Technical Architecture Diagram Description

1.Architecture Design Objectives

➢Architecture design goals must meet functional and non-functional requirements.

➢Maximize to use mature and open source technology to reduce costs.

➢Architecture must be simple, security upgrade extension, and scalability.

➢Define the scope of the system architecture completely and accurately. Define the boundaries between the internal and external systems and the interaction agreements clearly.

➢Determine the specific responsibilities of the subsystems, interaction protocols and procedures between subsystems.

➢Determine the development, deployment, and operation systems and corresponding specifications.

➢Provide principles, foundations, and specifications for system design and subsequent coding, testing, and maintenance.

2. Architecture Design Principles

➢Meet functional requirements and non-functional requirements. This is the most basic requirement of a software system and the most basic principle to follow when designing an architecture.

➢The Principles of Practicality. Each software system must be practical when it delivered to the user. It should solve the actual problems of the school, and the system must be stable and reliable with very good performance.

➢The Principle of Reusability. Standard modules are designed as reusable middleware or standard components to maximize developer productivity.

➢The Principle of Single Responsibility. Each subsystem focuses on completing a specific function without cross-coupling, and the systems interact through well-designed interfaces.

➢The Principle of Extensibility. Applying the idea of service orientation, the business module subsystem can afford the external interface by providing services, and the service can be extended directly in the future. The system can be flexibly deployed according to the size of the business volume which can be centralized or distributed.

3. Detailed instructions

3.1. APP/ Web and server communicate with Https

1) Authentication Server.

A trusted CA Authentication Certificate will be built into the client. A server certificate which issued by the CA agency authentication at the first stage. If the CA organization which authenticates the server certificate exists in the client list of trusted CA institutions, and the information in the server certificate and the network currently being accessed the station (domain name) is consistent as well, the client will think the server is trusted and get the server

public key from the server certificate for the subsequent process. Otherwise, the client will not continue to initiate the request.

2) Negotiate Session Key.

After authenticating the server and obtaining the public key of the server, the client will use the public key to encrypt communication with the server and negotiate two session keys, which are the client session keys used to encrypt the data sent by the client to the server. The server-side session key used to encrypt data sent by the server to the client. The reason why two symmetric keys should be negotiated under the premise is that the public key of the existing server can encrypt communication, is that asymmetric encryption is more complicated. In the process of data transmission, the use of symmetric encryption can save computing resources. In addition, the session key is randomly generated, each negotiation will have different results, therefore, the security is relatively high.

3) Encrypted Communication.

Both the client and server have the session key of this communication, and all the transmitted Http data are encrypted by the session key. In this way, other users on the network will be difficult to steal and tamper with the transmitted data between the client and server, so it can ensure the privacy and integrity.

3.2. The client uses the certificate locking technology in order to prevent the client certificate from being forged

If a malicious certificate is installed in the user undefined mobile phone, the client can eavesdrop on the user undefined communication and modify the data in request or response by a man-in-the-middle attack.

Mobile phone man-in-the-middle attack process:

1) When the client starts, it needs a handshake between the client and the server before transferring the data. In the process of handshake, the cryptographic information of the data will be encrypted by both sides.

2) In this process, the middleman intercepts the handshake information of the client requesting server and simulates the client to request to the server (sending a set of encryption rules supported by himself to the server). The server will select a set of encryption algorithms and HASH algorithms and send its identity information back to the client in the form of certificate. The certificate contains information about the address of the site, the cryptographic public key, and the certification authority etc.

3) In this case, the middleman intercepts the certificate information returned by the server and replaces it with its own certificate information.

4) The client chooses to encrypt the data with the middleman's certificate after getting the middleman's response.

5) The middleman decrypts the client end request data with his own certificate.

6) After eavesdropping or modifying the request data, simulate the client to encrypt the request data to the server. This completes the whole process of the man-in-the-middle attack.

Protection:

1) Public Key Locking

Write the certificate public key to the client end apk to check whether the certificate public key is the same as the server transport.

2) Certificate locking

The client issues the public key certificate to store in the mobile phone client. When it communicates via the https, it will get the certificate information fixed in the client side but not from the service side obtains.

3.3. The application gateway has the ability to balance the load and fuse of the application interface, and can retry the failure

1) Load Balancing

The request pressure of an interface sometimes will exceed the processing power of a single service instance. In this case, we need to start multiple service instances. In order to avoid the transmission of request pressure to one of the instances, we implement a polling policy at the gateway level, that is, each request is spread over a different service instance.

