules: reconstruction preprocessing, multi feature fusion analysis, and class relationship graph generation and clustering analysis. Refactoring the preprocessing module to verify the integrity of the source code package through SourceCodePreprocessor, unify the encoding to UTF-8, and repackage it; SemanticSimilarity Preprocessor extracts representation information such as class names, method names, and variable names and saves them to a local file; ASTSimilarityPreprocessor converts source code into a Class class file, builds an Abstract Syntax Tree (AST), and persistently stores it. The multi feature fusion analysis module integrates three types of correlation indicators: BertSemanticSimilarity Analyzer calculates the semantic similarity of class documents, ASTSimilarityAnalyzer quantifies the structural differences of AST using Levenshtein distance algorithm, DynamicInteractionFrequency Analyzer calculates the inter class interaction frequency based on AOP link tracking tool, and finally generates a comprehensive correlation value by weighted fusion of the three types of indicators through MultiFeature Fusion Analyzer. The class relation graph generation and clustering analysis module utilizes GraphGenerator to construct undirected weighted graphs, uses Chinese Whisper algorithm for label propagation clustering, outputs microservice extraction schemes, and supports result export.The tool implementation adopts mixed programming of Java and Python, with Java encapsulating core functions and Python calling Java methods through PyJNIus. The development environment includes JDK 8u202, IntelliJ IDEA 2022.1, Python 3.5.10, and PyCharm 2022.1. The tool interface is based on the B/S architecture. The left navigation bar includes functions such as refactoring preprocessing, microservice extraction, and system help. The right side displays the extraction results and supports export.The experimental verification takes the classic monomer system JPetStore as the object, and uses cohesion (COH) and coupling (COP) as evaluation indicators. By setting weight parameters (semantic similarity weight of 0.5, AST similarity of 0.25, interaction frequency of 0.25), the comprehensive inter class correlation value is calculated and a relationship graph is constructed. The Chinese Whisper algorithm (threshold of 0.1) is used for clustering. The experimental results show that the microservice scheme extracted by MicroRefactor has higher consistency on the SC1 and SC2 microservice components compared to the method in reference [51], and is basically consistent on SC3; The cohesion (COH) increased from 2.29 to 2.59, and the coupling (COP) decreased from 2.56 to 2.50, verifying the effectiveness of the tool in improving cohesion and reducing coupling.

4.2. Model experiment

After completing microservice extraction and database splitting, it is necessary to name and structurally encapsulate the extracted microservices. Based on the extracted class clusters, the class clusters related to product information and search functions (such as Product and SearchResult related classes) are named as product microservices, the class clusters related to user registration and login are named as user microservices, the class clusters related to shopping cart operations are named as shopping cart microservices, and the class clusters related to order creation and payment are named as order microservices. At the same time, independent "entity free class access" microservices are encapsulated for data tables (such as regional dictionary tables) corresponding to entity free classes. Each microservice adopts a layered architecture: the control layer (such as UserController, productController) processes HTTP requests, the business logic layer (such as UserServiceImpl, productServiceImpl) implements core functions, the data access layer (such as UserMapper, productMapper) encapsulates database operations, entity classes (such as User, Product) define data structures, and are equipped with startup classes (such as UserServiceApplication) for microservice instantiation.During the deployment phase, a microservice architecture is built based on the Spring Cloud framework, integrating Eureka as the service registry, Zuul as the gateway to handle request routing, Ribbon for client load balancing, Hystrix for circuit breaker downgrade mechanism, and Spring Boot Admin for service monitoring. The specific deployment process includes: creating an independent microservice module and managing dependencies through Maven, configuring Eureka Server and Client to achieve service registration and discovery, enabling Zul gateway to define routing rules, integrating Ribbon and Feign to simplify service calls, configuring Hystrix to prevent avalanche effects, and integrating Eureka through Spring Boot Admin to achieve service status monitoring. After the user request is initially processed by the Zuul gateway, the target service instance is discovered through Eureka and forwarded to the specific microservice through Ribbon load balancing. After processing is completed, the result is returned to the gateway and the client responds, forming a complete request processing chain.

