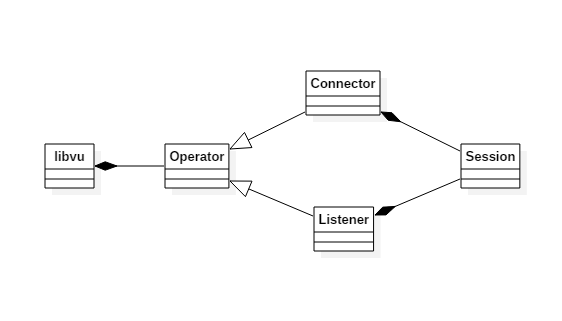
Maraton Framework V3

Design Spec

# Class Diagram



# Macro

List of macros and its functions

## uptr<T>

declare a std::unique\_ptr<T>

## sptr<T>

declare a std::shared\_ptr<T>

## make\_uptr(T,args…)

make a std::unique\_ptr<T>

args is the construction arrangements

## make\_sptr(T,args..)

make a std::shared\_ptr<T>

args is the construction arrangements

## move\_ptr( ptr )

move a uptr or sptr to another owner.

## LOG\_DEBUG( const char\* , args..)

Print a debug string on screen with args

# Utility Classes

Some tool classes.

## AsyncWorker

AsyncWorker will create a thread and do the job.

It is not thread safe, you cannot call network interface during job

### static void create( callback\_t action, callback\_t finish, void\* data)

Create a job.

Action will be invoked when the thread start.

Finish will be invoked when the job was done.

Action is in new thread and it is not thread safe.

Finish is in the main thread and it thread safe.

Data will be passed to the new thread.

## SyncWorker

SyncWorker will create a loop in main thread, so it is thread safe.

### static void create(callback, after\_callback, data)

Callback will not be invoked in main loop until next loop.

If callback return a true, the loop will break otherwise the loop will continue.

## WebClient

A web client. It can send request ( both post and get), upload single file, download single file.

### get( string url , callback\_response\_t callback)

Send a GET request to remote server with assigned url.

### post(string url, string data, callback\_response\_t callback)

Send a POST request to remote server with assigned data

### post\_file(string url,string file\_token,FILE\* file,callback\_response\_t callback)

Post a file to remote server.

File must be opened before invoking this method.

### dl\_file(string url, FILE\* file, callback\_response\_t callback)

Download a file from remote server.

File must be opened before invoking this method

PS: file must be smaller then 2GB.

## Buffer

Memory manage classes.

It consists by a char\* and its length.

Implement copy, move construction.

It can be instanced by char\*, std::string and size

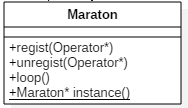
## json

checkout: https://github.com/Yhgenomics/json

# Main Classes

List of classes, include the public member methods and usages.

## Maraton



Maraton which is a Singleton class is used to manage all operators and resolve the domain address to IP address by DNS.

### void regist(Operator\*)

Register an operator, resolve the domain address. Start the function of the operator.

### void unregist(Operator\*)

Unregister an operator, stop functions of the operator.

### void loop()

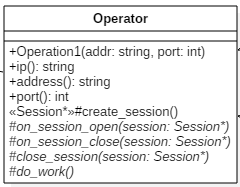
Run the main loop.

It will block the thread until there are not any action in network.

### static Maraton\* instance()

Return the singleton instance for Maraton.

## Operator



Operator is the base class of Connector and Listener. Classes which Inherit from must implement on\_session\_open, on\_session\_close, close\_session,do\_work.

### Construction

addr: server address which will be connected, it can be IP address or domain.

port: connecting port

### string ip()

Return IP address which socket will connect to. It will be filled after DNS query.

### string address()

Return the address which input in construction. It will pass to DNS to resolve the IP address

### int port()

Return the connection port which input in construction.

### Session\* create\_session()

Return the session instance.

The inherits classes must implement it to determine the session type.

### void on\_session\_open( Session \* )

The method will be invoked when new session has been created.

The inherits classes must implement it to manage the session.

### void on\_session\_close( Session \*)

The method will be invoked when a session is closed or destroyed.

The inherits class must implement it to manage the session.

### void close\_session( Session\* )

The method is used to close a session.

It will invoke the on\_session\_close method.

Session should not be closed in other ways.

## Connector

Connector is used to connect the remote server. It inherits from Operator.

The client must implement it. It inherits from Operator.

## Listener

Listener is used to accept connection from clients. It inherits from Operator.

The server must implement it.

## Session

System will create a session when a connection or accepted a new connection.

It is used to manage the connection, send data or receive data.

### void close()

Close the session. The session will close until next loop.

### void send( uptr<Buffer> buf)

Send buffer to remote, the data will be sent until next loop.

### void on\_connect()

The method will be invoked when the session connected to the remote

### void on\_read( uptr<Buffer> buf)

The method will be invoked when received data from remote

### void on\_write( uptr<Buffer>buf)

The method will be invoked when finished writing data to remote

### void on\_close()

The method will be invoked when the session closed.

# Program Guide

## Connect to server

This section will teach you how to connect to remote server.

### Implement the Session

Inherits the session class and implement virtual methods.

* on\_read

callback when the session received data

you can format your data or protocol here

* on\_close

callback when the session was closed

you can free your resources here

* on\_connect

callback when the session was connected.

you can do some initials in the method

### Implement the Connector Class

Inherits the connector class. Implement the absolutely method.

* Construction

you must pass the remote IP address and connecting port to the parent class

* create\_session

return the instance of your session class in 5.1.1

* on\_session\_open

invoke when the session has been created and connected to the remote server

you can add the session to your collections or something else to manage it

* on\_session\_close

invoke when the session was closed.

you can free you resources here.

after the callback, then Maraton::loop will not longer block the main thread if there are any network actions in loop.

### Make it working

The last step is to add your connector to the Maraton by using Maraton::instance()::regist() method.

## Listening a port

This chapter will teach you how to build a listener to listen on a TCP/IP port

### Implement the Session

As the same as 5.1.1.

### Implement the Listener Class

Inherits the connector class. Implement the absolutely method.

* Construction

you must pass the local IP address and listening port to the parent class

* create\_session

return the instance of your session class in 5.2.1

* on\_session\_open

invoke when the session has been created and connected to the remote server

you can add the session to your collections or something else to manage it

* on\_session\_close

invoke when the session was closed.

you can remove the session from your management collection and free you resources here.

after the callback, then Maraton::loop will continue block the main thread.

### Make it working

The last step is to add your connector to the Maraton by using Maraton::instance()::regist() method.