2)Route fusing

When there is an exception in our back-end service, we do not want to throw the exception to the most outert layer and expect the service to be automatically downgraded. When an exception occurs to a service, the default information will be returned directly.
We use a custom fallback method and assign it to a route to implement the fusing process of the route access problem.

3)Failure Retry

Sometimes because of the network or other reasons, the service may be temporarily unavailable. In that time, we hope to retry the service.

We try to call another service instance or return the specified information when an exception occurs in a service by custom retry policy. Opening a retry is might be problematic in some cases, for example, when the pressure is too high and one instance stops responding. The

route transfers the traffic to another instance which is likely to result in the eventual collapse of all instances.

3.4. Cache cluster mainly serves background and interface services

The back-end interface services are divided into three levels:

1) The Gateway Layer

Mainly dealing with authentication, fuse break, load balancing, failure retry

2)Application Service Layer

Responsible for business logic processing

3)Basic Service Layer

Provide to add, delete, change and queue task of processing for basic data

In general, we put the cache processing of data on the application service layer because this layer knows the business requirements best and can organize the structure of the cached data according to the characteristics of the business. It is even more flexible to cache data taken from multiple basic services. However, if necessary, it can also be provided by the infrastructure service layer, but the precondition is it must provide a non-cached interface.

3.5. The application layer access database using multiple data sources

The same project sometimes involves multiple databases, that is, multiple data sources. Multiple data sources can be divided into two situations:

1) Two or more databases have no correlation and are totally independent, which can be developed as two projects. For example, in the game development, one database is platform database, the other is the corresponding database of game under platform;

2) Two or more databases are master-slave relations, such as use mysql to build a master-master, and then with multiple slaves; or using MHA to build master-slave replication;

3) Configuring multiple data sources directly with spring configuration profile

In the absence of correlation between the two databases, it can configure multiple data sources directly in the spring configuration prefile, and then do the transaction configuration separately

4) Configuration of multiple data sources which is based on AbstractRoutingDataSource and AOP

The basic principle is that we define a DataSource class which is named ThreadLocalRountingDataSource to inherit AbstractRoutingDataSource, and then inject master and slave data into ThreadLocalRountingDataSource in the configuration file, and then

configure it flexibly through AOP to decide where need to choose the master data source and where need to choose slave source.

3.6 Block chains and IPFS can be invoked through public services only

Save the files on the IPFS network, and the data will be scattered across the network nodes. When the clients need to get the file, they only need to use the returned hash value when the files are stored; and find the data piece in the IPFS network and reassemble them.

3.7. Database master-slave replication adopts GTID mode

GTID Summary:

1) The global transaction identity.
2) GTID transactions are globally but unique, and one transaction corresponds to only one GTID.
3) A GTID is executed only once on a server to avoid duplication of execution resulting in data confusion or master-slave inconsistency.
4) The replication method used by GTID to instead of classic method is not use binlog+pos to enable replication. However, it use master_auto_postion=1 to automatically match GTID breakpoints for replication.
5) It begins to support by MySQL-5.6.5 and improvea by MySQL 5.6.10.
6) In the traditional slave side, the binlog is no need open; but in GTID, the binlog at the slave end must be turned on for the purpose of recording the executed GTID (mandatory).

Components of GTID:

Server_uuid in front: followed by a serial number

For Example：server_uuid：sequence number

7800a22c-95ae-11e4-983d-080027de205a:10

UUID: The unique ID for each mysql instance is also understood as the source ID because it is passed to slave.

Sequence number: On each MySQL server, the sequence number grows from 1, and each numeric value corresponding to a transaction.

The advantages of GTID over traditional replication are as follows:

1) It is easier to implement failover without having to look for log_file and log_Pos as before.
2) It is easier to build the master-slave replicas.
3) It is safer than traditional replication.
4) The GTID is continuously not empty. When there is a data conflict between the master and slave, it can skip by adding empty things.

GTID Operational Principles:

1) when the master update data, it generates GTID before the transaction and records them in the binlog log.

2) The i/o thread on the slave side writes the changed binlog to the local relay log.
3) The sql thread gets GTID from relay log, and then compares binlog on the slave side whether has records.
4) If there is a record, it means the GTID transaction has been executed and the slave will be ignored.
5) If it not recorded, the slave will perform the GTID transactions from the relay log and record them to binlog.
6) It will judge whether it has a primary kay in the process of parsing; if not, the secondary index will be used or the whole scan will be used.