scientific reports

OPEN

Research on multi-dimensional reconstruction mechanism of cloud native full link in the metaverse scenario

Shuo Sheng

Recently, the Microservices whole link whole life cycle optimization framework has been a research hotspot in the academic and engineering fields, especially how to encapsulate the component security mechanism and conduct high concurrency testing based on the existing active framework. The whole link delay, load, and cost have always been the key optimization goals of Microservices splitting and deployment logic. Based on this, this paper proposes a multi-objective optimization Microservices framework that takes into account the security mechanism, Define the fitness function, define the upper and lower limits, and perform multi-dimensional constraints to filter for global and local optima. At the same time, reconstruct the circuit breaker current limiting mechanism, dynamically detect protocol parameters, and perceive the logical relationship of heartbeat status in real-time. Experimental results have shown that this innovative framework can solve scenarios such as high concurrency, high reliability, and high availability Performance bottleneck, business degradation, especially in financial and securities scenarios.

With the popularity of Microservices, every Internet company has numerous services of all sizes in the background, and there are countless invocation relationships between services. To ensure the maturity and stability of the entire Microservices system, it is necessary to ensure the maturity of each Microservices. But how to define the maturity of a service? What latitude should we consider? What are the common problems in various latitudes? How to optimize?

With the development of mobile Internet and Internet plus, the original SOA architecture has encountered four problems: ① lack of effective service governance, mixed service assets, and no effective service management and control means; ② The business support response is slow, and the system is too big to fail, unable to achieve real-time updates and modular release; ③ Poor system availability, unable to achieve 7×24 h uninterrupted service; ④ Innovative business is difficult to support, especially innovative business with Internet characteristics. Based on this, developers propose an SOA architecture based on Microservices with the starting point of architecture optimization[1–6]; Its core concept is to decompose complex application systems into multiple services in the form of independent business units, each of which can adopt different implementation technologies and be independently designed, developed, and deployed in a lightweight and more flexible mode, running in independent processes to form highly cohesive autonomous units.

Specifically, SOA divides the system into different services, uses interfaces for data interaction, and services are interdependent and effectively integrated to achieve the overall function of the system[7,8]. In terms of architecture, the Microservices architecture is an improvement based on SOA architecture, and incorporates the idea of componentization and domain modeling. First, the system is divided into multiple Microservices, which are deployed independently in a Loose coupling manner; Secondly, each Microservices only needs to complete its own tasks with high quality, and each task represents a fine-grained business capability; As a result, various businesses have been completely componentized and service-oriented[9–12]; Finally, modules that provide domain service capabilities realize service composition and assembly in the underlying Microservices architecture. As shown in Fig. 2a, the overall architecture puts all software functions in a process, and multiple servers jointly support the running of computing tasks, and finally returns the running results to the user; The Microservices architecture (Fig. 2b) decomposes the overall function of the software into multiple services supported by different types of servers; Then, the data is fed back to the database, and users can obtain data from the database,

Key Laboratory of Embedded System and Service Computing, Ministry of Education, Tongji University, Shanghai, China. email: positive_y@163.com

which not only speeds up the overall response speed of the system, but also meets the business needs of front and back end separation under the Internet environment. It can be seen that the advantages of the Microservices architecture[10] are mainly reflected in: distributed (physical deployment, service deployment, data storage), high availability (distributed architecture, clustered deployment, automatic service registration) Scalable (on-demand resource allocation), intelligent operation and maintenance, and other aspects.

With the increasing expansion and complexity of software system functions, the SOA architecture based on Microservices has begun to rise. Specifically, Microservices supports the rapid response and personalization of business in the foreground. It uses service-oriented development SOD (service-oriented development), which is characterized by rapid development, timely response, easy implementation, etc.[13]; The ESB service bus, as the back-end, supports the integration of various systems, technologies and platforms, while the service-oriented infrastructure (SOI)[14–16] has the characteristics of stability and high integration. The Microservices technology and ESB technology complement each other and play different core roles in the SOA architecture to achieve the self-service effect of software systems.

This article introduces a model used by the technical product teams of mainstream frontline large factories in the financial field to measure service maturity, and evaluates multiple backend services based on this model. It summarizes some common low sub items and organizes relevant optimization solutions for the low sub items.

Optimization model

Service maturity model

In Fig. 1, in order to quantify the maturity of services, this study needs to evaluate services from multiple dimensions. Each service iterates continuously with the requirements, and each iteration cycle needs to go through the following three stages: development testing, operation and maintenance online, and online operation. At each stage, a series of measures are needed to ensure the final maturity and reliability of the service. This study focuses on each stage and identifies the indicators that need to be addressed to measure the overall quality of that stage. This study comprehensively rated the above three stages and graded service maturity based on the ratings. (1–5 points are classified as admission level, 6–7 points are classified as service level, and 8–10 points are classified as maturity level.) Through such a service maturity model, this study can intuitively evaluate the maturity of a service using quantitative indicators and identify areas for improvement.

Low item sorting

Based on the service maturity model, this study evaluated multiple backend services and found that some indicator items generally scored lower in each service. These indicators include: development testing: code

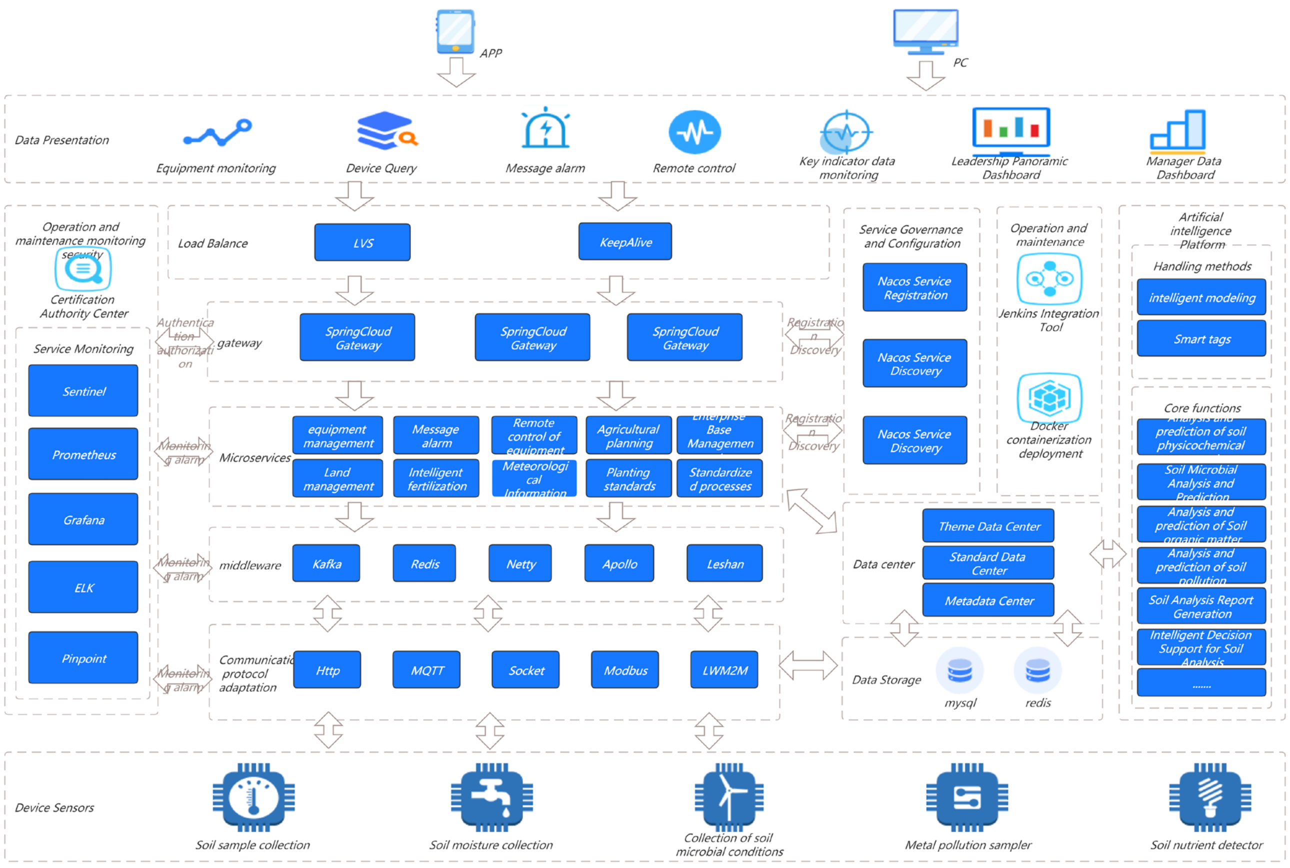


Figure 1. Full-link microservice security link topology.

a

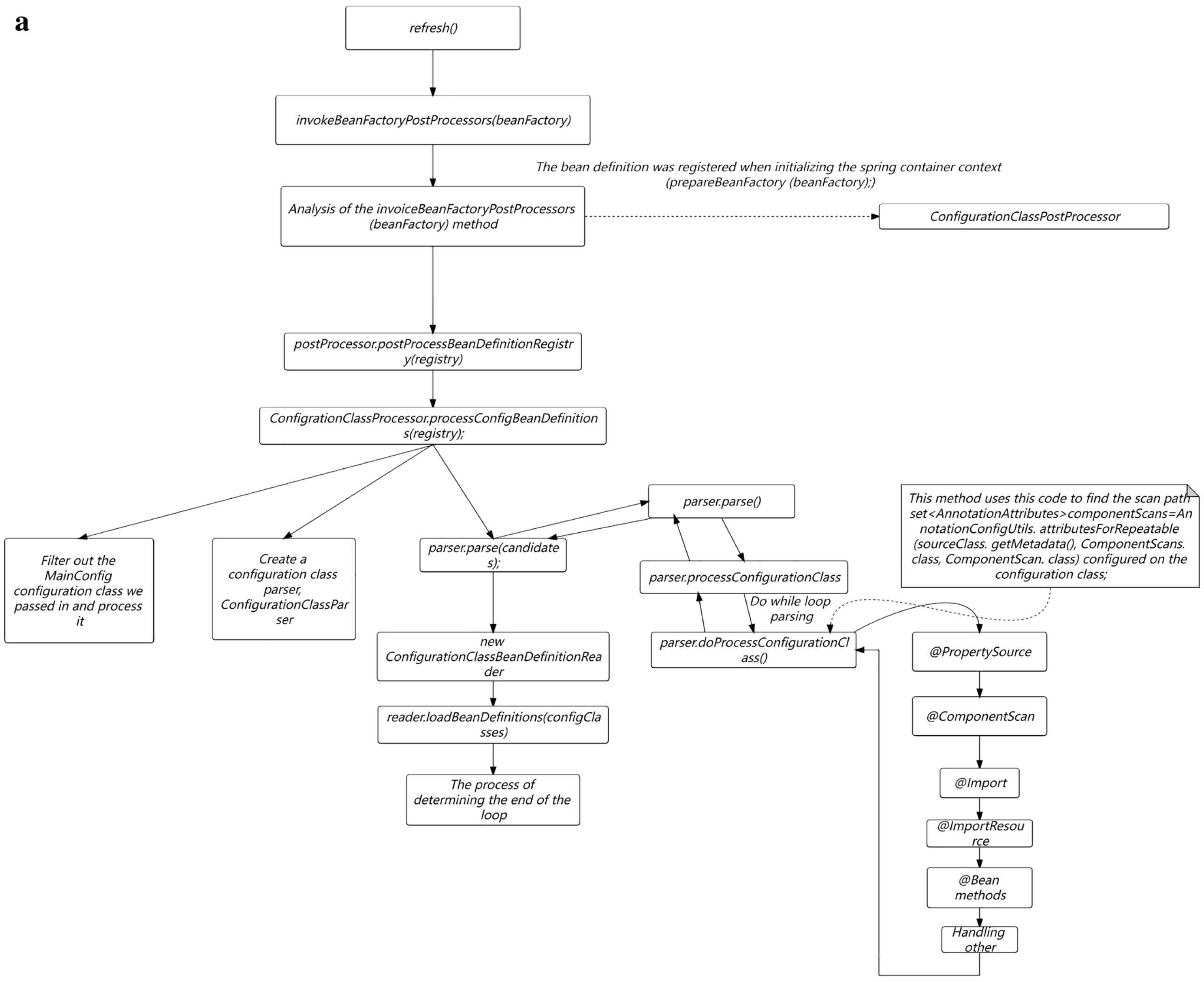

refresh()
invokeBeanFactoryPostProcessors(beanFactory)
The bean definition was registered when initializing the spring container context (prepareBeanFactory (beanFactory);)
Analysis of the invoiceBeanFactoryPostProcessors (beanFactory) method
ConfigurationClassPostProcessor
postProcessor.postProcessBeanDefinitionRegistry(registry)
ConfigrationClassProcessor.processConfigBeanDefinitions(registry);
Filter out the MainConfig configuration class we passed in and process it
Create a configuration class parser, ConfigurationClassParser
parser.parse(candidates);
parser.parse()
parser.processConfigurationClass
Do while loop parsing
parser.doProcessConfigurationClass()
This method uses this code to find the scan path set<AnnotationAttributes>componentScans=AnnotationConfigUtils. attributesForRepeatable (sourceClass. getMetadata(), ComponentScans. class, ComponentScan. class) configured on the configuration class;
new ConfigurationClassBeanDefinitionReader
reader.loadBeanDefinitions(configClasses)
The process of determining the end of the loop
@PropertySource
@ComponentScan
@Import
@ImportResource
@Bean methods
Handling other

Figure 2. (**a**) Full link testing core routing. (**b**) Full Link Metaverse Cloud Native Data Business Logic Flow View.

specification, unit testing, stress testing, interface testing; In terms of operation and maintenance online: grayscale online, online rollback; In terms of online availability: system protection, data backup, alarm processing, and emergency plans. Therefore, this study prioritized the optimization schemes for these low sub items, hoping to improve the maturity of services and elevate the services of this study to a mature level through the optimization of low sub items.

