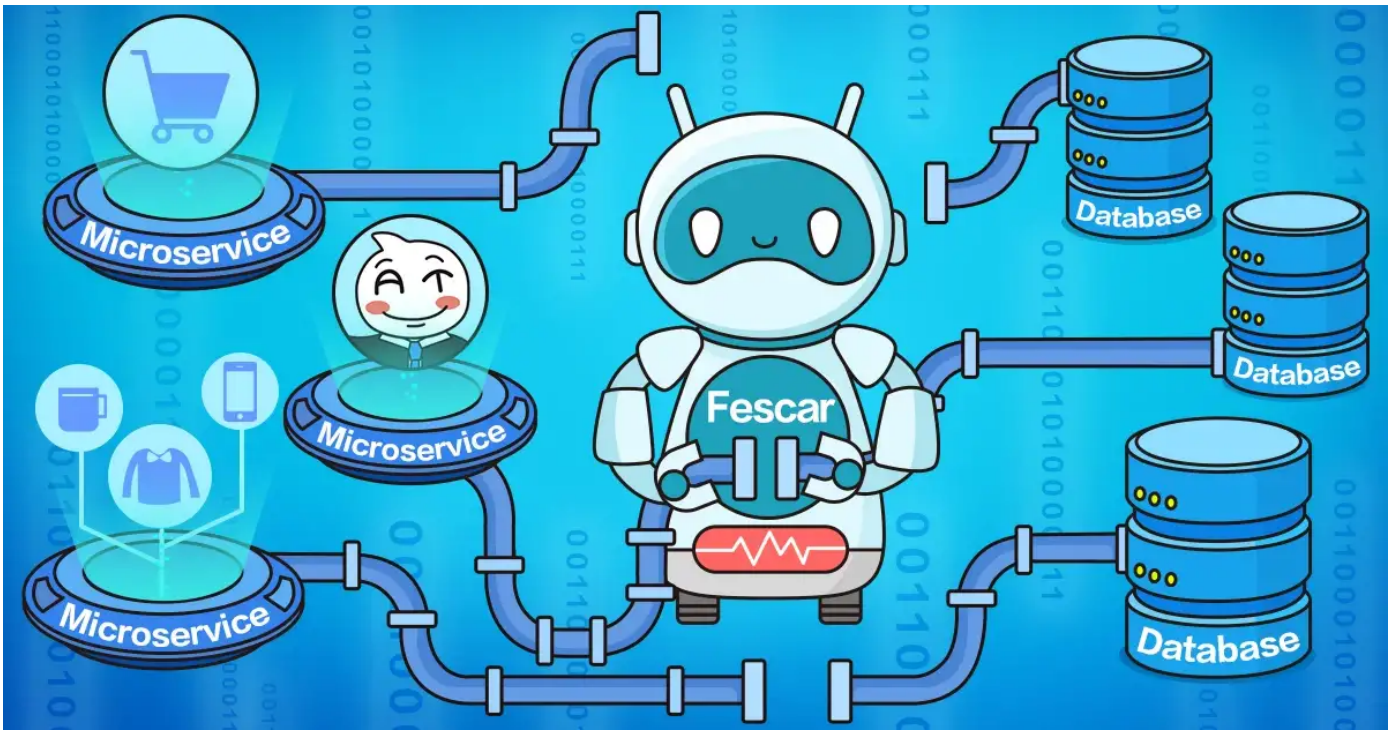
Published in HackerNoon.com

Alibaba Tech Follow

Feb 26, 2019 · 15 min read · Listen

Fescar: A Distributed Transaction Solution Open Sourced by Alibaba

To support microservice-based development, Alibaba has now launched Fescar, an open source version of its Global Transaction Service solution to the problem of distributed transactions.



*This article is part of the **Alibaba Open Source** series.*

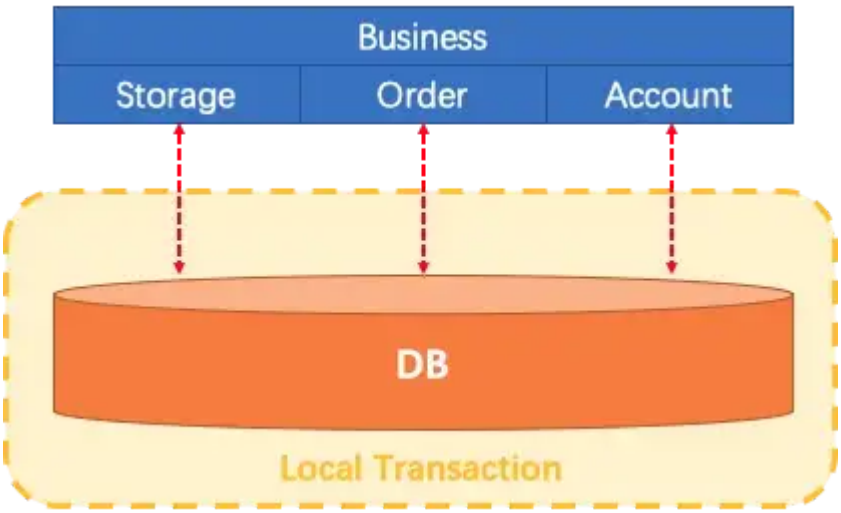
calling multiple services and operating multiple databases, introducing a formidable distributed transaction problem for service calls.

Currently, distributed transactions are the greatest obstacle to realizing microservices, and also the most challenging of relevant technical issues. To overcome these difficulties, Alibaba initially developed the Global Transaction Service (GTS) solution for its systems. Now, it has expanded this solution by launching an open source version called Fescar, short for "Fast and Easy Commit and Rollback".

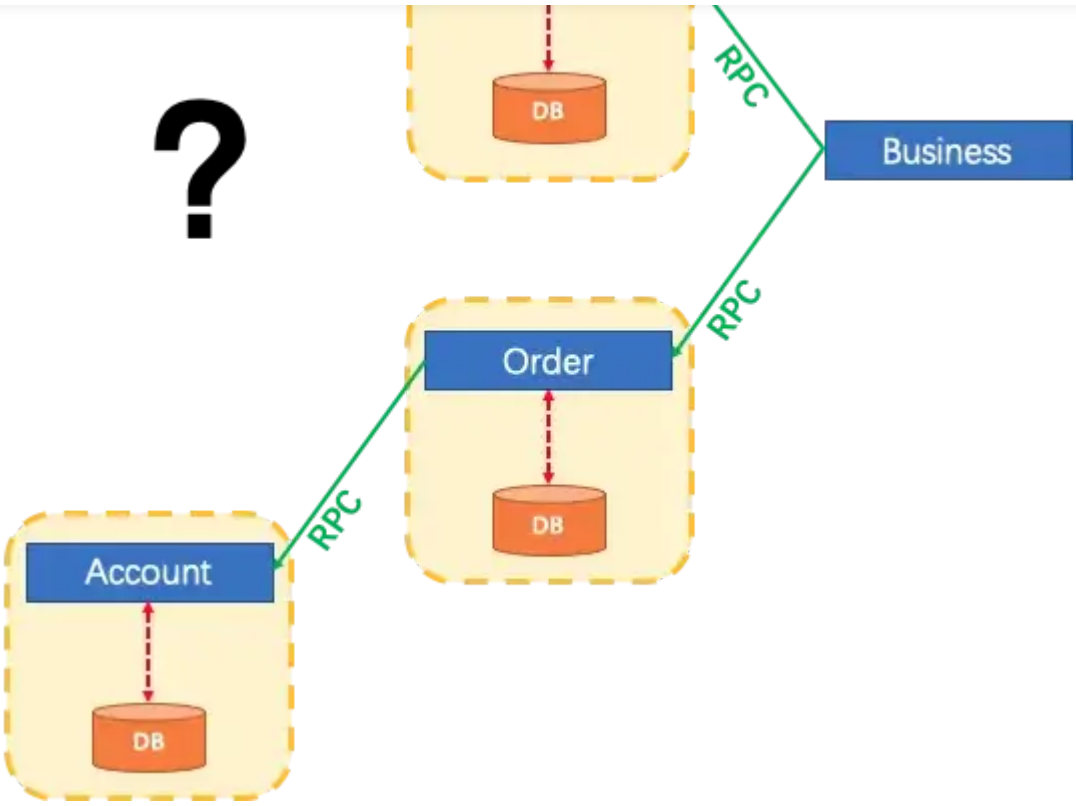
To mark its recent release on GitHub, this article looks in detail at Fescar's fundamentals and its support for microservice-based development.

