Other languages: 繁體中文

Database - FastDbClient

As the name implies, FastDbClient will provide higher performance than the normal DbClient. Unlike DbClient has own event loop, it shares the event loop with network IO threads and the main thread of the web application, which makes the internal implementation of FastDbClient available in a lock-free mode and more efficient.

Tests show that FastDbClient has a 10% to 20% performance improvement over DbClient under extremely high load conditions.

Create and Get

FastDbClient must be created automatically by the framework with the configuration file, or by calling the app.createDbClient() interface:

The sub-option `is_fast` of the db_client option in the configuration file indicates if the client is a FastDbClient.Or user can create a FastDbClient by calling the app.createDbClient() method with the last parameter set to true.

The framework creates a separate FastDbClient for each IO's event loop and the main event loop, and each FastDbClient manages several database connections internally. The number of event loop of IO is controlled by the framework's "threads_num" option, which is generally set to the number of CPU cores of the host. The number of the DB connections per event loop is the value of the DB client "connection_number" option. Please refer to Configuration File. Therefore, the total number of DB connections held by FastDbClient is `(threads_num+1) * connection_number`.

The interface to get a FastDbClient is similar to the normal DbClient, as follows:

```
orm::DbClientPtr getFastDbClient(const std::string &name = "default");
/// Use drogon::app().getFastDbClient("clientName") to get a FastDbClient
object.
```

It should be pointed out that due to the special nature of FastDbClient, the user must call the above interface in the IO event loop thread or the main thread to get the correct smart pointer. In other threads, only the null pointer can be obtained and cannot be used.

Usage

The use of FastDbClient is almost identical to that of the normal DbClient, except for the following limitations:

- Both the get and the use of it must be in the framework's IO event loop thread or the main thread. If it is used in other threads, there will be unpredictable errors (because the lock-free condition is destroyed). Fortunately, most of the application programming is in the IO thread, such as within the processing functions of various controllers, within the filter function of filters. It is easy to know that

the various callback functions of the FastDbClient interface are also in the current IO thread, and can be safely nestedly used.

- Never use the blocking interface of FastDbClient, because this interface will block the current thread, and the current thread is also the thread that handles the database IO of this object, which will cause permanent blocking, and the user has no chance to get the result.
- Synchronous transaction creation interfaces are likely to block (when all connections are busy), so FastDbClient's synchronous transaction creation interface returns null pointers directly. If you want to use transactions on FastDbClient, please use the asynchronous transaction creation interface.
- After using the FastDbClient to create an Orm Mapper object, you should also use only asynchronous non-blocking interfaces of the mapper object.

Next: Automatic batch mode