Low sub item optimization plan

In Fig. 2a,b, The services in this study are mainly developed based on the Java language, mainly using virtual machine deployment and a small amount of container deployment. So the optimization plan below mainly revolves around the current service status of this study.

Development testing

Online bugs are often introduced by developing new requirements. If problems are discovered as much as possible during the development and testing phase, improving code quality can minimize the impact of the problem. Here are some common measures and means during the development and testing phase:

Code specification

Programming style. Collaborating with GitLab CI and Sonar inspections, it is possible to conduct code specification checks on each submission and present them visually in the form of reports.

Unit testing

In Fig. 2a,b, Unit testing is located at the bottom of the testing pyramid. Fast running efficiency, typically completed test cases run for no more than 10 min, and can be integrated into the CI process to quickly provide feedback on issues. Low maintenance costs, unit testing can detect problems during the development phase, and

b

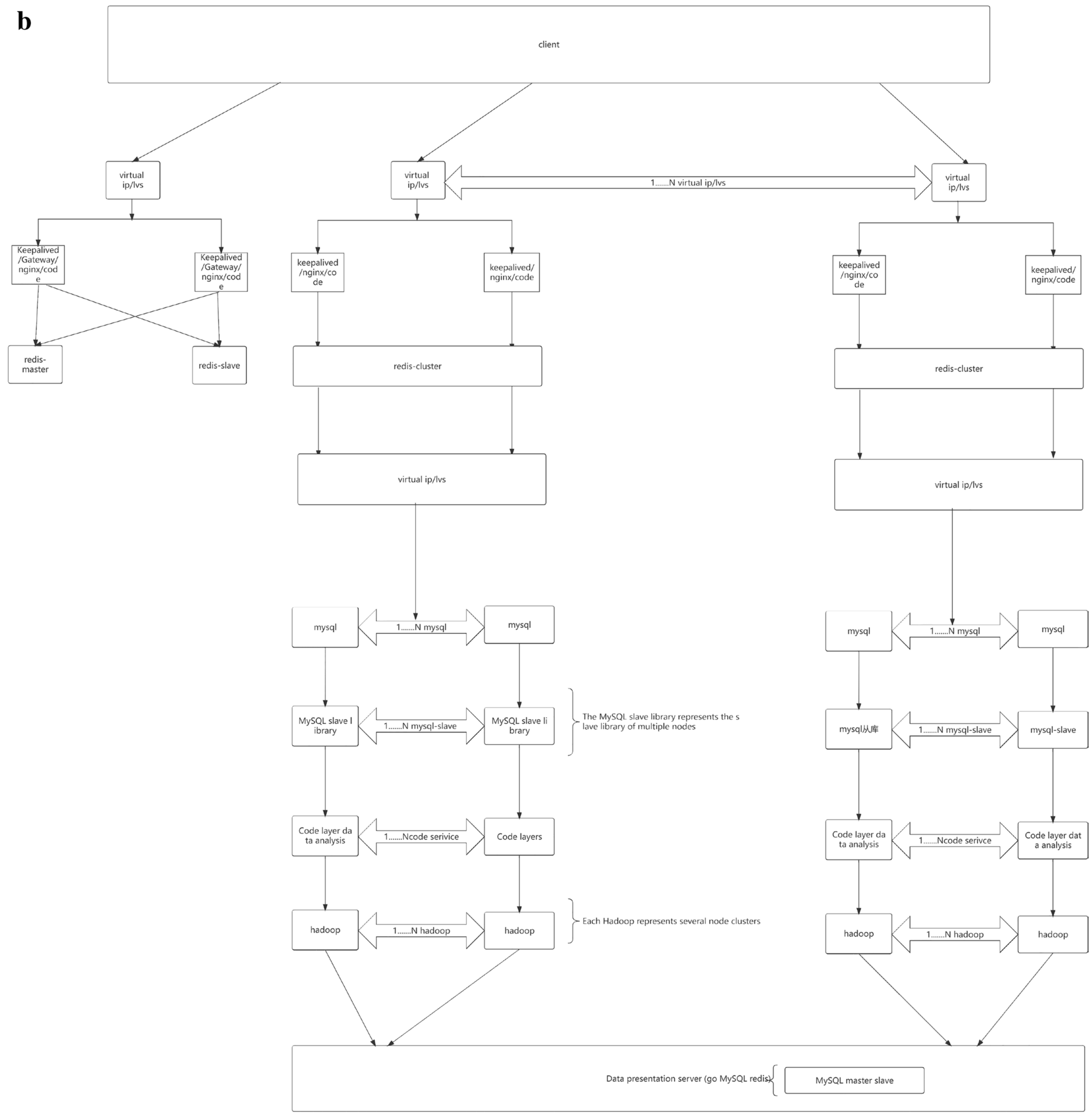


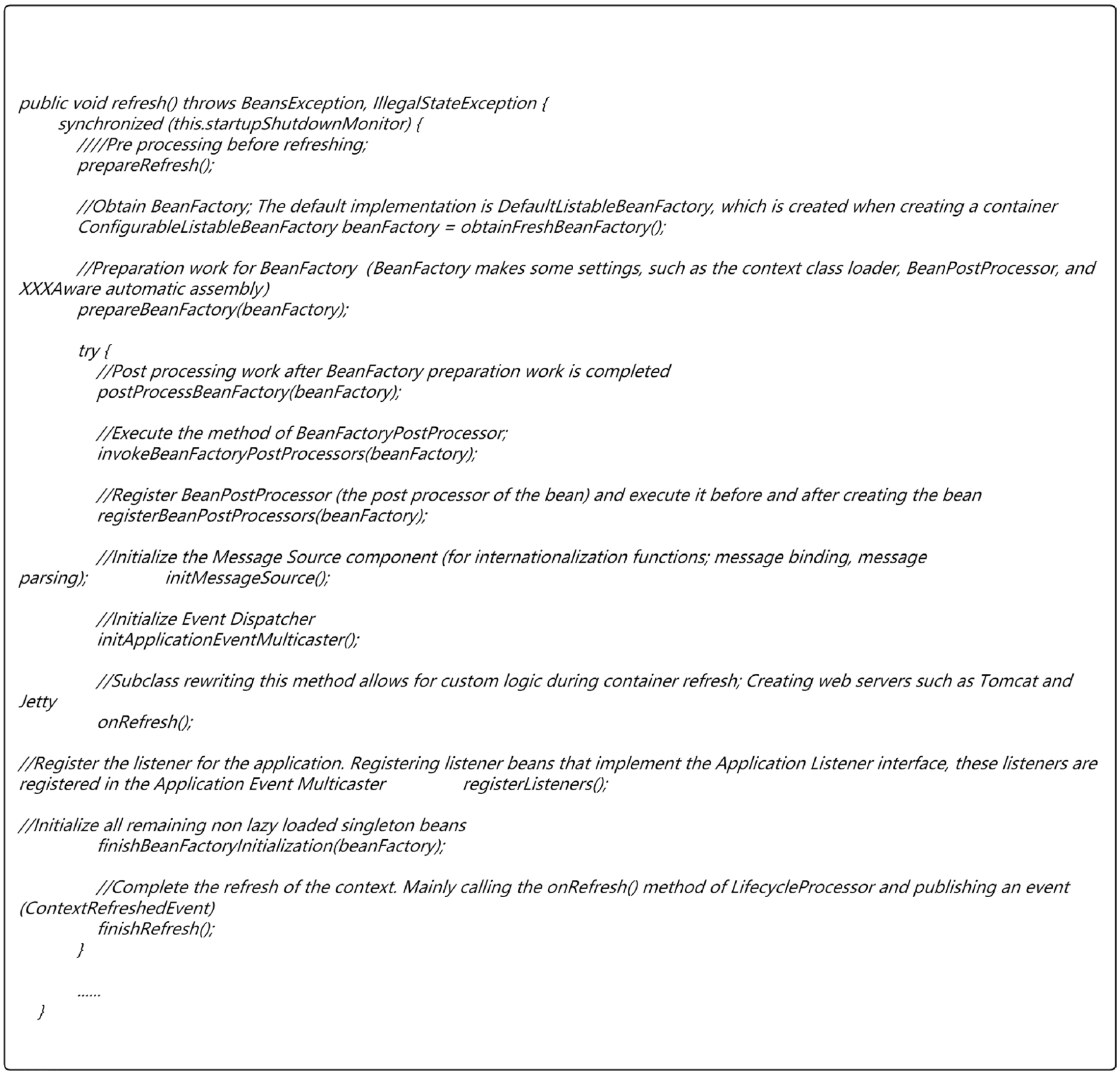
Figure 2. (continued)

repair costs are low. The longer the feedback time for testing the pyramid, the higher the cost of implementation. Therefore, it is recommended to increase the coverage of unit testing.

The most commonly used testing framework for Java is Junit. In addition, in conjunction with the Mock testing library Mockito and the testing coverage tool Jacobo, a complete set of unit testing tools is formed. Jacco plugins can be configured in both IDEA and SonarQube to visually display the coverage of unit testing.

In the process of promoting unit testing, the common problems encountered in this study are: supplementing unit testing for projects is time-consuming and laborious, and how to control the testing coverage of new code?

In Fig. 3, the solution of this study is to prioritize supplementing unit testing for important code (such as public libraries and tool classes) with existing code. Indicators for increasing Code coverage of new code (e.g., set the test coverage of new code to be no less than 90%); As new code is added and old code is modified, the overall testing coverage of the project will gradually increase. Add code testing coverage statistics: Using the diff cover tool, you can calculate the testing coverage of the current branch's modified code by analyzing branch code differences and Jacob test reports.



```
public void refresh() throws BeansException, IllegalStateException {
      synchronized (this.startupShutdownMonitor) {
         ////Pre processing before refreshing;
         prepareRefresh();

         //Obtain BeanFactory; The default implementation is DefaultListableBeanFactory, which is created when creating a container
         ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

         //Preparation work for BeanFactory  (BeanFactory makes some settings, such as the context class loader, BeanPostProcessor, and
XXXAware automatic assembly)
         prepareBeanFactory(beanFactory);

         try {
            //Post processing work after BeanFactory preparation work is completed
            postProcessBeanFactory(beanFactory);

            //Execute the method of BeanFactoryPostProcessor;
            invokeBeanFactoryPostProcessors(beanFactory);

            //Register BeanPostProcessor (the post processor of the bean) and execute it before and after creating the bean
            registerBeanPostProcessors(beanFactory);

            //Initialize the Message Source component (for internationalization functions; message binding, message
parsing);             initMessageSource();

            //Initialize Event Dispatcher
            initApplicationEventMulticaster();

            //Subclass rewriting this method allows for custom logic during container refresh; Creating web servers such as Tomcat and
Jetty
            onRefresh();

//Register the listener for the application. Registering listener beans that implement the Application Listener interface, these listeners are
registered in the Application Event Multicaster             registerListeners();

//Initialize all remaining non lazy loaded singleton beans
            finishBeanFactoryInitialization(beanFactory);

            //Complete the refresh of the context. Mainly calling the onRefresh() method of LifecycleProcessor and publishing an event
(ContextRefreshedEvent)
            finishRefresh();
         }

         ......
   }
```

Figure 3. Optimization of core branch business timing.

Unit testing

(1) Why do I need to do stress testing Through stress testing, this study was able to: identify performance bottlenecks; Estimate resource usage; Provide reference for current limiting configuration; Identify issues introduced by new requirements.
(2) What functions should stress testing tools have Common stress testing tools include JMeter and others. This study uses the LoadMaker cloud pressure testing system provided by mainstream cloud platforms in the financial field of first tier large enterprises, which has the following characteristics: multiple pressure clusters, and the pressure can reach high levels; Create and save pressure testing scenarios, repeat execution; Provide pressure test reports: QPS, response time, error rate, and other data; Meanwhile, in order to ensure that pressure testing does not affect the actual traffic of online users, this study constructed a separate pressure testing resource pool. When pressure testing is required, selecting machines and pressure testing modules in the resource pool can automatically build a pressure testing environment.
(3) When is pressure testing required? Routine pressure testing: Conduct routine pressure testing on key projects during the grayscale period of the version; Pre launch pressure testing for new projects/interfaces: Pre launch pressure testing for new projects to guide capacity estimation and current limiting configuration.