Origins of the Distributed Transaction Problem

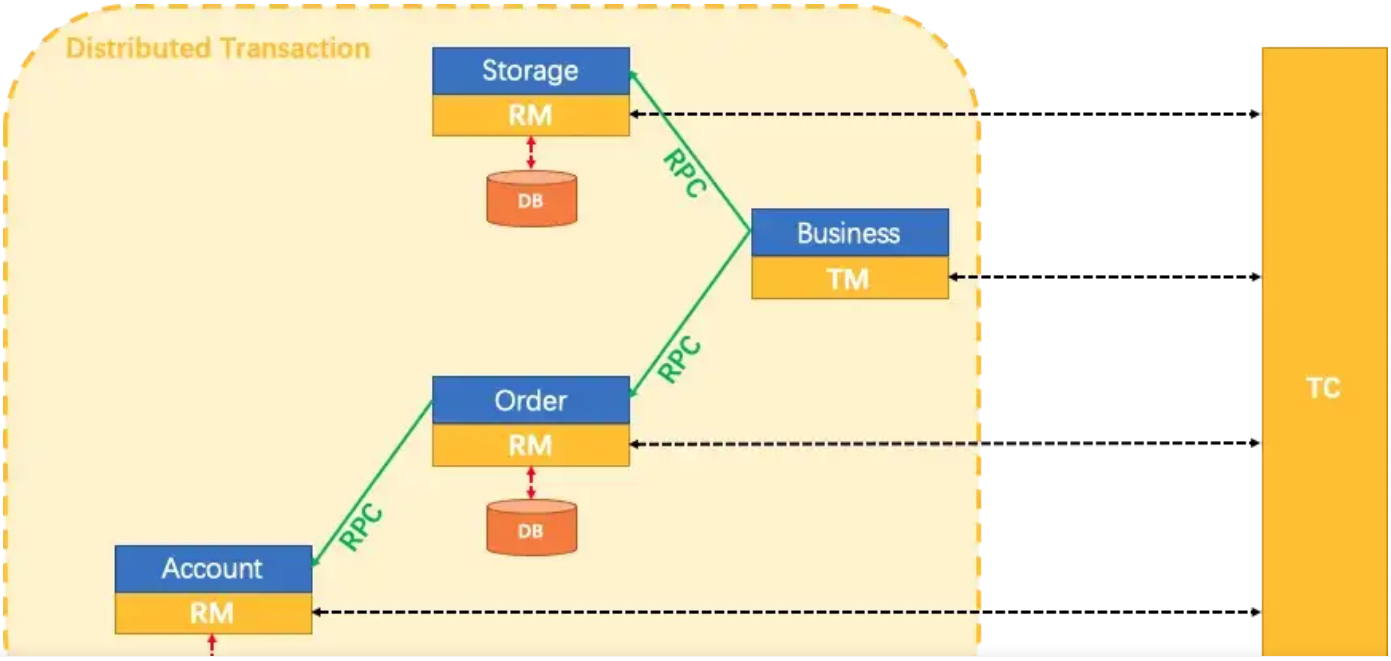
To understand how microservices cause the distributed transaction problem, it makes sense to consider the example of a traditional monolithic app that completes a business process by updating data on a single data source through three modules. Naturally, local transactions guarantee data consistency throughout the business process.



As business demands and architecture change, individual applications are divided into microservices. The original three modules are split into three separate services, each



In this case, local transactions are still able to guarantee data consistency within each service. However, to ensure consistency of global data across the entire business level, a distributed transaction solution becomes necessary.



distributed microservices, resulting in a long history of encounters with the problem of distributed transactions under the microservice architecture.

In 2014, Alibaba's middleware team released Taobao Transaction Constructor (TXC) to provide distributed transaction services for intra-group applications.

In 2016, TXC was implemented on Alibaba Cloud as a Global Transaction Service (GTS) following productization, becoming the only distributed transactional product on the cloud at the time. As such, it began to serve many external customers through Alibaba's public cloud and Apsara solutions.

Entering 2019, based on the technology accumulation from TXC and GTS, Alibaba's middleware team launched the open source Fescar project to further build its distributed transaction solution in collaboration with the development community.

TXC, GTS, and Fescar all arise from the same origins, and have together produced a distinctive solution to the problem of distributed transactions under the microservice architecture.

Original design goals

In the fast-growing internet era, the ability to quickly undergo trial and error is critical for businesses. On the one hand, the introduction of microservices and distributed transaction support to technical infrastructure should not generate any additional research and development burden on the business level. On the other hand, businesses that introduce distributed transaction support should maintain essentially the same level of performance, and should not be significantly hindered by transaction mechanisms.

Based on these criteria, the outset of Fescar's design involved two crucial considerations. First, it should not cause any intrusion into the business. "Intrusion" here means that the application is designed and modified at the business level because of technical problems caused by distributed transactions. Such design and modification often results in high R&D and maintenance costs for the application

Second, the design needed to ensure high performance. The introduction of distributed transaction guarantees inevitably brings about additional overhead, causing performance degradation. Alibaba needed to reduce the performance loss caused by the introduction of distributed transactions to the lowest possible level, so that the availability of applications would not be affected by the introduction of distributed transactions.

Shortcomings of previous solutions

Previous distributed transaction solutions can be categorized as those which are intrusive to businesses and those which are non-intrusive to businesses.

Among mainstream solutions, only XA-based solutions are non-intrusive to businesses. Nevertheless, XA-based solutions present three problems. Firstly, they require the database to provide support for XA; in the case of databases that do not support XA or that provide poor support, such as MySQL versions prior to 5.7, there is no way to use these solutions. Secondly, they are constrained by the protocol itself, and with them the transaction resource has a long lock-in cycle. Long-term resource locking is often unnecessary from a business perspective, but because the transaction resource manager is the database itself, the application layer cannot intervene. This results in poor performance and difficulty for optimization. Lastly, implemented XA-based distributed solutions rely on heavy-duty application servers like Tuxedo, WebLogic, or WebSphere, making them inapplicable to microservice architectures.

XA was in fact the only solution for distributed transactions during early development. It is complete, but in practice it often needs to be given up for a variety of reasons including but not limited to the problems discussed above. For example, eventual consistency solutions based on reliable messages, TCC, and Saga all belong to this category. These solutions' mechanisms are not explained in this article, due to the availability of information on the internet. In short, they require that distributed transactional technology constraints be considered at the business level of applications' designs. Usually, each service needs to be designed to implement forward

can effectively solve problems to play an important role in business application systems across various industries. Still, adopting such solutions should be seen as a distant second choice. Imagining that XA-based solutions could be lighter weight and ensure the performance requirements of businesses, it is unlikely people would willingly take the distributed transaction problem to the business level.

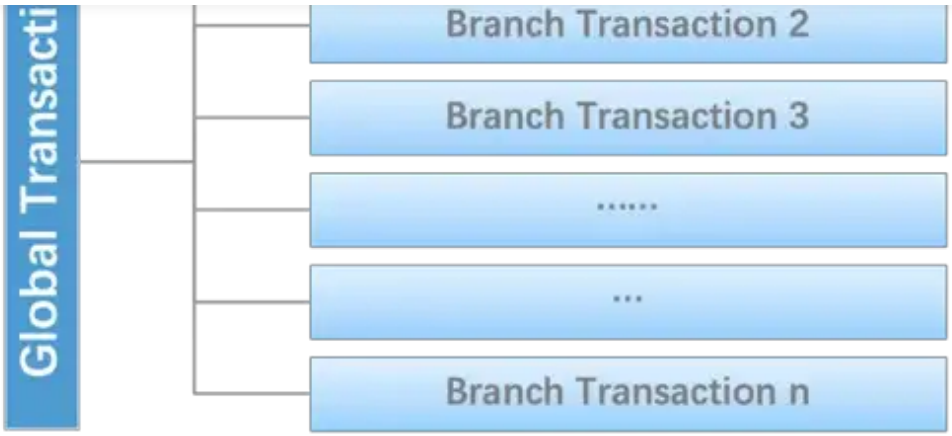
Thus, an ideal solution should be as simple as using a local transaction, with business logic only focusing on the needs of the business level and not regarding constraints on the transaction mechanism.