4.3. Effect Analysis

This article proposes a microservice reconstruction scheme for monolithic systems based on multi feature fusion. The core steps include feature fusion, database splitting, distributed transaction design, and tool implementation and verification.The solution first integrates three types of features: semantic similarity of class documents, structural similarity of abstract syntax tree (AST), and frequency of runtime interaction. It generates inter class association values through weighted averaging and uses unsupervised graph clustering algorithm (Chinese Whisper) to extract microservices. During the iteration process, nodes are reclassified based on the weights of adjacent nodes, and a threshold (usually 0.1-0.3) is set to balance stability and flexibility.Database splitting follows the principle of single responsibility, with entity class corresponding tables assigned to microservice databases, foreign key constraints removed, and foreign key operations encapsulated as APIs through ORM. Entity class tables (such as region tables) are generated in reverse through ORM to build DAO/Service/Controller layers, encapsulated as independent microservices (such as "entity class table access services"), avoiding trigger/middleware complexity.The distributed transaction adopts the TCC (Try Confirm Cancel) mode to ensure consistency: resources are reserved in the Try stage (such as order service creation records and reserved inventory of goods and services), confirmed and consumed in the Confirm stage, and resources are released in the Cancel stage (such as rolling back inventory when an order is cancelled). Need to handle empty rollback, idempotency, and hanging transactions, ensuring consistency through state records and transaction ID verification.In terms of tools, we will develop a mixed programming tool MicroRefactor (Java core+Python call), which supports pre-processing, extraction, and result export in the B/S architecture interface. The experiment used JPetStore as the object, with semantic weights of 0.5, AST and dynamic features of 0.25 each, and a threshold of 0.1. After clustering, the cohesion (2.59) was higher than the comparison method (2.29), and the coupling degree (2.50) was lower. After the actual e-commerce system reconstruction, the cohesion increased by 30% (1.87 → 2.43), the coupling degree decreased by 15% (3.12 → 2.65), and four microservices including user management and product search were successfully extracted.The current tool requires manual setting of feature weights and relies on a full set of test cases for link tracking (which takes about 2 hours in the middle system). Automation functions (such as interface document generation) need to be improved. The future plan is to introduce machine learning automatic parameter tuning, combine fuzzy testing to generate high coverage test cases, and enhance link tracing overhead through bytecode to expand cross language support.

5. Conclusion

Microservice architecture, with its advantages of modularity, scalability, and technology independence, provides a new path for extending the lifecycle of individual systems and improving efficiency, but specific transformation plans are relatively scarce. Therefore, this article proposes a single system reconstruction scheme based on multi feature fusion, covering five core contents: problem modeling, multi feature correlation value calculation, microservice extraction, database splitting, and reconstruction implementation. Research has shown that existing domain driven modeling has strong subjectivity, while automatic or semi-automatic methods often approach from a single dimension. Therefore, this approach enhances the scientificity of reconstruction through multi feature fusion. Based on this solution, a microservice extraction tool MicroRefactor was developed, which significantly improved the efficiency of microservice extraction for individual systems. The tool adopts a three-stage process: first, preprocessing is performed, then three types of inter class correlation values are calculated and weighted averaged, and finally a weighted undirected graph is constructed for clustering to extract microservices. The effectiveness of the tool was verified through the commonly used experimental system JPetStore in the field of software engineering. Furthermore, taking a medium-sized e-commerce system as a practical object, the application of the tool in actual projects was demonstrated. The refactoring process covered automatic extraction, database splitting, distributed transaction ensuring data consistency, and microservice deployment based on Spring Cloud. Tests showed that the system functionality was correct, complete, and the performance was significantly improved after refactoring. However, there are still two shortcomings in current research: firstly, MicroRefactor needs to improve its automated code processing function after extraction; Secondly, tools rely on manually designed comprehensive functional test cases, which are costly and prone to omission for complex large-scale systems. In the future, we will explore code automation processing technology and automated functional test case generation methods.

References

[1] Jordanov J , Petrov P , Kuyumdzhiev I ,et al.Domain-Driven Design in Cloud Computing:. NET and Azure Case Analysis[J].TEM Journal, 2025, 14(1).DOI:10.18421/TEM141-05.