Interface dial test
In Fig. 4, for backend interface services, this study is more concerned about whether the data returned by the online interface is correct and whether the new code has affected the existing logic? Interface testing can effectively cover the above two issues. On the one hand, it can be used for verifying the data returned by online interfaces and detecting interface anomalies in real-time. On the other hand, it can be used to automate the

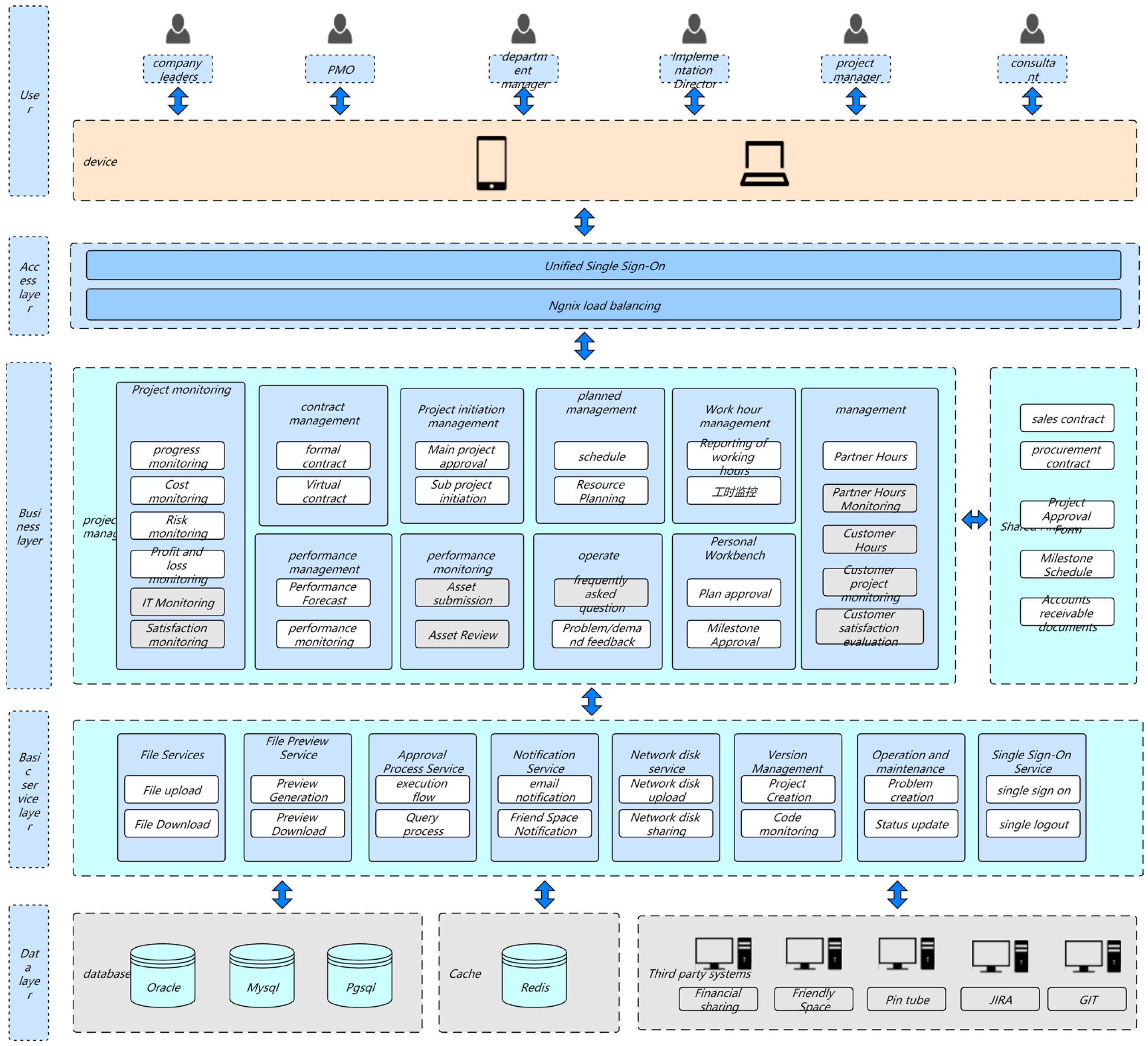


Figure 4. Full link security performance testing.

API testing, cooperate with the online process, find out whether the new code is introduced, and terminate the online operation in a timely manner. An interface testing system should have the following features: support verifying the interface return results; Complex verification supports writing scripts; Multiple test points can be set to simulate formal user request scenarios; Support for alarms; Detailed test reports; Get through with the online process, and support automated API testing after grayscale.

Operation and maintenance launch

Grayscale online mechanism

Passing code testing in a testing environment does not guarantee that the code has no problems at all, and there is still a possibility of undetected problems caused by testing omissions or environmental differences. If there is no grayscale online mechanism, these potential issues can directly go online, which may have catastrophic consequences for the online system. The grayscale launch needs to be accompanied by grayscale checks to ensure the safety of this launch. The inspection methods used in this study include: automatic interface detection for grayscale machines after grayscale is brought online; Compare the monitoring indicators of grayscale machines with those of online machines; Acceptance of grayscale environment testing function.

Online rollback

If an online problem occurs due to the launch of new code, the first thing to do should be a rollback operation, rather than analyzing the problem and fixing the code launch. The rollback operation needs to achieve the following points as much as possible: the rollback speed is fast, and if it is necessary to reverse the code, recompile and package it, and upload it to the server, the entire process will be relatively long, and the fault recovery time will also be relatively long; Rollback is complete, and some code and configuration files are dependent. If only the code is rolled back, it may cause exceptions after startup; Rollback capability requires architectural consideration to avoid situations where rollback is not possible, such as data writing after going live, which is incompatible with the rolled back code.

This study adopts the RPM packaging method for online and rollback. Package all the content (code and configuration) deployed at once in the form of RPM. During deployment, the server retains the historical version of the RPM package. When rollback is required, simply execute the Job to install the historical version of the RPM package. If the service is deployed using a container, it will be easier to roll back by simply starting the mirror of the historical version.

Online availability

System protection

In Fig. 5, online operation may face various problems, such as machine failures, network issues, dependent service provider crashes, and a surge in traffic. How to survive in such a dangerous environment? We cannot trust others, we can only improve the reliability of our services. The general methods adopted include: circuit breaker, current limiting, degradation, retry, etc.

Circuit breaker: When there is a problem with the service provider that this study relies on, the service of this study cannot be dragged down. The commonly used method is circuit breaker. When the success rate of the relying party is lower than the threshold set in this study, pause the call to it and retry after a certain interval. Commonly used fuse components include Hystrix, Sentinel, etc. The following table is a comparison between Hystrix and Sentinel. Considering that Netflix no longer maintains Hystrix and Sentinel has more diverse functions, this study chose Sentinel as the fuse component for this study[22–26].

Downgrading: When a sudden accident occurs and the overall service is overloaded, in order to avoid overall unavailability of the service, service degradation is generally enabled to ensure the normal operation of important or basic services, and non important services are delayed or suspended. Therefore, this study implemented a page degradation service. The service will periodically request the page interface and verify it. After passing the verification, the results will be saved; When the service encounters abnormalities or overload, turn on the downgrade switch, and the page service will directly read the previously saved static page data and return it. Sacrifice personalization and downgrade from thousands of people and faces to thousands of people and faces. Due to the removal of the page construction process, the processing power will be greatly improved to ensure the normal operation of the service[27–31].

In Fig. 6, current limiting: Current limiting is very important as it is an important level for protecting the normal operation of services. Especially in the event of a sudden increase in traffic, reasonable flow limiting can protect your service from being overwhelmed. If flow restriction is not configured, not only will the service be disrupted during a sudden increase in traffic, but it will also be difficult to recover because the service will be

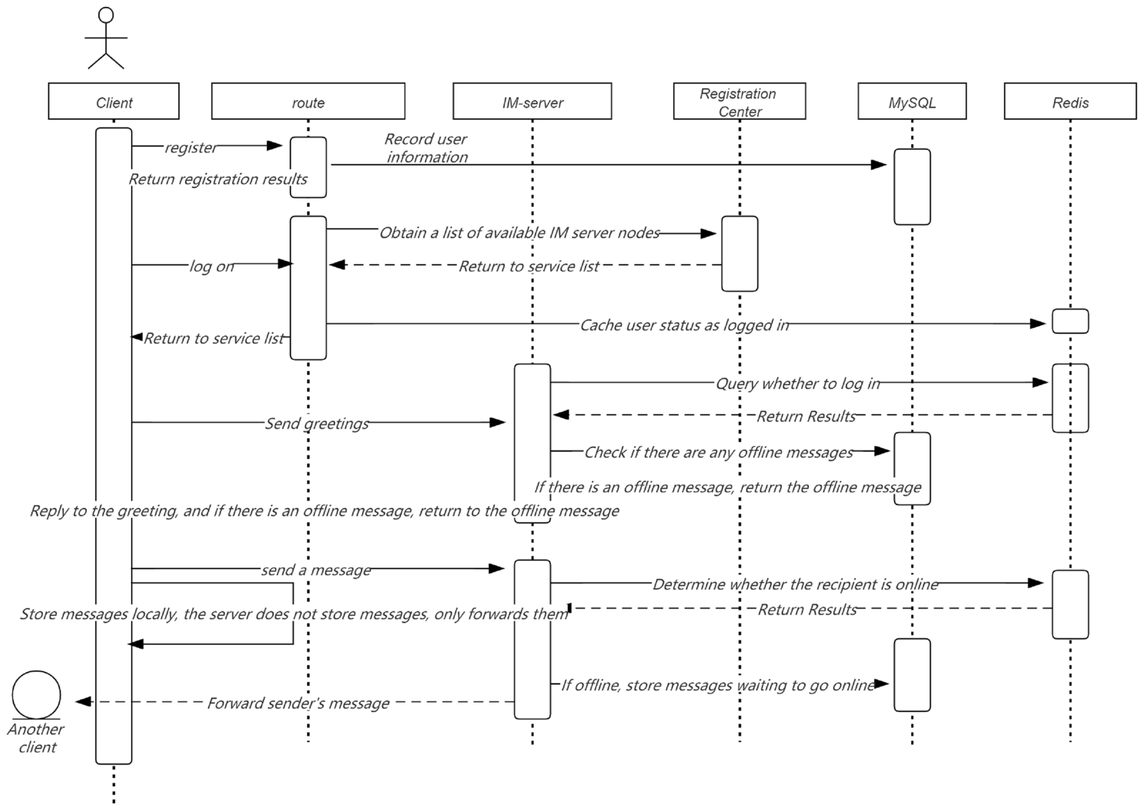

Client
route
IM-server
Registration Center
MySQL
Redis
register
Return registration results
Record user information
Obtain a list of available IM server nodes
log on
Return to service list
Cache user status as logged in
Return to service list
Query whether to log in
Return Results
Send greetings
Check if there are any offline messages
If there is an offline message, return the offline message
Reply to the greeting, and if there is an offline message, return to the offline message
send a message
Determine whether the recipient is online
Store messages locally, the server does not store messages, only forwards them
Return Results
If offline, store messages waiting to go online
Forward sender's message
Another client

Figure 5. Full link security and effective timing.