Solution Principles and Design

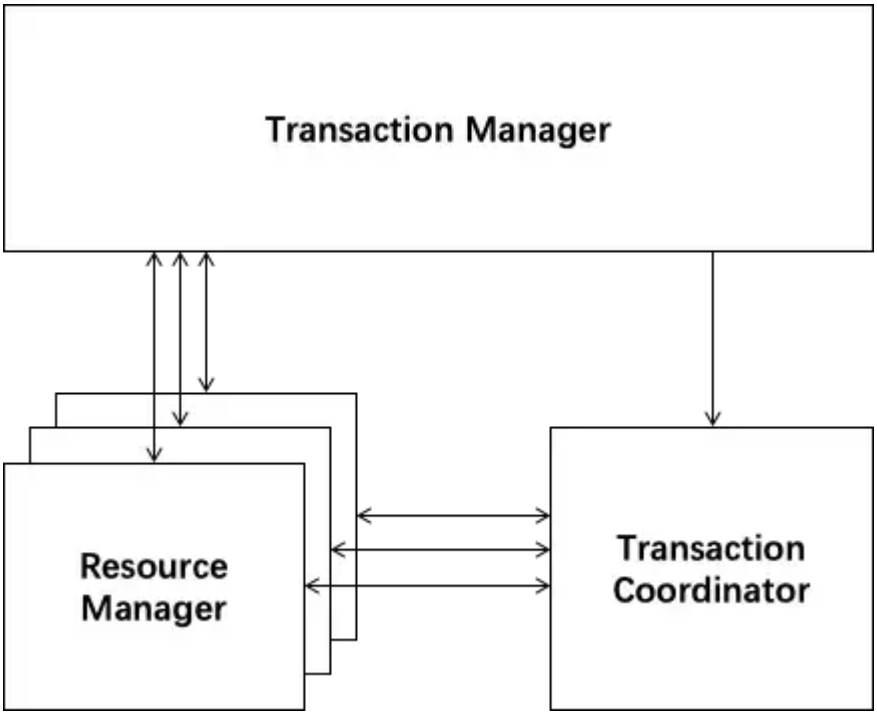
Treating a non-intrusive XA solution as the basis for a solution that is non-intrusive at the business level, the key question that needs to be solved is how XA can be adapted to resolve the issues it presents. The following sections discuss the principles involved in this adaptation in detail.

Defining a distributed transaction

First, a distributed transaction can naturally be viewed as a global transaction that contains several branch transactions. The responsibility of a global transaction is to coordinate the branch transactions under its jurisdiction so that they agree, either by successfully committing together or failing and rolling back together. In addition, the branch transaction itself is usually a local transaction that satisfies the ACID principles. This is a basic understanding of the distributed transaction structure, which is consistent with that of XA.



Second, similarly to the XA model, three components are defined to negotiate the processing of distributed transactions:



The transaction coordinator (TC) maintains the running state of global transactions and is responsible for coordinating and driving the commit or rollback of global transactions.

The transaction manager (TM) controls the boundaries of global transactions, is responsible for opening global transactions, and ultimately initiates the global commit

coordinator to drive the commit and rollback of branch (local) transactions.

A typical distributed transaction process works as follows:

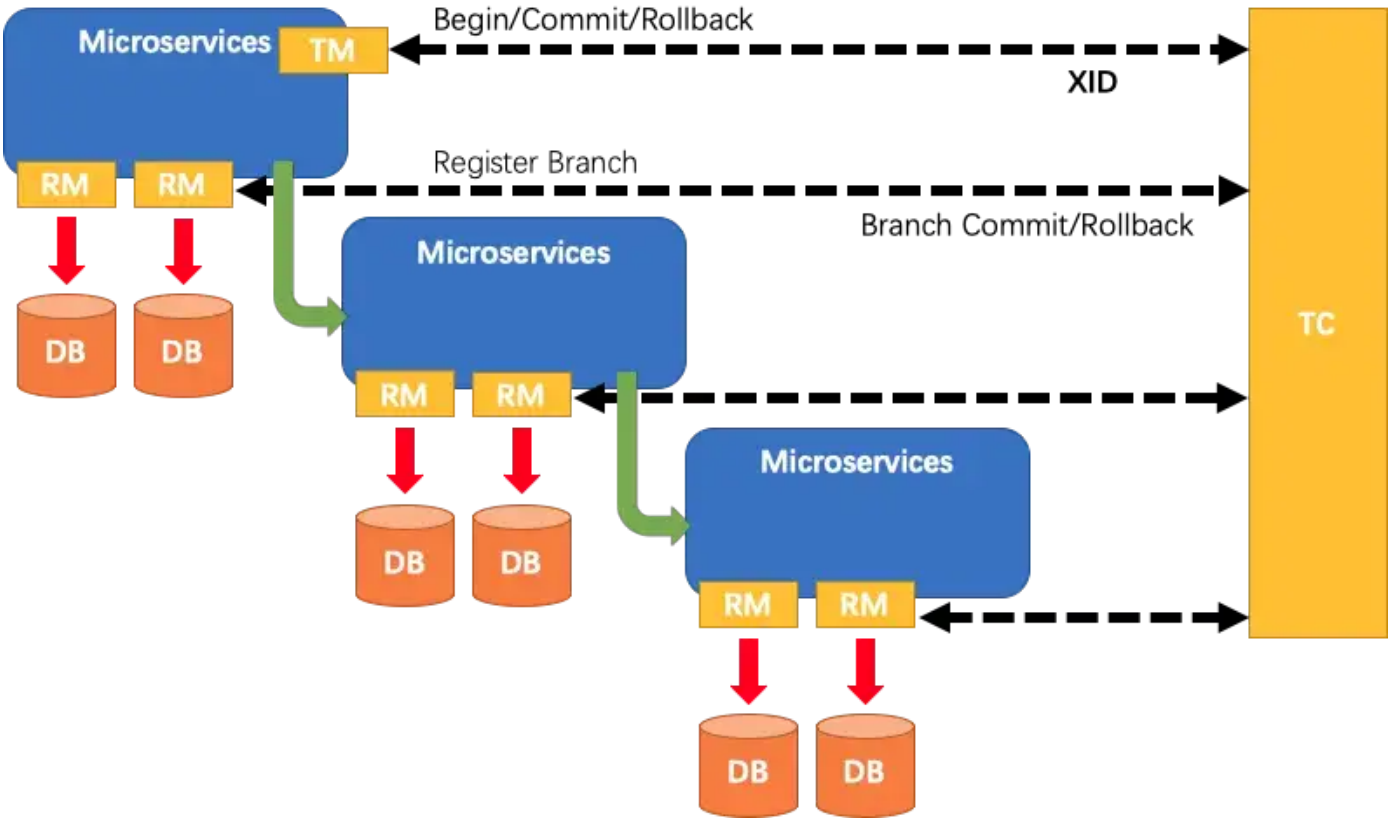
1. The TM requests that the TC open a global transaction and the global transaction is created, successfully generating a globally unique XID.

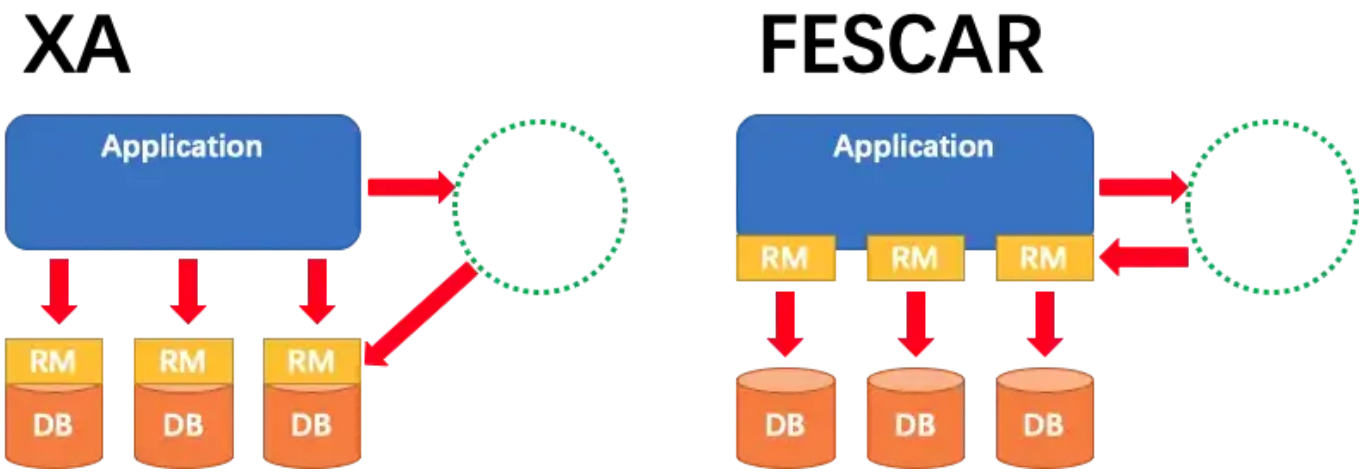
2. The XID is propagated in the context of the microservice call link.

