MEP 18 -- Support Milvus 2.0 C++ SDK

Current state: Accepted

ISSUE: https://github.com/milvus-io/milvus/issues/7713

PRs:

Keywords: C++ SDK

Released: with Milvus 2.1

Authors:

Summary

Deliver C++ SDK toolkit with full functionality for Milvus 2.0. Provide both static lib and dynamic lib for users.

Motivation

We've seen many users demands for C++ SDK, it is probably the most useful SDK which could be used in distributed systems.

Public Interfaces

Client interfaces declaration:

```
class MilvusClient {
 public:
    /**
     * Create a MilvusClient instance.
     *
     * @return std::shared_ptr<MilvusClient>
     */
    static std::shared_ptr<MilvusClient>
    Create();

    /**
     * Connect to Milvus server.
     *
     * @param [in] connect_param server address and port
     * @return Status operation successfully or not
     */
    virtual Status
    Connect(const ConnectParam& connect_param) = 0;

    /**
     * Break connections between client and server.
     *
     * @return Status operation successfully or not
     */
    virtual Status
    Disconnect() = 0;

    /**
     * Create a collection with schema.
     *
     * @param [in] schema schema of the collection
     * @return Status operation successfully or not
     */
    virtual Status
    CreateCollection(const CollectionSchema& schema) = 0;

    /**
     * Check existence of a collection.
     *
     * @param [in] collection_name name of the collection
```

```
     * @param [out] has true: collection exists, false: collection doesn't exist
     * @return Status operation successfully or not
     */
    virtual Status
    HasCollection(const std::string& collection_name, bool& has) = 0;

    /**
     * Drop a collection, with all its partitions, index and segments.
     *
     * @param [in] collection_name name of the collection
     * @return Status operation successfully or not
     */
    virtual Status
    DropCollection(const std::string& collection_name) = 0;

    /**
     * Load collection data into CPU memory of query node.
     * If the timeout is specified, this api will call ShowCollections() to check collection's loading state,
     * waiting until the collection completely loaded into query node.
     *
     * @param [in] collection_name name of the collection
     * @param [in] progress_monitor set timeout to wait loading progress complete, set to ProgressMonitor::
NoWait() to
     * return instantly
     * @return Status operation successfully or not
     */
    virtual Status
    LoadCollection(const std::string& collection_name, const ProgressMonitor& progress_monitor) = 0;

    /**
     * Release collection data from query node.
     *
     * @param [in] collection_name name of the collection
     * @return Status operation successfully or not
     */
    virtual Status
    ReleaseCollection(const std::string& collection_name) = 0;

    /**
     * Get collection description, including its schema.
     *
     * @param [in] collection_name name of the collection
     * @param [out] collection_desc collection's description
     * @return Status operation successfully or not
     */
    virtual Status
    DescribeCollection(const std::string& collection_name, CollectionDesc& collection_desc) = 0;

    /**
     * Get collection statistics, currently only return row count.
     * If the timeout is specified, this api will call Flush() and wait all segmetns persisted into storage.
     *
     * @param [in] collection_name name of the collection
     * @param [in] progress_monitor set timeout to wait flush progress complete, set to ProgressMonitor::
NoWait() to
     * return instantly
     * @param [out] collection_stat statistics of the collection
     * @return Status operation successfully or not
     */
    virtual Status
    GetCollectionStatistics(const std::string& collection_name, const ProgressMonitor& progress_monitor,
                            CollectionStat& collection_stat) = 0;

    /**
     * If the collection_names is empty, list all collections brief informations.
     * If the collection_names is specified, return the specified collection's loading process state.
     *
     * @param [in] collection_names name array of collections
     * @param [out] collections_info brief informations of the collections
     * @return Status operation successfully or not
     */
```

```
    virtual Status
    ShowCollections(const std::vector<std::string>& collection_names, CollectionsInfo& collections_info) = 0;

    /**
     * Create a partition in a collection.
     *
     * @param [in] collection_name name of the collection
     * @param [in] partition_name name of the partition
     * @return Status operation successfully or not
     */
    virtual Status
    CreatePartition(const std::string& collection_name, const std::string& partition_name) = 0;

    /**
     * Drop a partition, with its index and segments.
     *
     * @param [in] collection_name name of the collection
     * @param [in] partition_name name of the partition
     * @return Status operation successfully or not
     */
    virtual Status
    DropPartition(const std::string& collection_name, const std::string& partition_name) = 0;

    /**
     * Check existence of a partition.
     *
     * @param [in] collection_name name of the collection
     * @param [in] partition_name name of the partition
     * @param [out] has true: partition exists, false: partition doesn't exist
     * @return Status operation successfully or not
     */
    virtual Status
    HasPartition(const std::string& collection_name, const std::string& partition_name, bool& has) = 0;

    /**
     * Load specific partitions data of one collection into query nodes.
     * If the timeout is specified, this api will call ShowPartitions() to check partition's loading state,
     * waiting until the collection completely loaded into query node.
     *
     * @param [in] collection_name name of the collection
     * @param [in] partition_names name array of the partitions
     * @param [in] progress_monitor set timeout to wait loading progress complete, set to
     * ProgressMonitor::NoWait() to return instantly
     * @return Status operation successfully or not
     */
    virtual Status
    LoadPartitions(const std::string& collection_name, const std::vector<std::string>& partition_names,
                   const ProgressMonitor& progress_monitor) = 0;

    /**
     * Release specific partitions data of one collection into query nodes.
     *
     * @param [in] collection_name name of the collection
     * @param [in] partition_names name array of the partitions
     * @return Status operation successfully or not
     */
    virtual Status
    ReleasePartitions(const std::string& collection_name, const std::vector<std::string>& partition_names) = 0;

    /**
     * Get partition statistics, currently only return row count.
     * If the timeout is specified, this api will call Flush() and wait all segmetns persisted into storage.
     *
     * @param [in] collection_name name of the collection
     * @param [in] partition_name name of the partition
     * @param [in] progress_monitor set timeout to wait flush progress complete, set to ProgressMonitor::
NoWait() to
     * return instantly
     * @param [out] partition_stat statistics of the partition
     * @return Status operation successfully or not
     */
```

```
    virtual Status
    GetPartitionStatistics(const std::string& collection_name, const std::string& partition_name,
                           const ProgressMonitor& progress_monitor, PartitionStat& partition_stat) = 0;

    /**
     * If the partition_names is empty, list all partitions brief informations.
     * If the partition_names is specified, return the specified partition's loading process state.
     *
     * @param [in] collection_name name of the collection
     * @param [in] partition_names name array of the partitions
     * @param [out] partitions_info brief informations of the partitions
     * @return Status operation successfully or not
     */
    virtual Status
    ShowPartitions(const std::string& collection_name, const std::vector<std::string>& partition_names,
                   PartitionsInfo& partitions_info) = 0;

    /**
     * Create an alias for a collection. Alias can be used in search or query to replace the collection name.
     * For more information: https://wiki.lfaidata.foundation/display/MIL/MEP+10+--+Support+Collection+Alias
     *
     * @param [in] collection_name name of the collection
     * @param [in] alias alias of the partitions
     * @return Status operation successfully or not
     */
    virtual Status
    CreateAlias(const std::string& collection_name, const std::string& alias) = 0;

    /**
     * Drop an alias.
     *
     * @param [in] alias alias of the partitions
     * @return Status operation successfully or not
     */
    virtual Status
    DropAlias(const std::string& alias) = 0;

    /**
     * Change an alias from a collection to another.
     *
     * @param [in] collection_name name of the collection
     * @param [in] alias alias of the partitions
     * @return Status operation successfully or not
     */
    virtual Status
    AlterAlias(const std::string& collection_name, const std::string& alias) = 0;

    /**
     * Create an index on a field. Currently only support index on vector field.
     *
     * @param [in] collection_name name of the collection
     * @param [in] index_desc the index descriptions and parameters
     * @param [in] progress_monitor set timeout to wait index progress complete, set to ProgressMonitor::
NoWait() to
     * return instantly
     * @return Status operation successfully or not
     */
    virtual Status
    CreateIndex(const std::string& collection_name, const IndexDesc& index_desc,
                const ProgressMonitor& progress_monitor) = 0;

    /**
     * Get index descriptions and parameters.
     *
     * @param [in] collection_name name of the collection
     * @param [in] field_name name of the field
     * @param [out] index_desc index descriptions and parameters
     * @return Status operation successfully or not
     */
    virtual Status
    DescribeIndex(const std::string& collection_name, const std::string& field_name, IndexDesc& index_desc) = 0;
```