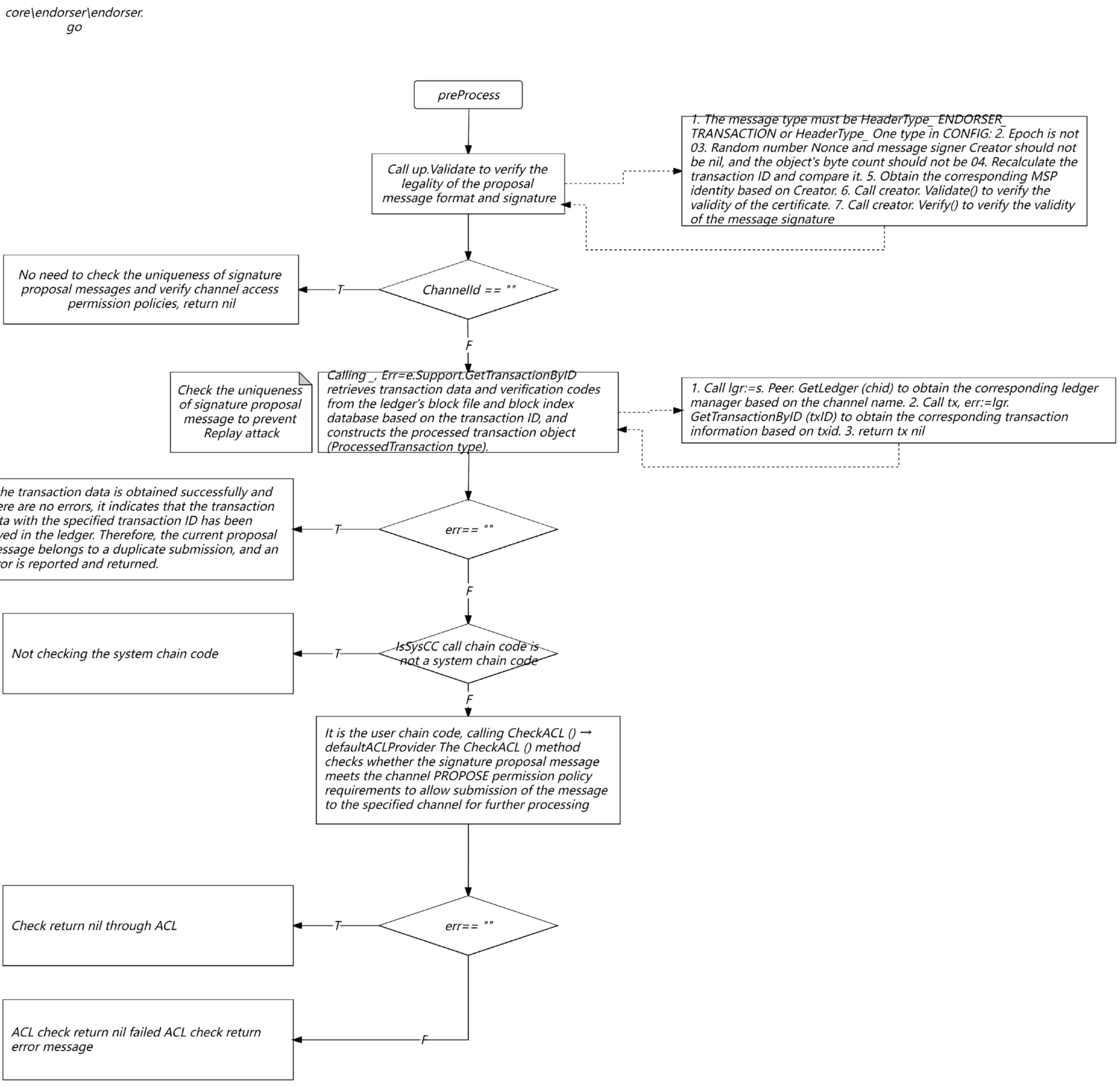

core\endorser\endorser.go
preProcess
Call up.Validate to verify the legality of the proposal message format and signature
1. The message type must be HeaderType_ ENDORSER_ TRANSACTION or HeaderType_ One type in CONFIG: 2. Epoch is not 03. Random number Nonce and message signer Creator should not be nil, and the object's byte count should not be 04. Recalculate the transaction ID and compare it. 5. Obtain the corresponding MSP identity based on Creator. 6. Call creator. Validate() to verify the validity of the certificate. 7. Call creator. Verify() to verify the validity of the message signature
ChannelId == ""
T
No need to check the uniqueness of signature proposal messages and verify channel access permission policies, return nil
F
Check the uniqueness of signature proposal message to prevent Replay attack
Calling _, Err=e.Support.GetTransactionByID retrieves transaction data and verification codes from the ledger's block file and block index database based on the transaction ID, and constructs the processed transaction object (ProcessedTransaction type).
1. Call lgr:=s. Peer. GetLedger (chid) to obtain the corresponding ledger manager based on the channel name. 2. Call tx, err:=lgr. GetTransactionByID (txID) to obtain the corresponding transaction information based on txid. 3. return tx nil
err== ""
T
If the transaction data is obtained successfully and there are no errors, it indicates that the transaction data with the specified transaction ID has been saved in the ledger. Therefore, the current proposal message belongs to a duplicate submission, and an error is reported and returned.
F
IsSysCC call chain code is not a system chain code
T
Not checking the system chain code
F
It is the user chain code, calling CheckACL () → defaultACLProvider The CheckACL () method checks whether the signature proposal message meets the channel PROPOSE permission policy requirements to allow submission of the message to the specified channel for further processing
err== ""
T
Check return nil through ACL
F
ACL check return nil failed ACL check return error message

Figure 6. Full Link System Security Routing Optimization.

disrupted again after restarting. This study generally configures current limiting at the gateway layer and service layer separately. The gateway layer can configure single machine current limiting using nginx's current limiting plugin. The service layer uses Sentinel flow limiting. Sentinel can limit traffic based on QPS, concurrency, and call relationships, and supports overall cluster traffic restriction. The following figure shows the implementation of Sentinel cluster current limiting.

In Fig. 7, Try again: From the client to the backend service, there may be exceptions in various stages, such as DNS failures, operator node failures, backend server failures, etc., resulting in request failures. For different situations, appropriate retries can increase the success rate of requests. There are several types of retries on the mainstream frontline financial industry APP end:

In Fig. 8, IP direct retry, which controls the number of retries by configuring the number of direct IPs; Super pipeline retry, the company's self-developed HTTP based gateway proxy service, which can achieve remote retry; HTTP retry; Try again with the original URL.

Retries can improve the success rate, but excessive retries may lead to excessive pressure on back-end services and Avalanche effect. So the following points should be noted in the retry strategy: the retry strategy can be cloud controlled, and in extreme cases, it can be configured in the background to turn off retry; Distinguish error codes and try again. Not all error codes need to be tried again, such as error codes returned by backend triggering current limiting, which do not need to be tried again.

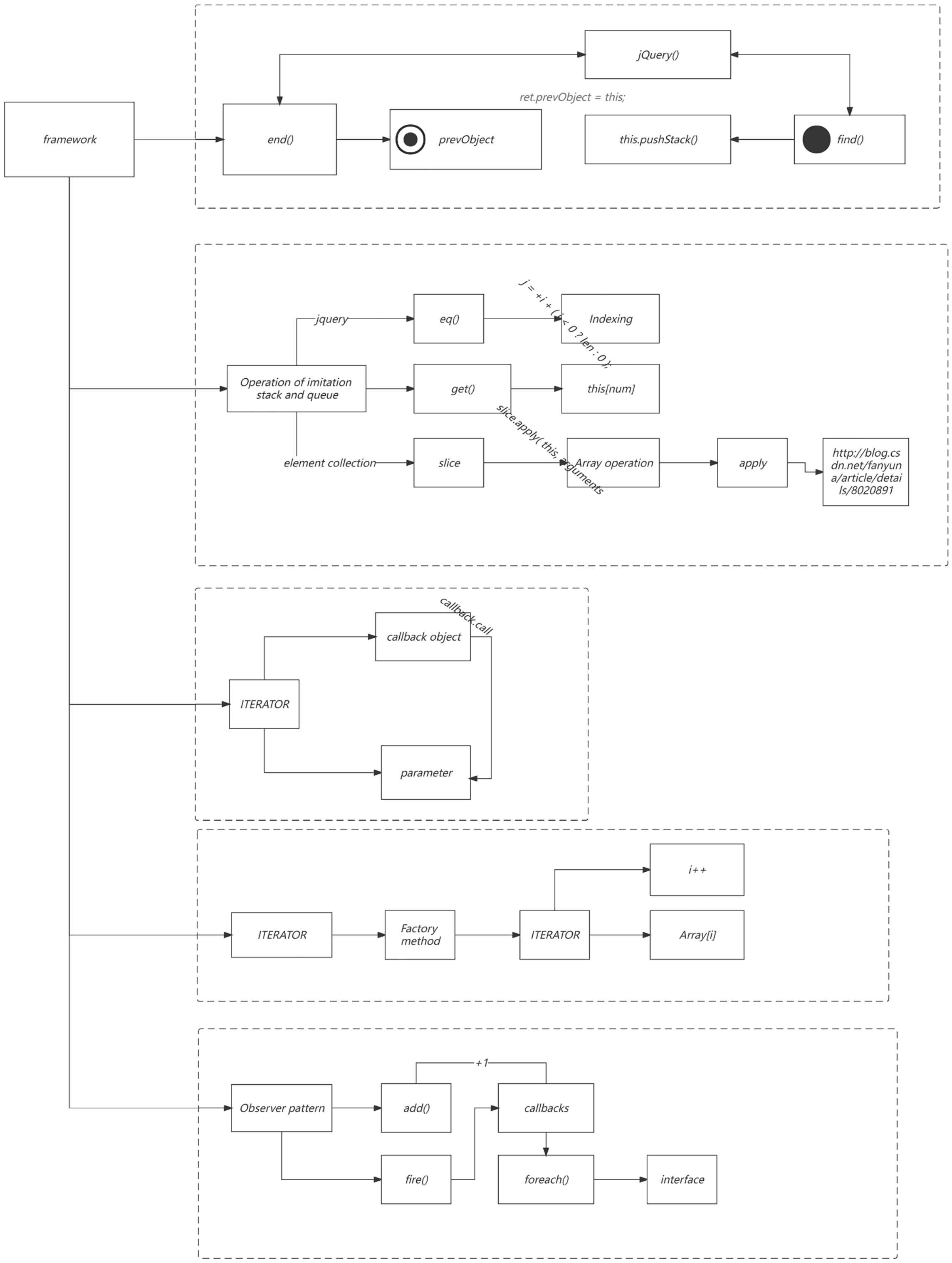
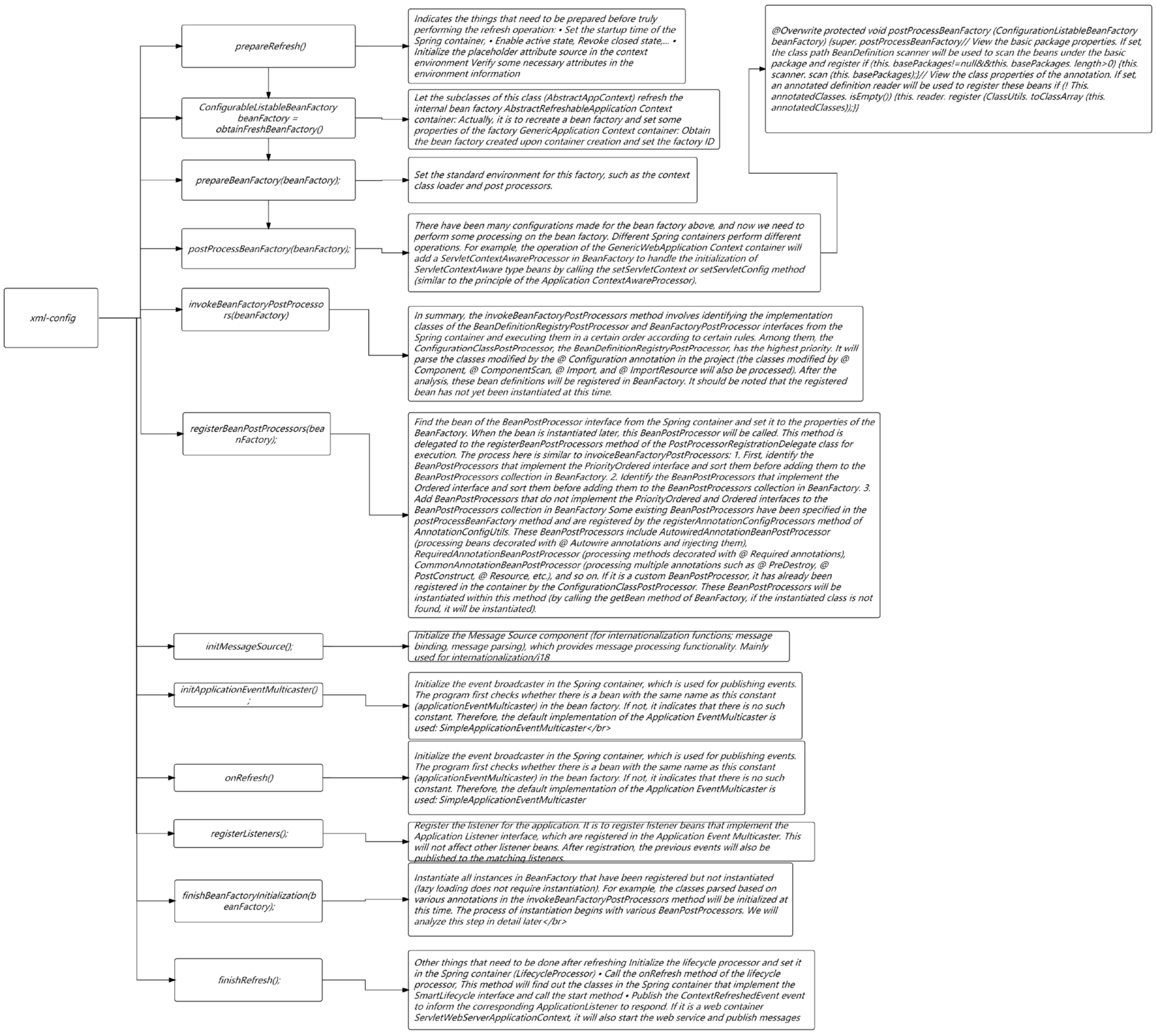


Figure 7. Core data security flow.