[2] Dobrinin M V .Extending RESTful web service resources in a JAVA-component-driven-architectureapplication:US18424015;US202400018424015;US202418424015A;US202418424015[P].US12238159B2;US2025012238159B2;US12238159B2;US12238159[2025-08-13].

[3] Tang S , Chen J , Wang D ,et al.MVDiffusion++: A Dense High-Resolution Multi-view Diffusion Model forSingle orSparse-View 3D Object Reconstruction[C]//European Conference on Computer Vision.Springer, Cham, 2025.DOI:10.1007/978-3-031-72640-8_10.

[4] Jadcherla S , Burnam M H .PHYSIOLOGICAL SENSING DEVICE HAVING DATA ACCESS LAYER, AND METHODS OF OPERATION OF SAME:USUS2024/041912;US202400000041912;US2024041912W;WO2024US41912[P].WO2025/035147A2;WO2025000035147A2;WO2025035147A2;WO2025035147[2025-08-13].

[5] Dai Z , Wang S , Lu Q ,et al.Nonlinear hysteresis system control based on sliding mode neural network and observer[J].Transactions of the Institute of Measurement & Control, 2025, 47(10).DOI:10.1177/01423312241279496.

[6] Zhu P. Construction and Experimental Verification of Automatic Classification Process Based on K-Mer Frequency Statistics[C]//The International Conference on Cyber Security Intelligence and Analytics. Cham: Springer Nature Switzerland, 2024: 391-400.

[7] Zhang Y. Research on Optimization and Security Management of Database Access Technology in the Era of Big Data[J]. Academic Journal of Computing & Information Science, 2025, 8(1): 8-12

[8] *Pan Y. Research on the Design of a Real-Time E-Commerce Recommendation System Based on Spark in the Context of Big Data[C]//2025 IEEE International Conference on Electronics, Energy Systems and Power Engineering (EESPE). IEEE, 2025: 1028-1033.*

[9] *Yan J. Analysis and Application of Spark Fast Data Recommendation Algorithm Based on Hadoop Platform[C]//2025 Asia-Europe Conference on Cybersecurity, Internet of Things and Soft Computing (CITSC). IEEE, 2025: 872-876.*

[10] *Wu, H. (2025). The Commercialization Path of Large Language Models in Start-Ups. European Journal of Business, Economics & Management, 1(3), 38-44.*

[11] *Xiu L. Analyses of Online Learning Behaviour Based on Linear Regression Algorithm[C]//2025 IEEE International Conference on Electronics, Energy Systems and Power Engineering (EESPE). IEEE, 2025: 1333-1338.*

[12] *Huang J. Resource Demand Prediction and Optimization Based on Time Series Analysis in Cloud Computing Platform[J]. Journal of Computer, Signal, and System Research, 2025, 2(5): 1-7.*

[13] *Cai, Y. (2025). Research on Positioning Technology of Smart Home Devices Based on Internet of Things. European Journal of AI, Computing & Informatics, 1(2), 80-86.*

[14] *Xu D. Design and Implementation of AI-Based Multi-Modal Video Content Processing[J]. European Journal of AI, Computing & Informatics, 2025, 1(2): 44-50.*

[15] *Zhu, Z. (2025). Cutting-Edge Challenges and Solutions for the Integration of Vector Database and AI Technology. European Journal of AI, Computing & Informatics, 1(2), 51-57.*

[16] *Huang, J. (2025). Reuse and Functional Renewal of Historical Buildings in the Context of Cultural Heritage Protection. International Journal of Humanities and Social Science, 1(1), 42-50.*

[17] *Lai L. Data-Driven Credit Risk Assessment and Optimization Strategy Exploration[J]. European Journal of Business, Economics & Management, 2025, 1(3): 24-30.*

[18] *Lu, C. (2025). The Application of Point Cloud Data Registration Algorithm Optimization in Smart City Infrastructure. European Journal of Engineering and Technologies, 1(1), 39-45.*

[19] *Ye, J. (2025). Optimization and Application of Gesture Classification Algorithm Based on EMG. Journal of Computer, Signal, and System Research, 2(5), 41-47.*

[20] *Zhu, Z. (2025). Application of Database Performance Optimization Technology in Large-Scale AI Infrastructure. European Journal of Engineering and Technologies, 1(1), 60-67.*