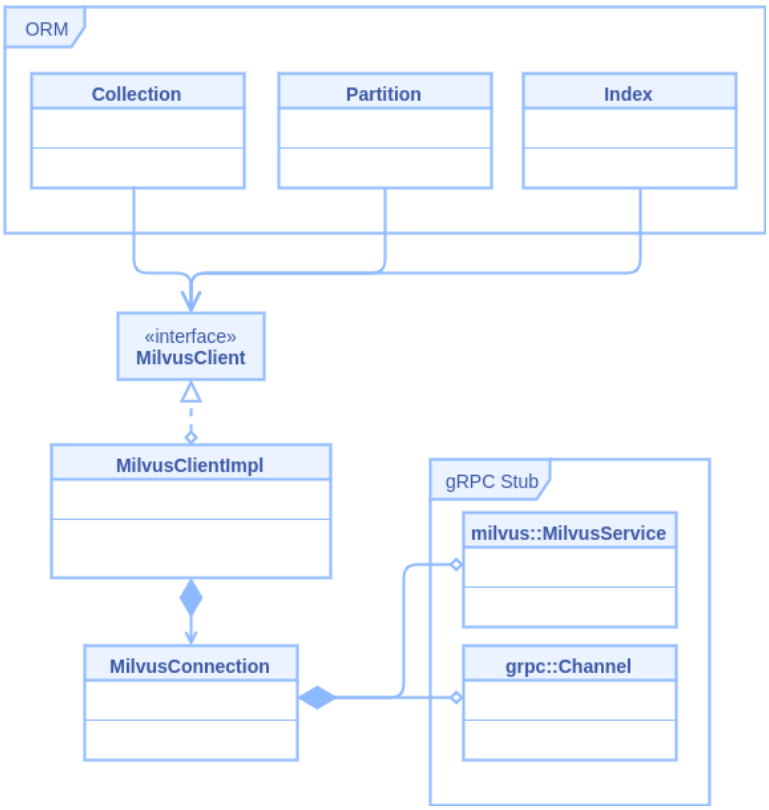
```
    /**
     * Get state of an index. From the state client can know whether the index has finished or in-progress.
     *
     * @param [in] collection_name name of the collection
     * @param [in] field_name name of the field
     * @param [out] state index state of field
     * @return Status operation successfully or not
     */
    virtual Status
    GetIndexState(const std::string& collection_name, const std::string& field_name, IndexState& state) = 0;

    /**
     * Get progress of an index. From the progress client can how many rows have been indexed.
     *
     * @param [in] collection_name name of the collection
     * @param [in] field_name name of the field
     * @param [out] progress progress array of field, currently only return one index progress
     * @return Status operation successfully or not
     */
    virtual Status
    GetIndexBuildProgress(const std::string& collection_name, const std::string& field_name,
                          IndexProgress& progress) = 0;

    /**
     * Drop index of a field.
     *
     * @param [in] collection_name name of the collection
     * @param [in] field_name name of the field
     * @return Status operation successfully or not
     */
    virtual Status
    DropIndex(const std::string& collection_name, const std::string& field_name) = 0;
};
```