Data backup

In terms of database usage, this study needs to consider the high availability of the database. In this regard, the company's service cloud team has made many optimizations, which can be found in this article. This article will not be elaborated on. As a business unit, it is important to note that the use of databases should strictly follow the architecture of two locations and three centers for application and use.


xml-config
prepareRefresh()
ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory()
prepareBeanFactory(beanFactory);
postProcessBeanFactory(beanFactory);
invokeBeanFactoryPostProcessors(beanFactory)
registerBeanPostProcessors(beanFactory);
initMessageSource();
initApplicationEventMulticaster();
onRefresh()
registerListeners();
finishBeanFactoryInitialization(beanFactory);
finishRefresh();

Indicates the things that need to be prepared before truly performing the refresh operation: • Set the startup time of the Spring container, • Enable active state, Revoke closed state,... • Initialize the placeholder attribute source in the context environment Verify some necessary attributes in the environment information

Let the subclasses of this class (AbstractAppContext) refresh the internal bean factory AbstractRefreshableApplication Context container: Actually, it is to recreate a bean factory and set some properties of the factory GenericApplication Context container: Obtain the bean factory created upon container creation and set the factory ID

Set the standard environment for this factory, such as the context class loader and post processors.

There have been many configurations made for the bean factory above, and now we need to perform some processing on the bean factory. Different Spring containers perform different operations. For example, the operation of the GenericWebApplication Context container will add a ServletContextAwareProcessor in BeanFactory to handle the initialization of ServletContextAware type beans by calling the setServletContext or setServletConfig method (similar to the principle of the Application ContextAwareProcessor).

@Overwrite protected void postProcessBeanFactory (ConfigurationListableBeanFactory beanFactory) {super. postProcessBeanFactory// View the basic package properties. If set, the class path BeanDefinition scanner will be used to scan the beans under the basic package and register if (this. basePackages!=null&&this. basePackages. length>0) {this. scanner. scan (this. basePackages);}// View the class properties of the annotation. If set, an annotated definition reader will be used to register these beans if (! This. annotatedClasses. isEmpty()) {this. reader. register (ClassUtils. toClassArray (this. annotatedClasses));}}

In summary, the invokeBeanFactoryPostProcessors method involves identifying the implementation classes of the BeanDefinitionRegistryPostProcessor and BeanFactoryPostProcessor interfaces from the Spring container and executing them in a certain order according to certain rules. Among them, the ConfigurationClassPostProcessor, the BeanDefinitionRegistryPostProcessor, has the highest priority. It will parse the classes modified by the @ Configuration annotation in the project (the classes modified by @ Component, @ ComponentScan, @ Import, and @ ImportResource will also be processed). After the analysis, these bean definitions will be registered in BeanFactory. It should be noted that the registered bean has not yet been instantiated at this time.

Find the bean of the BeanPostProcessor interface from the Spring container and set it to the properties of the BeanFactory. When the bean is instantiated later, this BeanPostProcessor will be called. This method is delegated to the registerBeanPostProcessors method of the PostProcessorRegistrationDelegate class for execution. The process here is similar to invoiceBeanFactoryPostProcessors: 1. First, identify the BeanPostProcessors that implement the PriorityOrdered interface and sort them before adding them to the BeanPostProcessors collection in BeanFactory. 2. Identify the BeanPostProcessors that implement the Ordered interface and sort them before adding them to the BeanPostProcessors collection in BeanFactory. 3. Add BeanPostProcessors that do not implement the PriorityOrdered and Ordered interfaces to the BeanPostProcessors collection in BeanFactory Some existing BeanPostProcessors have been specified in the postProcessBeanFactory method and are registered by the registerAnnotationConfigProcessors method of AnnotationConfigUtils. These BeanPostProcessors include AutowiredAnnotationBeanPostProcessor (processing beans decorated with @ Autowire annotations and injecting them), RequiredAnnotationBeanPostProcessor (processing methods decorated with @ Required annotations), CommonAnnotationBeanPostProcessor (processing multiple annotations such as @ PreDestroy, @ PostConstruct, @ Resource, etc.), and so on. If it is a custom BeanPostProcessor, it has already been registered in the container by the ConfigurationClassPostProcessor. These BeanPostProcessors will be instantiated within this method (by calling the getBean method of BeanFactory, if the instantiated class is not found, it will be instantiated).

Initialize the Message Source component (for internationalization functions; message binding, message parsing), which provides message processing functionality. Mainly used for internationalization/i18

Initialize the event broadcaster in the Spring container, which is used for publishing events. The program first checks whether there is a bean with the same name as this constant (applicationEventMulticaster) in the bean factory. If not, it indicates that there is no such constant. Therefore, the default implementation of the Application EventMulticaster is used: SimpleApplicationEventMulticaster</br>

Initialize the event broadcaster in the Spring container, which is used for publishing events. The program first checks whether there is a bean with the same name as this constant (applicationEventMulticaster) in the bean factory. If not, it indicates that there is no such constant. Therefore, the default implementation of the Application EventMulticaster is used: SimpleApplicationEventMulticaster

Register the listener for the application. It is to register listener beans that implement the Application Listener interface, which are registered in the Application Event Multicaster. This will not affect other listener beans. After registration, the previous events will also be published to the matching listeners.

Instantiate all instances in BeanFactory that have been registered but not instantiated (lazy loading does not require instantiation). For example, the classes parsed based on various annotations in the invokeBeanFactoryPostProcessors method will be initialized at this time. The process of instantiation begins with various BeanPostProcessors. We will analyze this step in detail later</br>

Other things that need to be done after refreshing Initialize the lifecycle processor and set it in the Spring container (LifecycleProcessor) • Call the onRefresh method of the lifecycle processor, This method will find out the classes in the Spring container that implement the SmartLifecycle interface and call the start method • Publish the ContextRefreshedEvent event to inform the corresponding ApplicationListener to respond. If it is a web container ServletWebServerApplicationContext, it will also start the web service and publish messages

Figure 8. Time series flow native distribution mechanism.

Monitoring alarms

A comprehensive monitoring and alarm system can enable this study to detect and handle service issues in a timely manner, avoiding their spread. From different latitudes, this study requires different monitoring, mainly including:

Indicator monitoring; The key indicators of monitoring services include three parts: machine indicators (CPU, memory, network traffic, etc.), service indicators (QPS, success rate, response time, etc.), and third-party interface indicators (QPS, success rate, response time, etc.).

The specific implementation plan is to report indicators through Collected and self-developed Meerkat components, submit them to Graphint for summary and storage, and display them through Grafana.

Application log monitoring: Based on application log monitoring, this study can discover more fine-grained exceptions, such as exceptions thrown within the application, internal business logic errors, etc. Firstly, this study defines a format specification for log printing, where all logs that need to be monitored are output in a unified format. Then, through the log collection component Venus, the logs on the machine are delivered to the Kafka queue, and finally entered into the Druid temporal database for multi-dimensional analysis and storage. The front-end report is presented using Graphna[17–20].

Full link monitoring: If you want to monitor contextual link relationships, cross system fault localization, and other related issues, you need to perform full link monitoring. For detailed information, please refer to the article "Exploration and Practice of Full Link Automated Monitoring Platforms in the Financial Field of Mainstream First tier Large Factories" in this article.

Emergency plan
Although a series of availability optimization measures were listed above in this study, various anomalies and failures still occur in real online environments, requiring manual handling in this study. When a problem truly arises, if you go to the scene to think about how to handle it, you may not have achieved the optimal solution due to panic. Therefore, this study needs to anticipate potential issues and how to handle them in advance, and form a written document. When problems arise, this study only needs to process the document step by step[21].

Service quality inspection

With an optimization plan in place, how can I determine if the service has been implemented according to the optimization plan? How to ensure high long-term availability of services? This study does not want to complete a round of optimization. After the service maturity is improved, if there is no follow-up for a long time, the service maturity will gradually begin to decline.

Therefore, automated checks were conducted on these common service maturity indicators, and daily health reports were generated to monitor the maturity of services. Once there is an abnormality in a certain indicator, this study can handle it in a timely manner. This keeps the service maturity of this study in a long-term stable state.

Prototype experiment

Modeling prototype:

$$\min f_1 = \sum_{j=1}^{n} \omega_j * T_j$$

$$\min f_2 = 10 * \log \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{p=1}^{n}\sum_{v=1}^{u_i} 10^{0.1*PN_{iv}} * \frac{l_j}{V_{iv}} * x_{jipv}}{\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{p=1}^{n}\sum_{v=1}^{\mathrm{u}_i} \frac{l_j}{V_{iv}} * x_{jipv}}$$

$$\min F(x) = [f_1(x), f_2(x), ..., f_m(x)]$$

$$s.t.\begin{cases} g(x) \leq 0, i = 1, 2, ..., p \\ h(x) = 0, j = 1, 2, ..., q \end{cases}$$

As shown in Fig. 9, the native temporal and global architecture of the metaverse full link is shown in the figure. From the UE end to the firewall, the load balancing agent optimizes multi-dimensionally, accesses the port port and IP whitelist based on REST, and performs multi-dimensional DDD partitioning. At the same time, the upper and lower boundaries are clearly defined, including third-party services and configuration file collections. The architecture is deployed in the form of multiple nodes, multiple live in different places, and dual live in different places, The database adopts a hybrid approach of static and dynamic multidimensional filtering for persistence. The specific network protocol, data direction, logical architecture, and parameter configuration are shown in the figure. Containerization: To achieve environmental consistency and rapid deployment of cloud native architecture, containerization technology is used to package applications and their dependencies into independent containers, achieving environmental consistency and rapid deployment. Containerization enables applications to maintain consistency in different environments, and containers can be quickly deployed and started, accelerating application delivery speed.Metaverse Cloud Native Architecture: To achieve flexibility and scalability, the cloud native architecture adopts a microservice architecture that splits applications into small, autonomous microservices. The microservice architecture enables different parts of an application to be independently developed, deployed, and extended, achieving loose coupling and flexibility. Each microservice can independently scale according to demand, providing better ability to cope with high loads and peak traffic.

Continuous delivery and integration: Achieve rapid feedback and delivery. The cloud native architecture supports the practice of continuous delivery and integration, and achieves rapid feedback and delivery through automated construction, testing, and deployment processes. Continuous integration ensures that every code submission undergoes automated testing, improving software quality and stability. Continuous delivery achieves rapid delivery through automated processes, meeting the rapid changes in market demand and user feedback.

Automated management: Realize elastic scaling and fault recovery with cloud native architecture. Utilize automated management tools to achieve elastic scaling and fault recovery. Automated scaling automatically adjusts the number of application instances based on load changes, ensuring system resilience and high availability. Automated management tools can automatically configure and manage various components of an application, reducing manual operations and errors. At the same time, monitoring and automated fault detection tools can timely detect and handle application failures, and quickly recover.

As shown in Fig. 10, the database deployment architecture is shown in the figure, with hierarchical deployment persistence. In the configuration database, the binlog is divided into databases, tables, and mapped table fields. The parameters are dynamically transmitted to the application server and traffic model. The data link with the lower layer is read and written based on socket rest, and the DEV is not delivered in one click mode. The portal is responsible for attribute partitioning on the interface side and preprocessing of parameter proxies, The configuration server is responsible for encapsulating and distributing user role attributes and related business operations.