3. The RM registers the branch transaction with the TC and incorporates it into the jurisdiction of the global transaction corresponding to the XID.

4. The TM initiates a global commit or rollback resolution for the XID to the TC.

5. The TC schedules all branch transactions under the XID to complete the commit or rollback request.

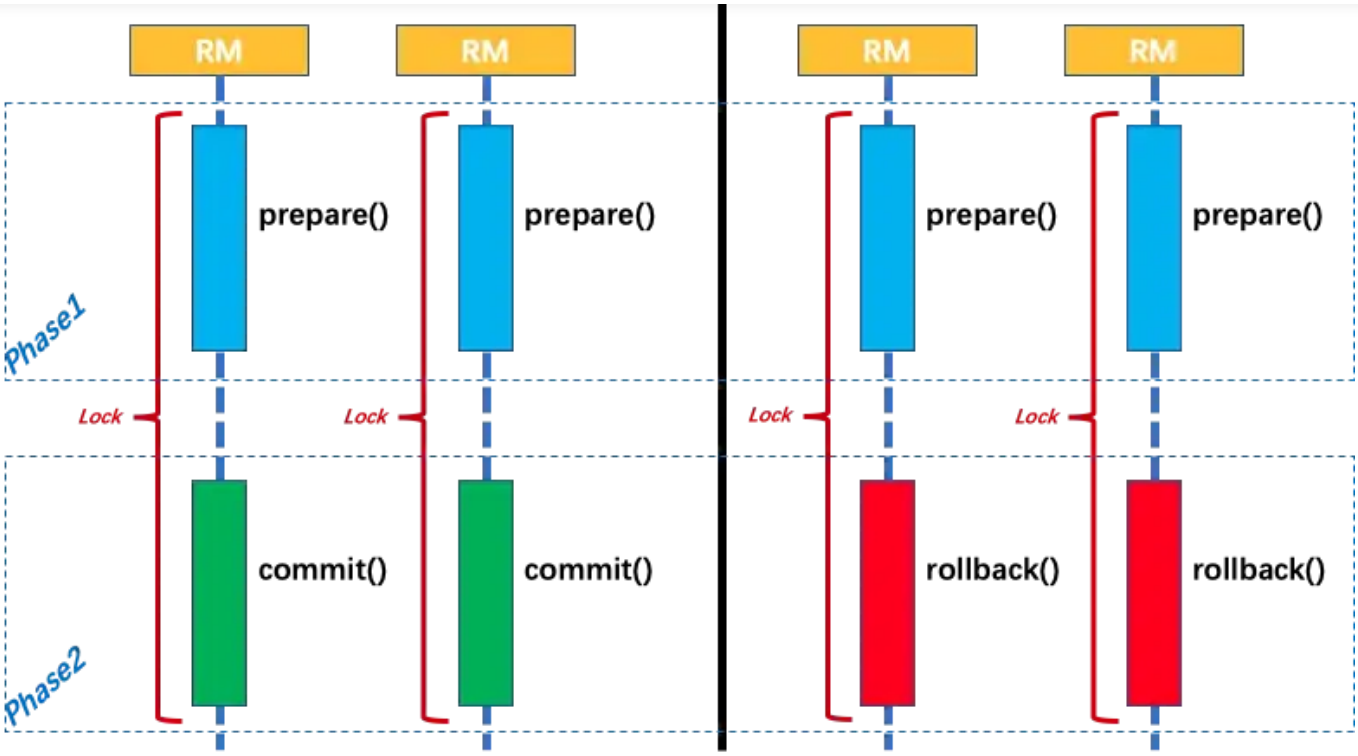




The XA solution's RM is actually at the database level, and the RM is essentially the database itself (available for applications' use by providing XA-enabled drivers).

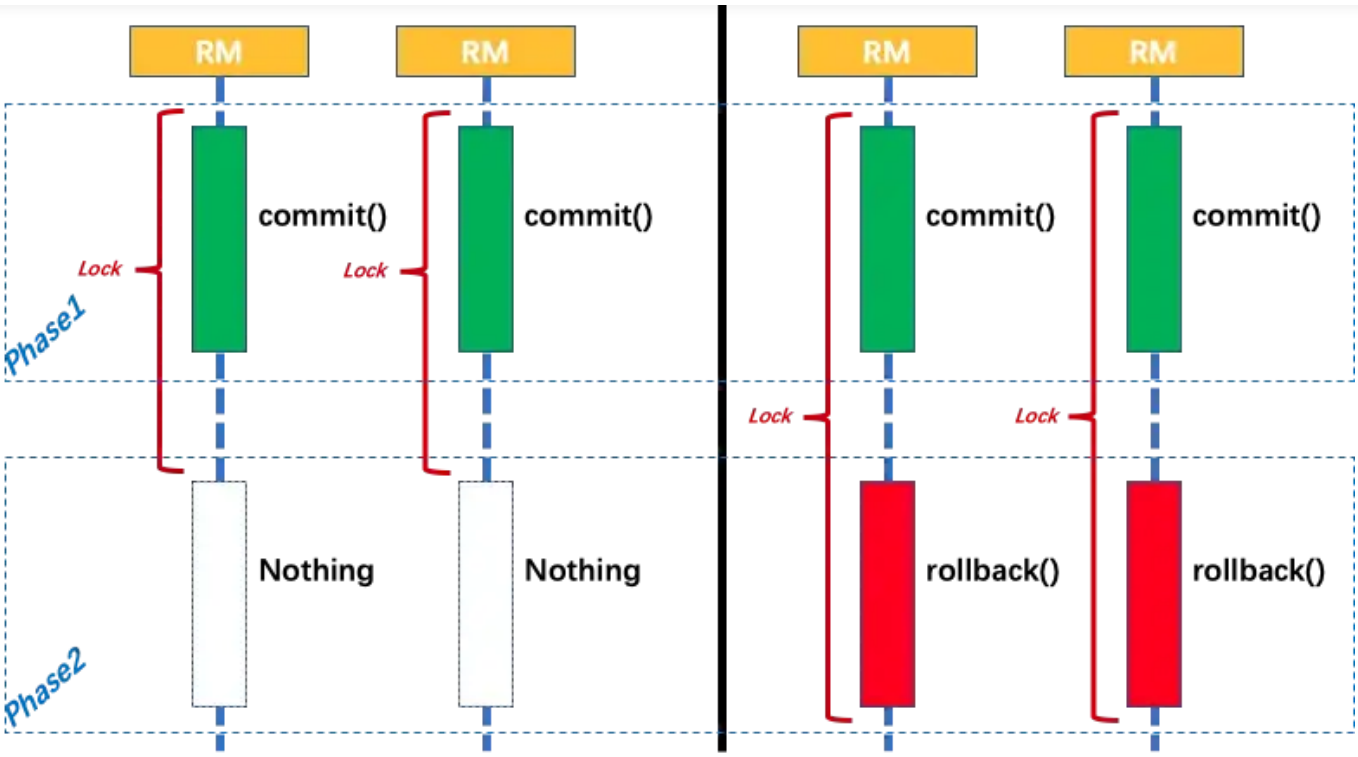
Fescar's RM is deployed on the application side as a middleware layer in the form of a second-party package. It does not depend on the support for the XA protocol provided by the database itself, and does not require the database to support the protocol. This is very important for a microservices architecture: the application layer does not need to adapt two different database drivers for the two different scenarios of local transactions and distributed transactions. This design principle strips off the distributed transaction solution's requirements for database support for the protocol.

Fescar's second major difference concerns two-phase commit. The following shows XA's 2PC process:



Whether Phase 2's resolution is commit or rollback, the transactional resource locks are kept until Phase 2 is completed.

Imagine a normal business with a high probability that more than 90% of the transactions will eventually be successfully committed. Can local transactions be committed in Phase 1? With a percentage of more than 90%, committing earlier can save the lock time in Phase 2 and improve the overall efficiency.