Design Details

Project framework

The C++ sdk can be designed as two levels:

- the orm classes: ConnectionInstance/Collection/Partition/Index/Schema/Parameters, and maybe ConnectionPool
- the client implementation: a class to maintain grpc channel, a class to transfer parameters to rpc interface



CI Workflow

Use github ci process to run code lint, clang format check, compile project and run unittest.

Use mergify to automatically add ci-passed label.

Use code coverage tool to generate report and upload to codecov.io.

API Document

Add description for each class/method/constant, follow the Doxygen comment style.

ORM

To be determined.

C++ versions to support

Support C++ version above C++11. The Reason is:

1. Wider user range, as may organizations and devices support C++11.
2. Easy to maintain as it will get a large group of developer support.

OS platform to support

For the supported platform, need to be tested with mainstream distributions(e.g. Ubuntu 18.04+, CentOS 7+) using google tests.

Milvus cpp SDK 1.1 using cmake, and only build a shared library as output. In this new SDK for milvus 2, the user can choose which version to be built by setting cmake options. Quality gates such as clang-format, clang-tidy, cpplint are needed.

Code Style

A basic rule of C++ code style:

- Namespace should use **`lower_case`**
- Class name should use **`CamelCase`**
- `Class member name should use` **`lower_case_`** `(with a underscore append)`
- Enum member name should use **`UPPER_CASE`**
- The static/public Function name should use **`CamelCase`**, and the private/protected member Function name use **`camelBack`**

`For more details, follow the` Google C++ Style Guide`.`

Test Plan

1. Unit test
    a. C++ SDK will implement a mock milvus for basic testing, need to be tested with mainstream distributions(e.g. Ubuntu 18.04+, CentOS 7+)
    b. Start a standalone milvus complicated test.
2. CI test
    a. Do we need to setup basic CI test for further improvement?
3. Examples
    a. finish all the examples in user guide and make sure it works like https://milvus.io/docs/v2.0.0/example_code.md

References

Current state: Accepted

ISSUE: https://github.com/milvus-io/milvus/issues/7713

PRs:

Keywords: C++ SDK

Released: with Milvus 2.1

Authors: @matrixji @ArkaprabhaChakraborty @yhmo