Distributed transactions are usually run on machines with multiple nodes, and there is a process of RPC during runtime. Compared to a single system database, there are more abnormal links in ensuring the atomicity of transactions, and there are more recovery and rollback situations. Currently, most of them are solved through

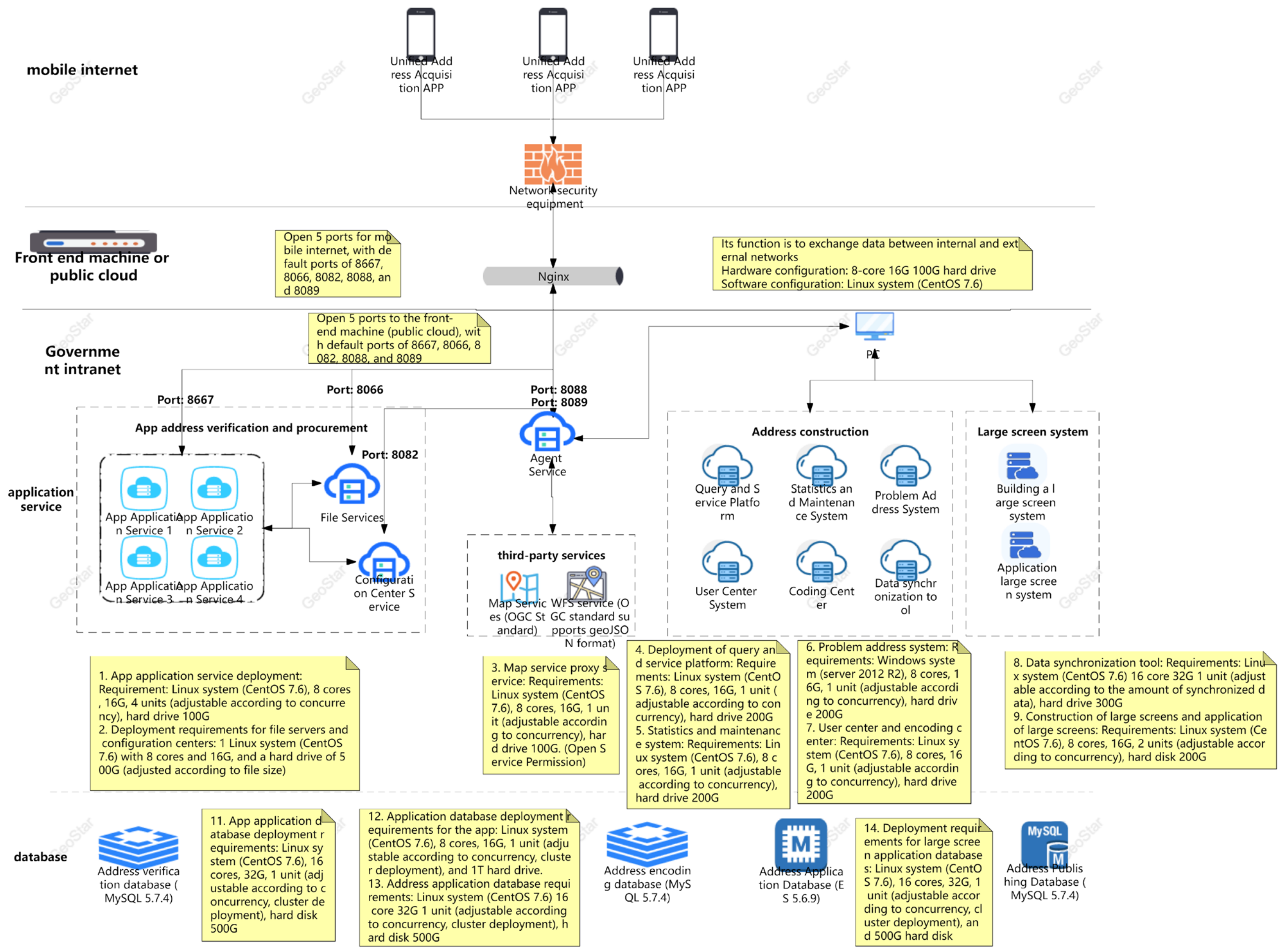

mobile internet
Unified Address Acquisition APP
Unified Address Acquisition APP
Unified Address Acquisition APP
Network security equipment
Front end machine or public cloud
Open 5 ports for mobile internet, with default ports of 8667, 8066, 8082, 8088, and 8089
Nginx
Its function is to exchange data between internal and external networks
Hardware configuration: 8-core 16G 100G hard drive
Software configuration: Linux system (CentOS 7.6)
Government intranet
Open 5 ports to the front-end machine (public cloud), with default ports of 8667, 8066, 8082, 8088, and 8089
PC
Port: 8667
Port: 8066
Port: 8088
Port: 8089
application service
App address verification and procurement
Port: 8082
App Application Service 1
App Application Service 2
App Application Service 3
App Application Service 4
File Services
Configuration Center Service
Agent Service
third-party services
Map Services (OGC Standard)
WFS service (OGC standard supports geoJSON format)
Address construction
Query and Service Platform
Statistics and Maintenance System
Problem Address System
User Center System
Coding Center
Data synchronization tool
Large screen system
Building a large screen system
Application large screen system
1. App application service deployment: Requirement: Linux system (CentOS 7.6), 8 cores, 16G, 4 units (adjustable according to concurrency), hard drive 100G
2. Deployment requirements for file servers and configuration centers: 1 Linux system (CentOS 7.6) with 8 cores and 16G, and a hard drive of 500G (adjusted according to file size)
3. Map service proxy service: Requirements: Linux system (CentOS 7.6), 8 cores, 16G, 1 unit (adjustable according to concurrency), hard drive 100G. (Open Service Permission)
4. Deployment of query and service platform: Requirements: Linux system (CentOS 7.6), 8 cores, 16G, 1 unit (adjustable according to concurrency), hard drive 200G
5. Statistics and maintenance system: Requirements: Linux system (CentOS 7.6), 8 cores, 16G, 1 unit (adjustable according to concurrency), hard drive 200G
6. Problem address system: Requirements: Windows system (server 2012 R2), 8 cores, 16G, 1 unit (adjustable according to concurrency), hard drive 200G
7. User center and encoding center: Requirements: Linux system (CentOS 7.6), 8 cores, 16G, 1 unit (adjustable according to concurrency), hard drive 200G
8. Data synchronization tool: Requirements: Linux system (CentOS 7.6) 16 core 32G 1 unit (adjustable according to the amount of synchronized data), hard drive 300G
9. Construction of large screens and application of large screens: Requirements: Linux system (CentOS 7.6), 8 cores, 16G, 2 units (adjustable according to concurrency), hard disk 200G
database
Address verification database (MySQL 5.7.4)
11. App application database deployment requirements: Linux system (CentOS 7.6), 16 cores, 32G, 1 unit (adjustable according to concurrency, cluster deployment), hard disk 500G
12. Application database deployment requirements for the app: Linux system (CentOS 7.6), 8 cores, 16G, 1 unit (adjustable according to concurrency, cluster deployment), and 1T hard drive.
13. Address application database requirements: Linux system (CentOS 7.6) 16 core 32G 1 unit (adjustable according to concurrency, cluster deployment), hard disk 500G
Address encoding database (MySQL 5.7.4)
Address Application Database (ES 5.6.9)
14. Deployment requirements for large screen application databases: Linux system (CentOS 7.6), 16 cores, 32G, 1 unit (adjustable according to concurrency, cluster deployment), and 500G hard disk
Address Publishing Database (MySQL 5.7.4)
GeoStar

Figure 9. Full Link Technical Architecture for Production Environment.

two-stage submission protocols. When a distributed transaction is submitted to multiple servers for processing, each server needs to record logs. When a transaction fails, the corresponding server needs to perform a rollback operation, which is a typical distributed transaction processing process.

Upgrading from "two places and three centers" to "three places and five centers" is a major upgrade of infrastructure, not just an increase in simple data replicas. The architectural improvements it brings include the following points.

(1) The database has urban level disaster recovery capabilities.
(2) The application has urban level disaster tolerance capability, and the deployment of the application can achieve a dual center model of City 1 and City 2, enhancing the disaster tolerance capability of the application.
(3) The capacity of the database has increased, the number of read-only replicas has increased, and service capabilities have been enhanced.

This upgrade also introduces new challenges to the original data center architecture.

(1) The increased time consumption brought about by cross city has an impact on batch processing of business, overall link time consumption, and hot lines.
(2) The number of data synchronization replicas has increased, and the network in the original machine room needs to be expanded.
(3) The number of data synchronization replicas has increased, and the hardware resources of the original tenants have expanded.

During the process of architecture upgrade, it is necessary to always maintain disaster recovery capability: after any single computer room fails, the cluster remains available, and in case of a computer room failure in any city other than the city 1 where the main database is located, the cluster remains available and can still provide services.

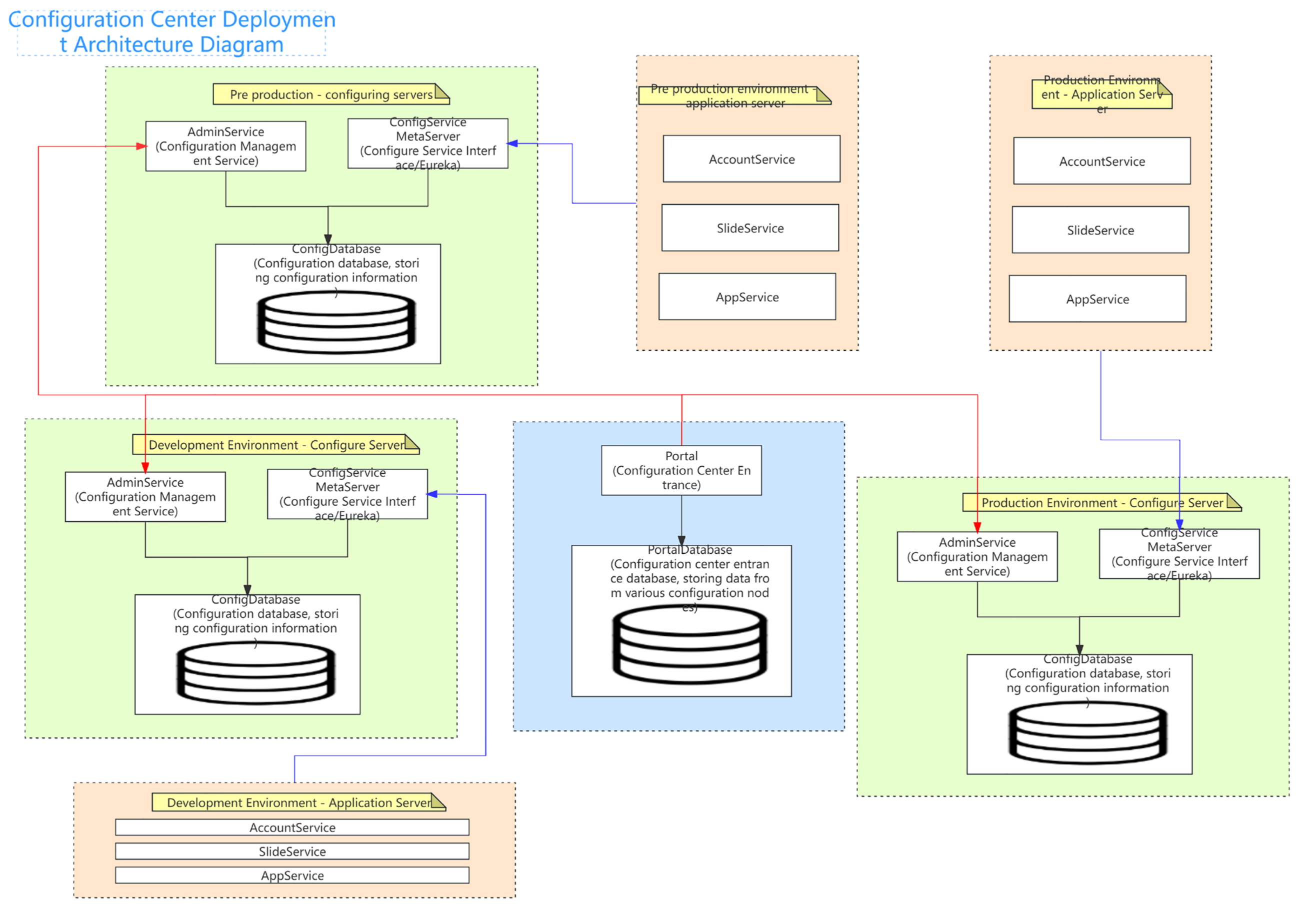

Configuration Center Deployment Architecture Diagram
Pre production - configuring servers
AdminService (Configuration Management Service)
ConfigService MetaServer (Configure Service Interface/Eureka)
ConfigDatabase (Configuration database, storing configuration information)
Pre production environment application server
AccountService
SlideService
AppService
Production Environment - Application Server
Development Environment - Configure Server
Portal (Configuration Center Entrance)
PortalDatabase (Configuration center entrance database, storing data from various configuration nodes)
Production Environment - Configure Server
Development Environment - Application Server

Figure 10. Full Link Data Storage Security Configuration.

In Fig. 11, the logic of the application side full link sorting data interface is shown in the figure. On the left is a universal platform for multidimensional authorization and standardization of heterogeneous data. For the service layer, regular matching of third-party data sources is required, especially the logical verification of API interface message headers and information entropy. On the right is the API parameters for the binding interface of the service provider platform. For other third-party applications, payment and order placement are required, Facilitate the ecological construction of the original metaverse of the financial circle link cloud.

Before submitting, the business service sends events to the event service, which only records events and does not send them. The business service notifies the event service after submitting or rolling back, and the event service sends or deletes events. Don't worry about the business system crashing after submitting or rolling over, and unable to send confirmation events to the event service, because the event service will regularly retrieve all events that have not yet been sent and query the business system, and decide whether to send or delete the event based on the return of the business system.

Although external events can decouple the business system from the event system, they also bring additional workload: external event services incur two additional network communication costs compared to local event services (before submission and after submission/rollback), and the business system also needs to provide a separate query interface for the event system to determine the status of unsent events.

Precautions for reliable event notification mode:

There are two points to note about the reliable event pattern: 1 Correct sending of events; 2. Repeated consumption of events.By using asynchronous messaging services, it is possible to ensure the correct sending of events. However, it is possible for events to be sent repeatedly. Therefore, it is necessary for the consumer to ensure that the same event is not consumed repeatedly, in short, to ensure the idempotence of event consumption.