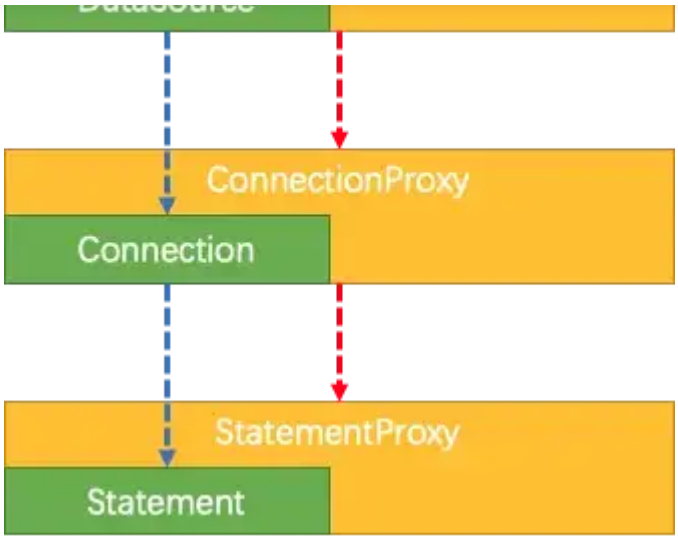


As shown above, Fescar's design reduces the transaction lock time in most scenarios, thus increasing the concurrency of transactions.

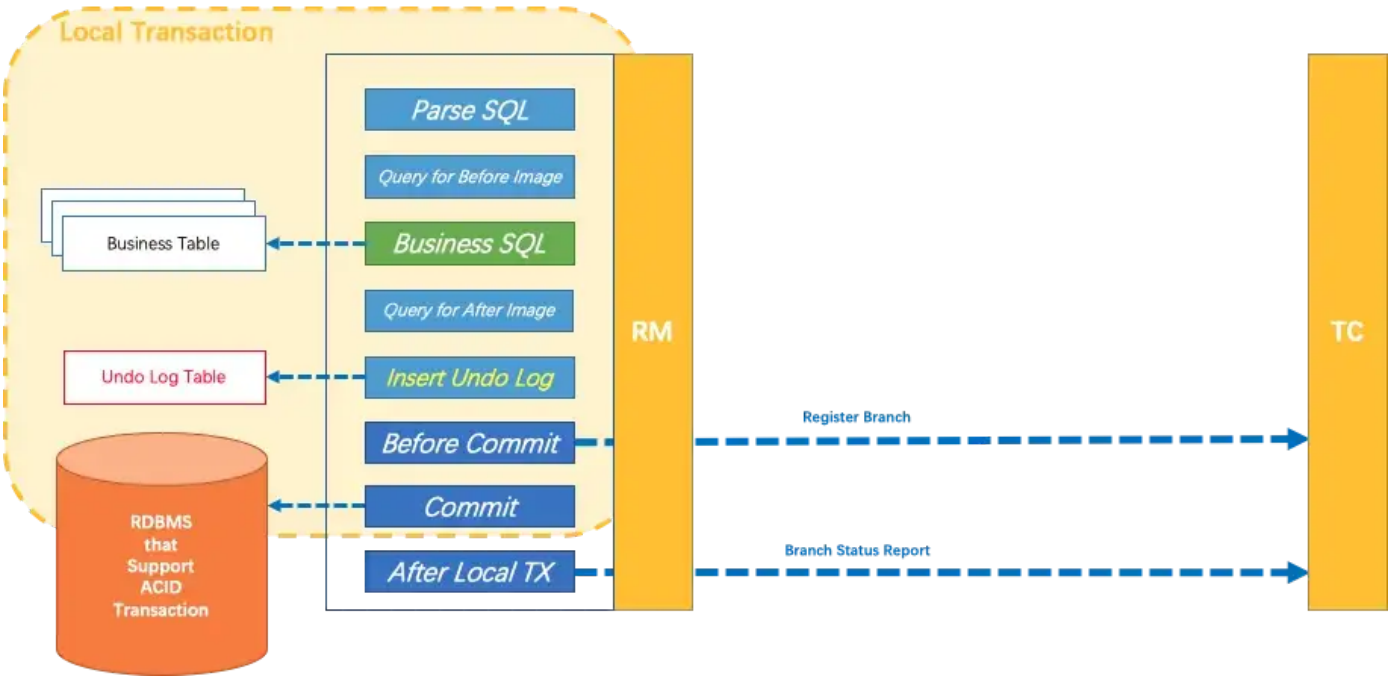
The next question is of course, how does Phase 2 roll back when Phase 1 is committed?

How branch transactions commit and rollback

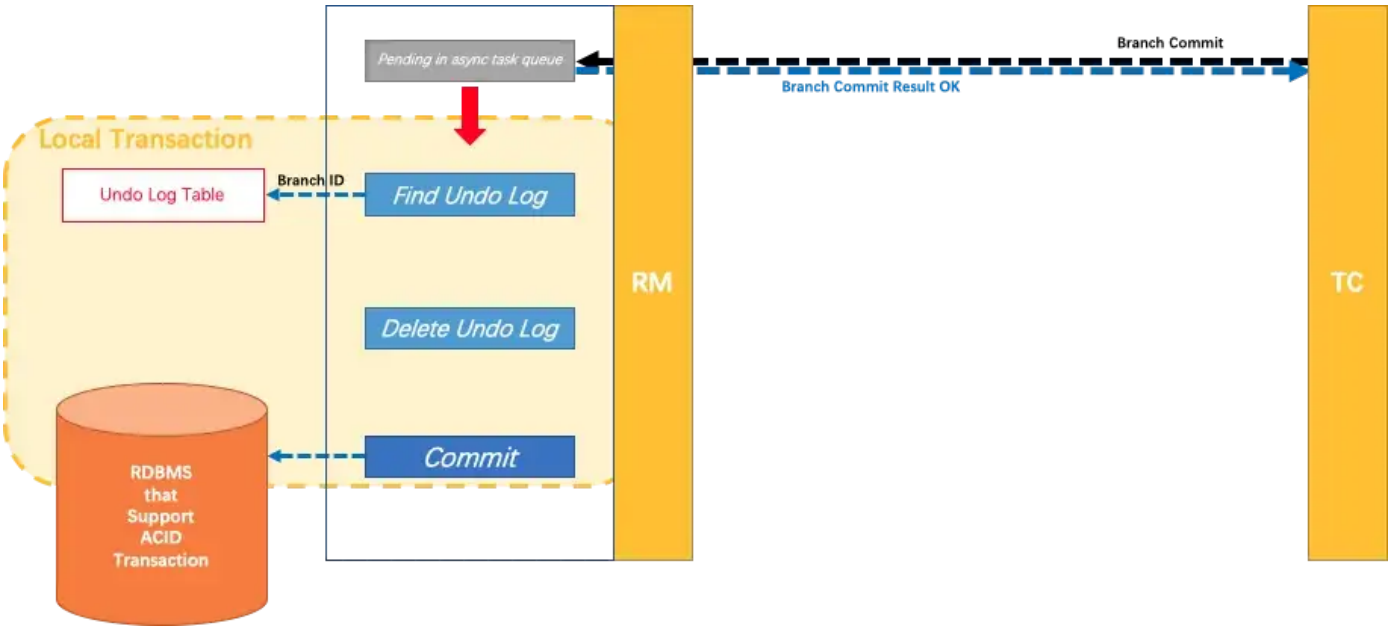
First, the application needs to use Fescar's JDBC data source proxy, which is Fescar's RM.



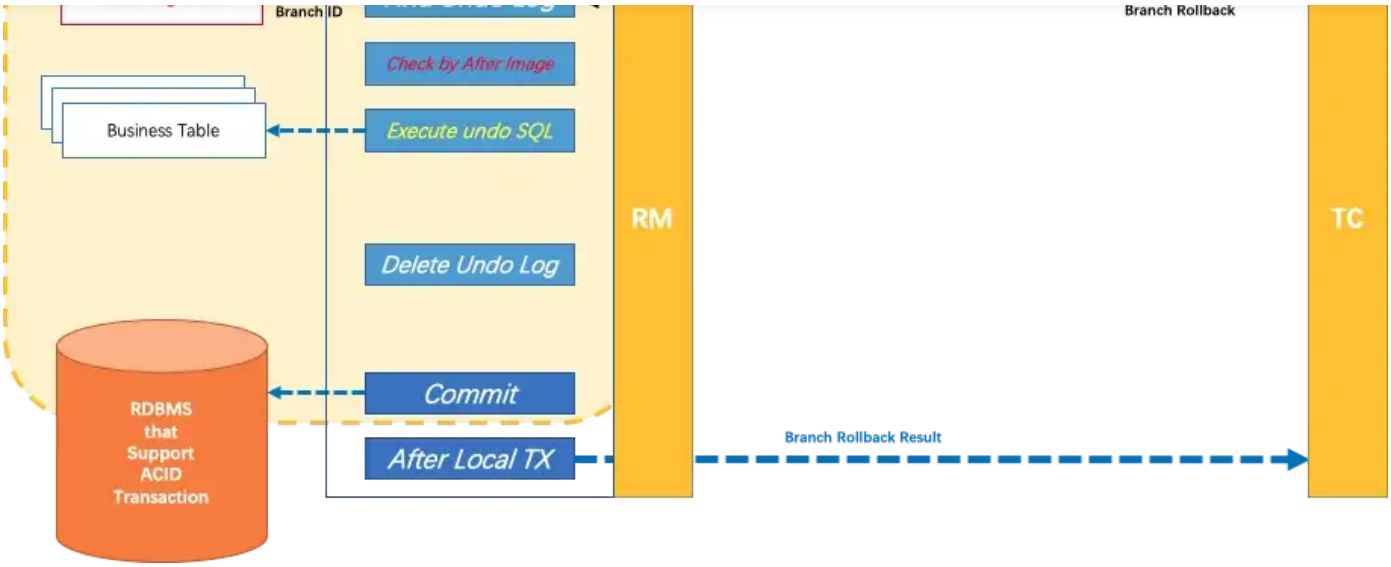
Phase 1 works as follows: By parsing the business SQL, Fescar's JDBC data source proxy organizes the data images of business data before and after the SQL execution into an undo log, and uses the ACID feature of the local transaction to write the update of the business data and the undo log in the same local transaction to commit. In this way, it can guarantee that any update of the committed business data definitely has a corresponding undo log.



Phase 2 works as follows: If the resolution is global commit, then the branch transaction has already completed the commit at the time and no synchronous reconciliation is required. (Only the rollback log needs to be cleaned up asynchronously). Phase 2 can thus be completed very quickly.



If the resolution is global rollback, the RM receives the rollback request from the coordinator, finds the corresponding undo log record through the XID and the Branch ID, and generates the reverse update SQL, which it executes by rolling back the record to complete the branch rollback.



The transaction propagation mechanism

XID is the unique identifier of a global transaction. The transaction propagation mechanism needs to pass the XID through the service call link and bind it to the service's transaction context. In this way, the database update operation in the service link is registered with the global transaction represented by the XID and is included in the jurisdiction of the same global transaction.

Based on this mechanism, Fescar can support any microservice RPC framework, provided that it finds a mechanism in a specific framework that can transparently propagate the XID, for example, Dubbo's Filter + RpcContext.

Corresponding to the Java EE specification and Spring-defined transaction propagation properties, propagations supported and not supported by Fescar are as follows:

· PROPAGATION_REQUIRED: supported by default

· PROPAGATION_SUPPORTS: supported by default

· PROPAGATION_MANDATORY: supported through the API

· PROPAGATION_REQUIRES_NEW: supported through the API

· PROPAGATION_REQUIRED_NESTED: not supported

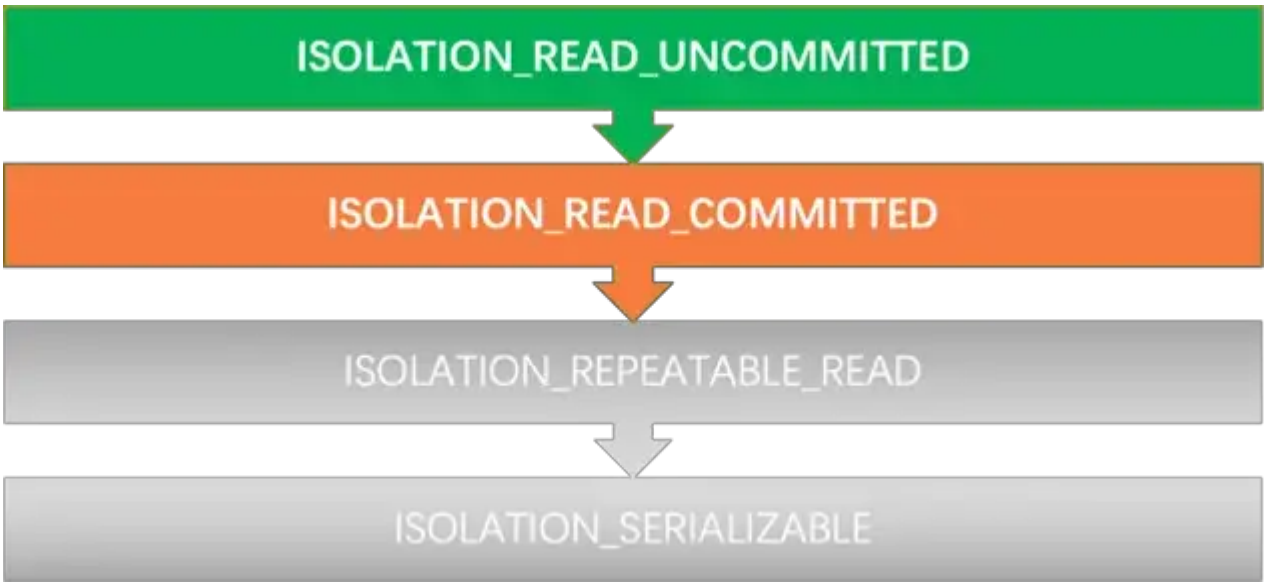
Isolation

The isolation of global transactions is based on the local isolation level of branch transactions.

Based on the premise that the database local isolation level is “read committed” or above, Fescar is designed to use the global write exclusive lock maintained by the transaction coordinator to ensure the write isolation between transactions, and the global transaction is defined by default on the isolation level of “read uncommitted”.

The consensus on isolation levels is that most applications work without problems under the isolation level of “read committed”. In fact, in most scenarios, applications also have no problem working under the isolation level of “read uncommitted”.

In extreme scenarios, if the application needs to reach global read committed, Fescar also provides a mechanism for achieving that goal. By default, Fescar works under the isolation level of “read uncommitted”, ensuring efficiency for most scenarios.



The ACID attributes of the transaction is a complicated topic in Fescar, and will be explained in-depth in a future article.

resources involved in branch transactions must be relational databases that support ACID transactions. The branch commit and rollback mechanisms depend on the guarantee of local transactions. Therefore, if the database used by the application does not support transactions, or is not a relational database at all, Fescar is not applicable.

In addition, there are still some limitations in the implementation of Fescar. For example, the transaction isolation level is supported up to the level of read committed; the SQL parsing does not cover all the syntax; and so on.

To cover the deficiency that the Fescar native mechanism does not currently cover, another working mode is defined. The Fescar-native working mode described above is called AT (Automatic Transaction) mode, which is non-intrusive to the business. Another working mode corresponding to this is MT (Manual Transaction) mode, in which branch transactions need to be defined by the business.

Basic branch behavior

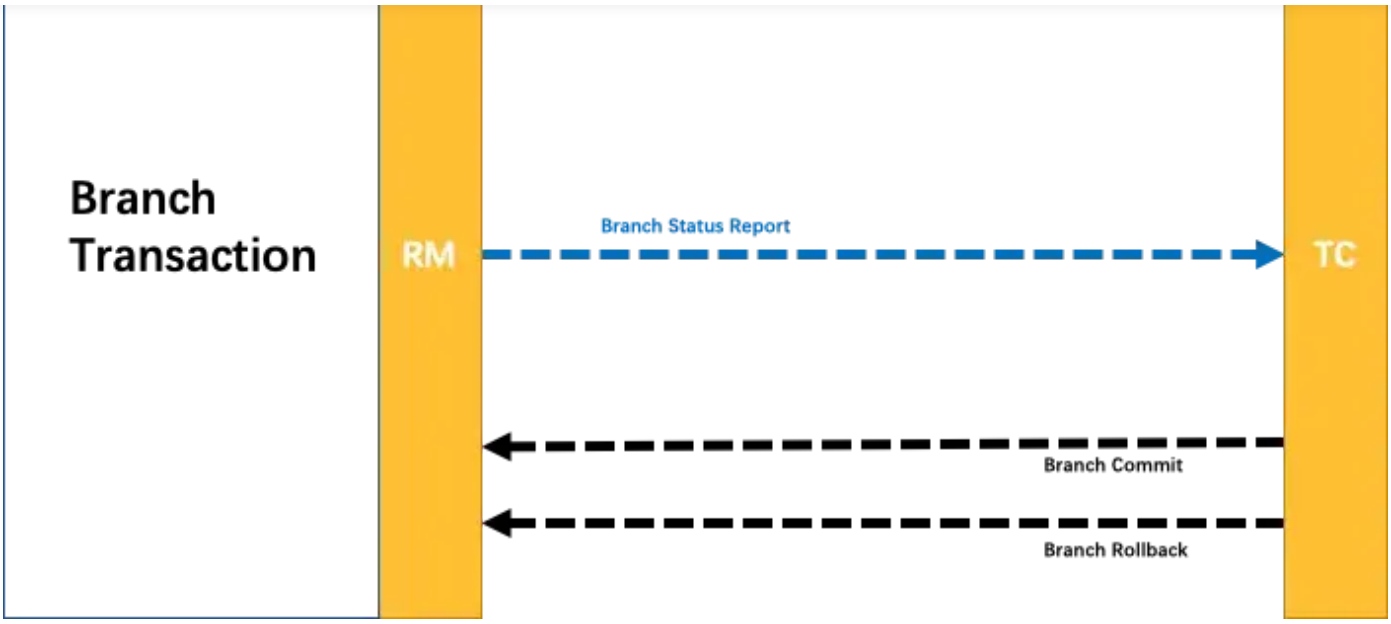
A branch transaction that is part of a global transaction, in addition to its own business logic, includes four actions that interact with the coordinator:

· Branch registration: Before its data operations are performed, the branch transaction needs to register with the coordinator, enabling the upcoming data operations to be included in the management of an opened global transaction. After the branch registration is successful, the data can be operated.