If the event itself is a state type event with idempotence, such as notification of order status (placed, paid, shipped, etc.), the order of the events needs to be determined. Generally, it is determined through a timestamp that after consuming a new message, when receiving an old message, it is directly discarded and not consumed. If a global timestamp cannot be provided, consideration should be given to using a globally unified serial number.

For events that do not have idempotence, they are generally action behavior events, such as a deduction of 100 or a deposit of 200. The event ID and event result should be persisted, and the event ID should be queried before consuming the event. If it has already been consumed, the execution result should be returned directly; If there is a new message, execute it and store the execution result.

As show in Fig. 12, The details of the full link implementation are shown in the figure. Under the multi-dimensional mapping mechanism of each fitness multithreading, thread pooling operations are carried out,

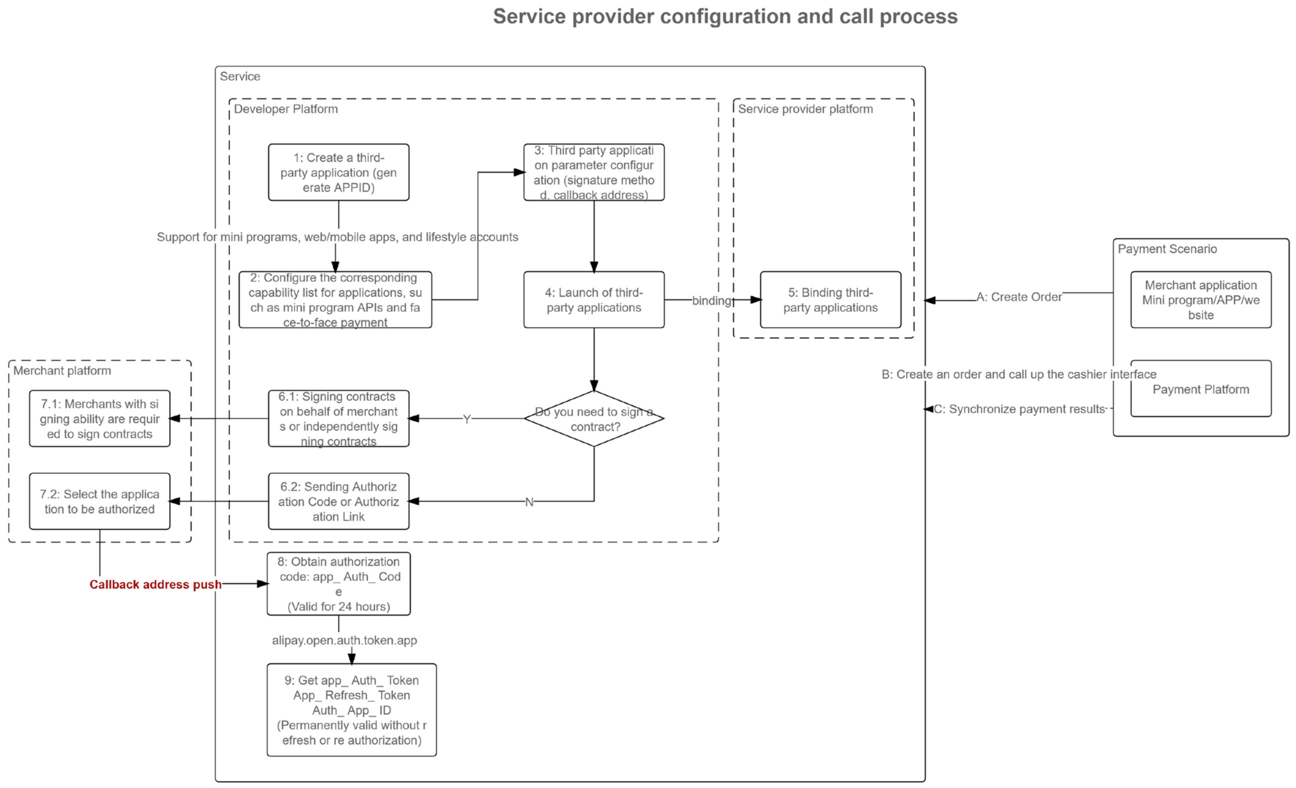

Service provider configuration and call process
Service
Developer Platform
1: Create a third-party application (generate APPID)
Support for mini programs, web/mobile apps, and lifestyle accounts
2: Configure the corresponding capability list for applications, such as mini program APIs and face-to-face payment
3: Third party application parameter configuration (signature method, callback address)
4: Launch of third-party applications
binding
Service provider platform
5: Binding third-party applications
Payment Scenario
Merchant application Mini program/APP/website
A: Create Order
B: Create an order and call up the cashier interface
Payment Platform
C: Synchronize payment results
Merchant platform
7.1: Merchants with signing ability are required to sign contracts
6.1: Signing contracts on behalf of merchants or independently signing contracts
Y
Do you need to sign a contract?
N
7.2: Select the application to be authorized
6.2: Sending Authorization Code or Authorization Link
Callback address push
8: Obtain authorization code: app_ Auth_ Code (Valid for 24 hours)
alipay.open.auth.token.app
9: Get app_ Auth_ Token App_ Refresh_ Token Auth_ App_ ID (Permanently valid without refresh or re authorization)

Figure 11. Full Link Application Architecture Parameter Transmission Details.

and the aggregation interface of local attribute categories is divided into different threshold ranges and priority settings. At the same time, multi-dimensional mapping is required for the protocol stack of multi thread program metaverse overflow, which facilitates optimization of high concurrency, high reliability, end-to-end latency, and load balancing in available scenarios.

Monitor the status of GC

Use various JVM tools to view the current logs, analyze the current JVM parameter settings, and analyze the current heap memory snapshot and GC logs. Based on the actual memory partitioning and GC execution time of each region, decide whether to optimize.

For example, some phenomena before a system crash:The time for each garbage collection is getting longer and longer, from the previous 10 ms to around 50 ms, and the time for FullGC is also extended from the previous 0.5 s to 4-5 s.

The frequency of FullGC is increasing, with the most frequent occurrence occurring less than a minute apart,The memory of the older generation is getting larger and no memory is released after each FullGC,Afterwards, the system will be unable to respond to new requests and gradually reach the critical value of OutOfMemoryError. At this point, it is necessary to analyze the JVM memory snapshot dump.

Generate a dump file for the heap

Generate the current Heap information through JMX's MBean, which is an hprof file of size 3G (the size of the entire heap). If JMX is not started, the file can be generated through Java's jmap command.

Analyze the dump file

To open this 3G heap information file, it is obvious that typical Windows systems do not have such a large amount of memory and must use highly configured Linux and several tools to open the file:

Visual VM
IBM HeapAnalyzer
Hprof tool included with JDK
Mat (a specialized static memory analysis tool for Eclipse) is recommended for use

Note: The file is too large. It is recommended to use Eclipse's specialized static memory analysis tool Mat to open the analysis.

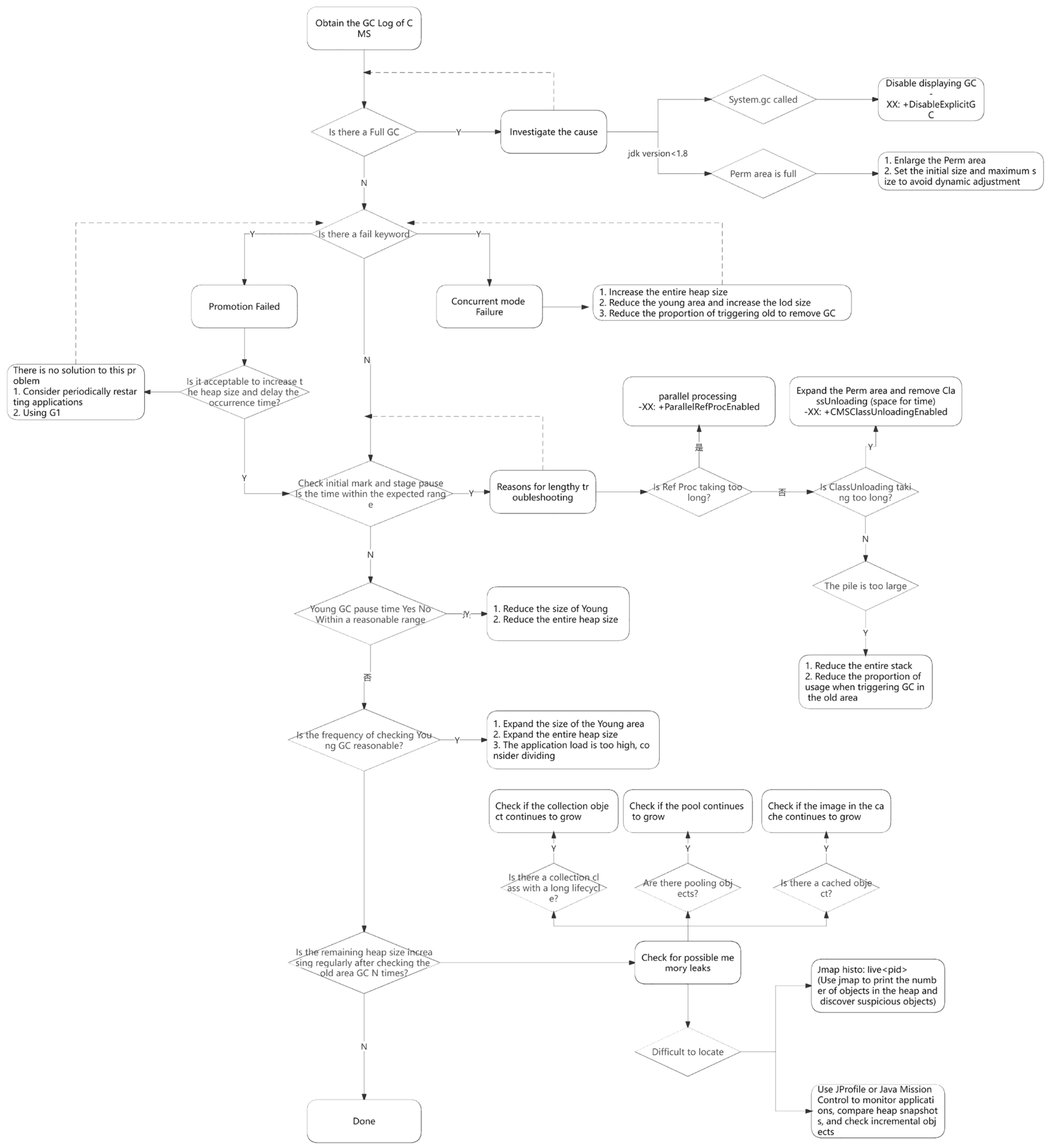

Obtain the GC Log of C MS
Is there a Full GC
Y
Investigate the cause
System.gc called
Disable displaying GC - XX: +DisableExplicitG C
jdk version<1.8
Perm area is full
1. Enlarge the Perm area
2. Set the initial size and maximum s ize to avoid dynamic adjustment
N
Is there a fail keyword
Y
Y
Promotion Failed
Concurrent mode Failure
1. Increase the entire heap size
2. Reduce the young area and increase the lod size
3. Reduce the proportion of triggering old to remove GC
There is no solution to this pr oblem
1. Consider periodically restar ting applications
2. Using G1
Is it acceptable to increase t he heap size and delay the occurrence time?
N
parallel processing -XX: +ParallelRefProcEnabled
Expand the Perm area and remove Cla ssUnloading (space for time) -XX: +CMSClassUnloadingEnabled
是
Y
Y
Check initial mark and stage pause Is the time within the expected rang e
Y
Reasons for lengthy tr oubleshooting
Is Ref Proc taking too long?
否
Is ClassUnloading taki ng too long?
N
N
The pile is too large
Young GC pause time Yes No Within a reasonable range
Y
1. Reduce the size of Young
2. Reduce the entire heap size
Y
1. Reduce the entire stack
2. Reduce the proportion of usage when triggering GC in the old area
否
Is the frequency of checking You ng GC reasonable?
Y
1. Expand the size of the Young area
2. Expand the entire heap size
3. The application load is too high, co nsider dividing
Check if the collection obje ct continues to grow
Check if the pool continues to grow
Check if the image in the ca che continues to grow
Y
Y
Y
Is there a collection cl ass with a long lifecycl e?
Are there pooling obj ects?
Is there a cached obje ct?
Is the remaining heap size increa sing regularly after checking the old area GC N times?
Check for possible me mory leaks
Jmap histo: live<pid> (Use jmap to print the numb er of objects in the heap and discover suspicious objects)
N
Difficult to locate
Use JProfile or Java Mission Control to monitor applicati ons, compare heap snapshot s, and check incremental obj ects
Done

Figure 12. Metaverse Full Link Native Multi Thread High Concurrency Response Mechanism.

Analyze the results to determine whether optimization is needed

If the parameters are set reasonably, there are no timeout logs in the system, the GC frequency is not high, and the GC time is not high, then there is no need to perform GC optimization. If the GC time exceeds 1–3 s or frequent GC, optimization is necessary.