· Status reporting: After its data operations are completed, the branch transaction must report the execution result to the transaction coordinator.

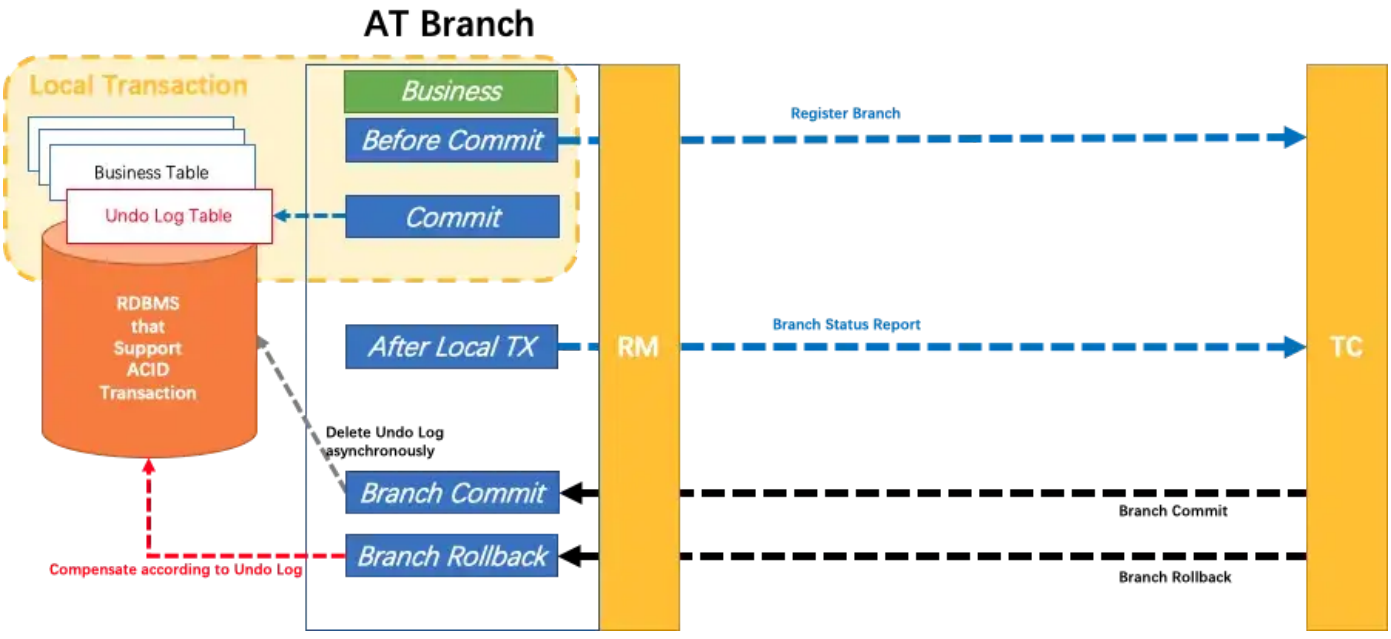
· Branch commit: The branch transaction completes the branch commit in response to such a request issued by the coordinator.

· Branch rollback: The branch transaction completes the branch rollback in response to such a request issued by the coordinator.

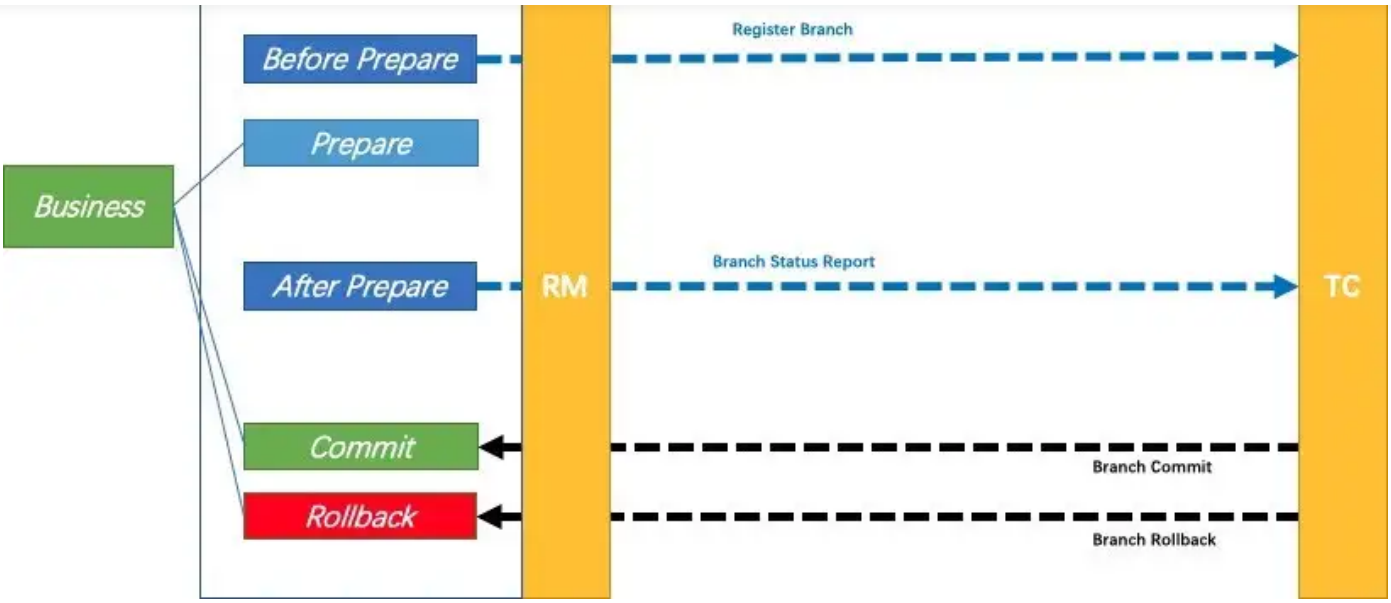


AT branch behavior mode

The business logic does not pay attention to the transaction mechanism, and the interaction process between the branch and the global transaction is performed automatically.