Note: If the following indicators are met, GC is generally not required:

The execution time of Minor GC is less than 50 ms;
Minor GC execution is not frequent, approximately every 10 s;
Full GC execution time is less than 1 s;
Full GC execution frequency is not considered frequent, not less than once every 10 min.

Adjust GC type and memory allocation

If the memory allocation is too large or too small, or if the GC collector used is relatively slow, priority should be given to adjusting these parameters, and one or several machines should be first selected for beta testing. Then, the performance of the optimized machine and the unoptimized machine should be compared, and targeted final choices should be made.

Continuous analysis and adjustment

Through continuous experimentation and trial and error, analyze and find the most suitable parameters. If the most suitable parameters are found, apply these parameters to all servers.

As can be seen from Fig. 13, MC block represents the optimized Microservices security mechanism framework proposed in this paper. In the QPS range of 100, the energy consumption is lower than that of other typical algorithms, because in the low concurrency scenario, the gateway side and link tracking module have low space and time in the process of api encryption and decryption, and the input transcoding rate for the fitness function is low, so there is some performance improvement. However, with the increase of the workload, Especially in the interval [200500], the energy consumption of the optimization framework proposed in this article increases because the task volume in this interval needs to be further decomposed into itasks. For other algorithm frameworks, there is no need for transcoding point frame mapping mechanism, so the performance of other algorithms has advantages. However, as the level of data increases, the advantage of the multi-objective optimization framework for full link energy consumption performance proposed in this article is more obvious, with an average improvement of 5.38%

From Fig. 14, it can be seen that within the [100150] interval, the proposed optimization algorithm has achieved a 3.8% improvement in local search and global convergence, reducing the load balance throughout the entire lifecycle. However, within the [200800] interval, the loss is relatively high due to the large number of task blocks in this segment and the diversification of structured semi structured data attributes, resulting in triggered multithreading issues in components such as fuse reduction in the registry, Load balancing requires re detecting the information entropy of each node based on the perceptual balancing algorithm. However, within the [8001400] interval, load balancing has gradually improved, with an improvement rate of 1.82%. In high concurrency scenarios, in addition to aggregation on the gateway side, the security filtering mechanisms for circuit breaker reduction, task arrangement, and cluster deployment stages are all placed in the multi-objective function optimization system, The number of invalid threads and coroutines is within the configured Thread pool range.

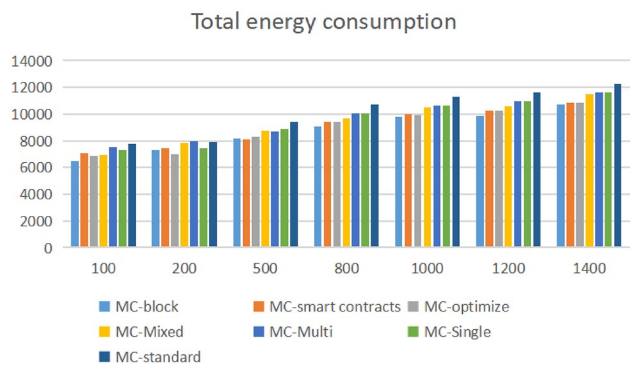


Figure 13. Total energy consumption.

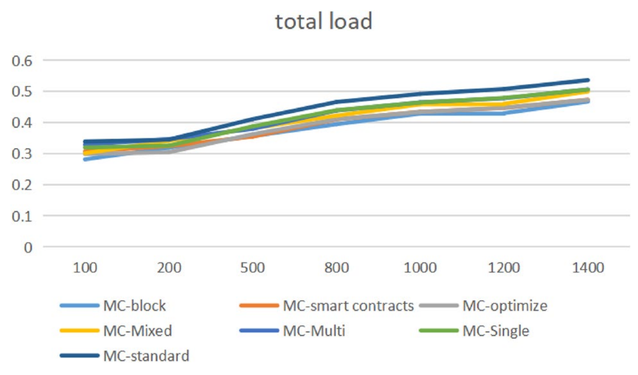


Figure 14. Full-link load balance.

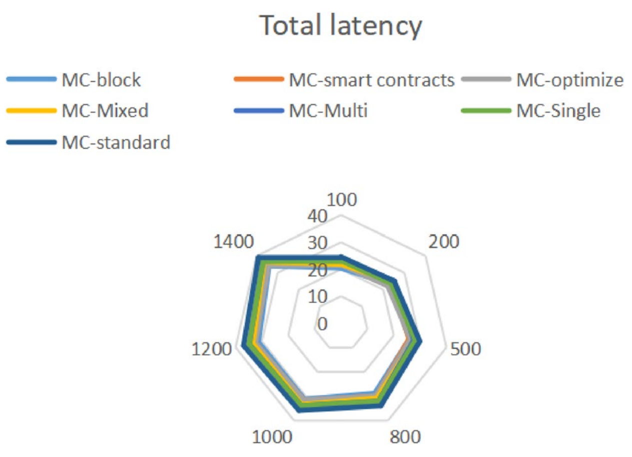


Figure 15. Full link delay.

From Fig. 15, it can be seen that the latency and MC block improvement of the entire link are all in the lower 7th level, because components with high latency dependency are safe detection mechanisms, and the reconstruction mechanism of the detection mechanism is defined in the custom definition of multi-objective functions. The quantization factor of multi-objective subfunctions is also the standard for quantifying the topology construction in this article, such as link tracking and traffic detection component modules. Therefore, there is also a filtering mechanism for global interception, The adaptation of the proxy mechanism is included in the encapsulated security proxy detection pool. Overall increase of 3.12%.

Conclusions

We introduces the definition of a service maturity model proposed by mainstream frontline large corporations in the financial field, which evaluates the maturity of services using quantifiable indicators. Then, based on the model as a standard, the backend services were evaluated and the indicators with generally lower scores for each backend service were summarized. We have sorted out optimization plans for these low sub items. And an automated service quality detection mechanism has been established to ensure long-term stability of service maturity. Through availability optimization, the maturity model scores of multiple important services have been raised to over 8 points. Through an automated service quality detection mechanism, various indicators of maturity have been monitored for a long time. Once some indicators are found to be abnormal, they should be repaired and improved in a timely manner. In the future, continuous improvements and enhancements will be made in the following two areas: automation of maturity model scoring. Avoid subjective factors of manual grading while improving efficiency; The automated service quality detection mechanism adds more indicators. I hope to continuously improve the service availability of this study through optimization based on the service maturity model.

This paper proposes a multi-objective optimization Microservices framework that takes into account the security mechanism, Define the fitness function, define the upper and lower limits, and perform multi-dimensional constraints to filter for global and local optima. At the same time, reconstruct the circuit breaker current limiting mechanism, dynamically detect protocol parameters, and perceive the logical relationship of heartbeat status in real-time. Experimental results have shown that this innovative framework can solve scenarios such as high concurrency, high reliability, and high availability Performance bottleneck, business degradation, especially in financial and securities scenarios.

Data availability

The dataset generated and/or analyzed during the current research period is not publicly available, but can be obtained from corresponding authors upon reasonable request.

Received: 10 July 2023; Accepted: 29 November 2023
Published online: 12 December 2023

References

1. Ritterbusch, G. D. & Teichmann, M. Defining the metaverse: A systematic literature review. *IEEE Access* **11**, 12368–12377 (2023).
2. Lv, Y., Shi, W., Zhang, W., Hui, Lu. & Tian, Z. Do not trust the clouds easily: The insecurity of content security policy based on object storage. *IEEE Internet Things J.* **10**(12), 10462–10470 (2023).
3. Wang, Z. *et al.* A covert channel over blockchain based on label tree without long waiting times. *Comput. Networks* **232**, 109843 (2023).
4. Newman, S. A. Building microservices: Designing fine-grained systems. (2015).
5. Newman, S. *Building Microservices* (O'Reilly Media, Inc., 2015).
6. Thones, J. Microservices. *Softw. IEEE* **32**(1), 116–116. https://doi.org/10.1109/MS.2015.11 (2015).
7. Dineva K., & Atanasova, T. Architectural ML Framework for IoT Services Delivery Based on Microservices. (2021).

8. Li, J. *et al.* A novel method for identifying microservices by considering quality expectations and deployment constraints. *Int. J. Softw. Eng. Knowl. Eng.* **3**, 32. https://doi.org/10.1142/S021819402250019X (2022).
9. Hassan, S., Bahsoon, R. & Buyya, R. Systematic scalability analysis for microservices granularity adaptation design decisions. *Softw. Pract. Exp.* **52**(6), 1378–1401. https://doi.org/10.1002/spe.3069 (2022).
10. Liu, H. *et al.* BlueNavi: A microservices architecture-styled platform providing maritime information. *Sustainability* **14**, 2173 (2022).
11. Das, A. & Chakraborty, S. Where do all my smart home data go? Context-aware data generation and forwarding for edge-based microservices over shared IoT infrastructure. *Future Gener. Comput. Syst.* **134**, 204 (2022).
12. Stavrinides, G. L. & Karatza, H. D. Containerization, microservices and serverless cloud computing: Modeling and simulation. *Simul. Model. Pract. Theory Int. J. Feder. Eur. Simul. Soc.* **118**, 102551 (2022).
13. Menhour, H. *et al.* Searchable Turkish OCRed historical newspaper collection 1928–1942. *J. Inf. Sci.* **49**(2), 335–347. https://doi.org/10.1177/01655515211000642 (2023).
14. Khoshnevis, S. A search-based identification of variable microservices for enterprise SaaS. 中国计算机科学前沿 **17**(3), 173208. https://doi.org/10.1007/s11704-022-1390-4 (2023).
15. Schabbink, J. ASSIST-IoT: A modular implementation of a reference architecture for the next generation Internet of Things. *Electronics* https://doi.org/10.3390/electronics12040854 (2023).
16. De Castro L. F. S., & Rigo, S. Relating edge computing and microservices by means of architecture approaches and features, orchestration, choreography, and offloading: A systematic literature review. https://doi.org/10.48550/arXiv.2301.07803 (2023).
17. Yao K., Ni, H. X., Wang, Y., *et al.* Low cost and high concurrency ID maker in distributed environment. https://doi.org/10.1051/itmconf/20171203003. (2017).
18. Peng, J., Cai, K. & Jin, X. High concurrency massive data collection algorithm for IoMT applications. *Comput. Commun.* https://doi.org/10.1016/j.comcom.2020.04.045 (2020).
19. Huang, W. H. High-concurrency custom-build relational database system's design and SQL parser design based on turing-complete automata. *Comput. Sci.* https://doi.org/10.48550/arXiv.2008.04640 (2020).
20. Mathur, R. Index contention under high concurrency in a database system:US16858856. US20210334273A1 [2023-07-04].
21. Guo, Y. Low-latency and high-concurrency 5G wireless sensor network assists innovation in ideological and political education in colleges and universities. *Hindawi Lim.* https://doi.org/10.1155/2021/9867800 (2021).
22. Sun Q .HIGH-CONCURRENCY QUERY METHOD, INTELLIGENT TERMINAL AND STORAGE MEDIUM:WO2020CN134276[P].WO2022062184A1[2023–07–04].
23. Chi, X., Liu, B., Niu, Q., *et al.* Web load balance and cache optimization design based Nginx under high-concurrency environment. In: Third International Conference on Digital Manufacturing & Automation. IEEE, https://doi.org/10.1109/ICDMA.2012.241. (2012).
24. Min-Xia, Z. The design and implementation of B2B business systems based on high-concurrency and mass data. *J. Henan Univ. Technol.* (2009).
25. Galdámez, P., Muoz-Escoí, F. D. & Bernabéu-Aubán, J. M. *HIDRA: Architecture and High Availability Support[J].*(2022).
26. Hirano, Y., Satoh, T. & Miura, F. Improved extendible hashing with high concurrency. *Syst. Comput. Jpn.* https://doi.org/10.1002/scj.4690261301 (1995).
27. Johnson, T., Maugis L. Two approaches for high concurrency in multicast-based object replication. (1995).
28. Youyi, Z. Design and application of main memory database in high-concurrency cluster monitoring system. *Comput. Appl. Softw.* https://doi.org/10.4028/www.scientific.net/AMR.154-155.87 (2011).
29. Gerard, S., Mraidha, C., Terrier, F., *et al.* A UML-based concept for high concurrency: the real-time object. In *7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2004)*, 12–14 May 2004, Vienna, Austria. IEEE, https://doi.org/10.1109/ISORC.2004.1300330 (2004)
30. Behren, R. V., Condit, J., & Brewer, A. E. Why events are a bad idea (for high-concurrency servers). Hotos.2003.DOI:EP2326143 B1. (2003).
31. Mohan, C. & Levine, F. ARIES/IM: An efficient and high concurrency index management method using write-ahead logging. *Acm Sigmod Record* **21**(2), 371–380. https://doi.org/10.1145/130283.130338 (1992).

Author contributions
S.S. is the first person in charge of the project and the only author.

Competing interests
The author declares no competing interests.

Additional information
Correspondence and requests for materials should be addressed to S.S.

Reprints and permissions information is available at www.nature.com/reprints.