MT branch behavior mode



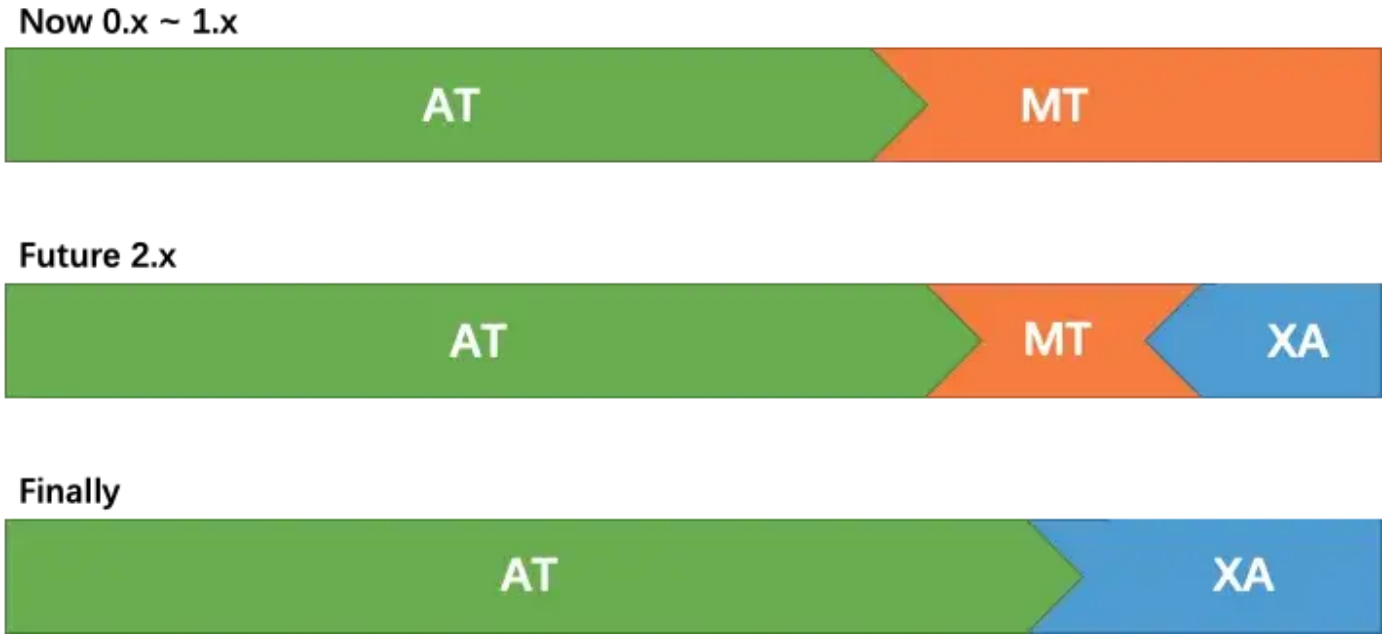
On the one hand, the MT mode is complementary with the AT mode. On the other hand, its more important value is that through it many non-transactional resources it can be included in the management of the global transaction.

Hybrid mode

Because AT branches and MT branches are fundamentally consistent in their behavioral patterns, they are fully compatible; that is, in a global transaction, both AT and MT branches can exist. In this way, the purpose of comprehensive coverage for business scenarios can be achieved. For those that support AT mode, AT mode is used; for those that cannot currently support AT mode, the MT mode is used instead. In addition, MT-managed non-transactional resources can also be included in the management of the same distributed transaction along with relational database resources that support transactions.

Application scenario outlook

In terms of the original intent of Fescar's design, an ideal distributed transaction solution should be non-intrusive at the business level. MT mode is a natural supplement in cases where AT mode cannot fully cover all scenarios. The Fescar team hopes to gradually expand the range of supported scenarios through the continuous evolution of AT mode, while MT mode gradually converges. In the future, Fescar will



Extension Points

Microservices framework support

The propagation of transaction contexts between microservices requires tailoring the optimal solution, which is transparent to the application layer, based on the mechanisms of the microservices framework itself. Developers interested in building in this area can refer to the built-in support for Dubbo to support other microservices frameworks.

Supported database types

Because AT involves the parsing of SQL, there are some specific adaptations that work on different types of databases. Developers interested in building in this area can refer to the built-in support for MySQL to support other databases.

Configuration and service registration discovery

Fescar supports solutions that access different configurations and support service registration discovery, such as Nacos, Eureka, and ZooKeeper.

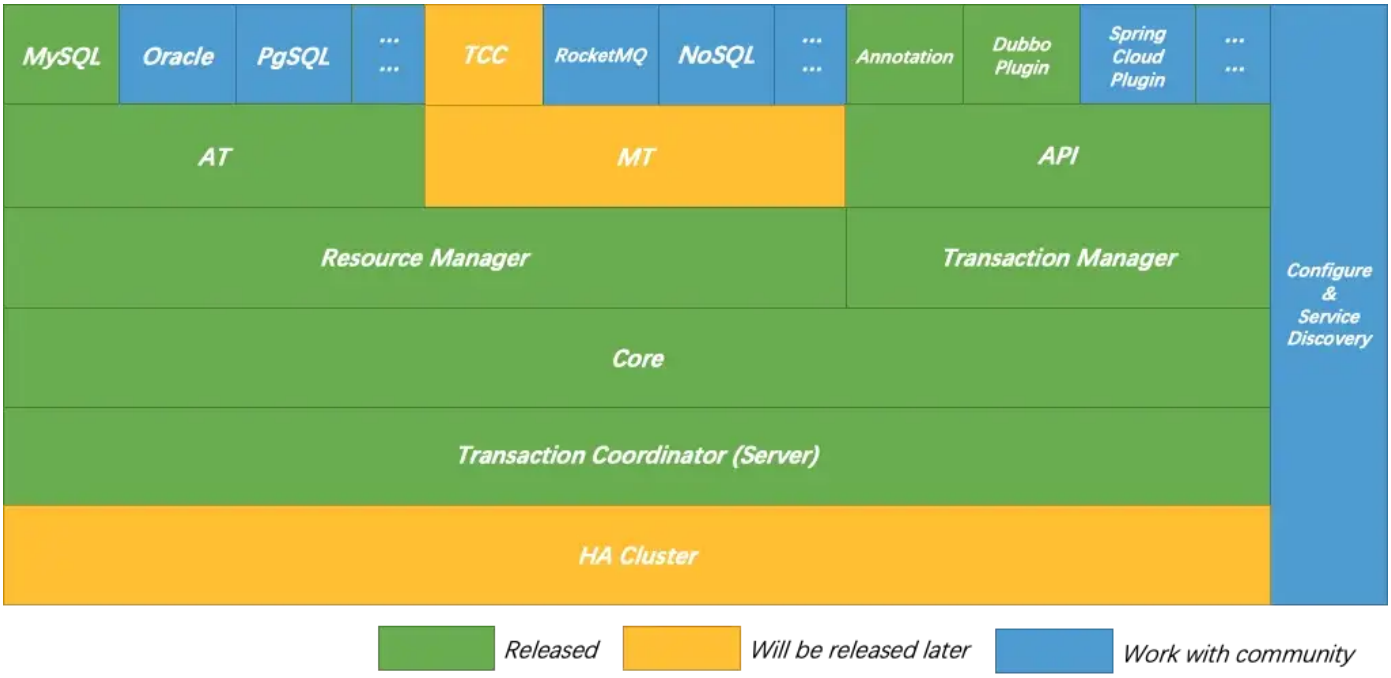
packaging of MT branches, for example with Redis, HBase, and RocketMQ transaction messages. Developers interested in building in this area can contribute a range of ecosystem adaptation options.

High-availability solutions for the distributed coordinator

For different scenarios, different methods are supported as highly available solutions for the Transaction Coordinator Server side. For example, the persistence of the transaction state can be a file-based implementation of a database-based implementation; state synchronization between clusters can be based on RPC communications or on high-availability KV storage.

Roadmap

Blueprint



In the above blueprint, green areas have been released and open sourced, while yellow areas will be released by Alibaba in subsequent versions. Blue areas are ecosystem parts jointly built by Alibaba and the community.

For support for different microservices frameworks, developers can refer to Dubbo's implementation.

For support for MQ and NoSQL, developers can refer to the implementation of TCC.

Regarding configuration and service registration discovery, developers can access any framework that can provide such services with a small amount of work.

Of course, the community is also welcome to participate in sections outside of the blue area and contribute better solutions.

In addition, XA is a standard for distributed transactions and is indispensable for a complete distributed transaction solution. In the vision for its planning, Fescar must add support for XA.

Preliminary version planning

The following are planned details for release versions to come.

v0.1.0:

· Microservices Framework Support: Dubbo

· Database support: MySQL

· Annotation based on Spring AOP

· Transaction Coordinator: Stand-alone version

v0.5.x:

· Microservices Framework Support: Spring Cloud

· MT mode

· Transaction Coordinator: High-availability cluster version

v0.8.x:

· Metrics

· Console: Monitoring/Deployment/Upgrade/Extension

v1.0.0:

· General Availability: Applicable to the production environment

v1.5.x:

· Database support: Oracle/PostgreSQL/OceanBase

· Annotation that does not rely on Spring AOP

· Optimized processing mechanism for hotspot data

· RocketMQ transaction messages are included in global transaction management

· NoSQL is incorporated in the adaptation mechanism of global transaction management

· Support for HBase

· Support for Redis

v2.0.0:

· Support for XA

Over the course of its development, the Fescar team will strongly emphasize priorities voiced in the community, and will communicate its road map accordingly. Readers can learn more on GitHub or through Alibaba Cloud.

Sign up for Get Better Tech Emails via HackerNoon.com

By HackerNoon.com

how hackers start their afternoons. the real shit is on hackernoon.com. Take a look.

Emails will be sent to patel321nisarg@gmail.com. Not you?

Get this newsletter