**configure: A C++ simple configuration library**

Author: Yongtian Zhang (yongtianzhang@gmail.com)

Date: 2011.10.23

目录

[Framework 2](#_Toc307144196)

[Basic configuration procedure 2](#_Toc307144197)

[Flow chart 2](#_Toc307144198)

[Configuration file model 3](#_Toc307144199)

[Command line configuration option model 3](#_Toc307144200)

[Reference 3](#_Toc307144201)

[Namespace configure 3](#_Toc307144202)

[Class Configure 4](#_Toc307144203)

[Constructor 4](#_Toc307144204)

[Configuration reader and cleaner 4](#_Toc307144205)

[Configuration getter 4](#_Toc307144206)

[Configuration getter (section support) 5](#_Toc307144207)

[Global standard configure object 6](#_Toc307144208)

[Global convenient functions 6](#_Toc307144209)

[FAQ 7](#_Toc307144210)

[Q: Why I need a global configure object? 7](#_Toc307144211)

# Framework

## Basic configuration procedure

### Flow chart

main()

configure

module1

module2

configure::read(“conf.cfg”)

module1.init()

configure::get<T>(“key1”)

module2.init()

configure::get<T>(“key2”)

**Step1: read configuration from configuration file**

// main.cpp

#include “configure.h”

int main(int argc, char\* argv[]) {

configure::read(“config.cfg”);

// software module code

return 0;

}

**Step2: read configuration value in software module**

// software module

#include “configure.h”

int init() {

int value1 = configure::get<int>(“key1”);

std::string value2 = configure::get<std::string>(“key2”);

// other initialization code

return 0;

}

## Configuration file model

## Command line configuration option model

# Reference

## Namespace configure

This is the namespace of this library, covers all the classes, functions and objects that defined in this library.

## Class Configure

### Constructor

Default constructor, nothing is done when object constructing.

### Configuration reader and cleaner

1. **void configure::Configure::read(const std::string& cfg\_file);**

Read configuration from configuration file /cfg\_file/. If read failed, std::invalid\_argument will be thrown out.

1. **void configure::Configure::read(int argc, char\* argv[]);**

Filter configuration command line option match format “—key=vlaue”.

1. **void configure::Configure::clear();**

clear up all the configuration items stored in the Configure object.

### Configuration getter

1. **template <typename T>**

**T configure::Configure::get(const std::string& key) const;**

Get configuration value of /key/ with specified type /T/. If /key/ is not in the configuration variable table, std::invalid\_argument will be thrown.

Usage:

Configure conf;

conf.read(“config.cfg”);

double value1 = conf.get<double>(“key1”);

1. **template <typename T>**

**T configure::Configure::default\_get(**

**const std::string& key, const T& default\_value) const;**

Get configuration value of /key/ with specified type /T/. If /key/ is not in the configuration variable table, /default\_value/ will be returned.

Usage:

Configure conf;

conf.read(“config.cfg”);

double value1 = conf.default\_get<double>(“key1”, 1.0);

1. **bool configure::Configure::verify(const std::string& key) const;**

Check whether /key/ is in configuration variable table or not, return true if /key/ exists, false otherwise.

### Configuration getter (section support)

1. **template <typename T>**

**T configure::Configure::get(**

**const std::string& section, const std::string& key);**

1. **template <typename T>**

**T configure::Configure::default\_get(const std::string& section,**

**const std::string& key, const T& default\_value);**

1. **bool configure::Configure::verify(**

**const std::string& section, const std::string& key);**

## Global standard configure object

1. **configure::Configure& configure::standard\_configure();**

## Global convenient functions

1. **void configure::read(const std::string& cfg\_file);**
2. **void configure::read(int argc, char\* argv[]);**
3. **void configure::clear();**
4. **template <typename T> T configure::get(const std::string& key);**
5. **template <typename T> T configure::default\_get(const std::string& key, const T& default\_value);**
6. **bool configure::verify(const std::string& key);**
7. **template <typename T> T configure::get(const std::string& section, const std::string& key);**
8. **template <typename T> T configure::default\_get(const std::string& section, const std::string& key, const T& default\_value);**
9. **bool configure::verify(const std::string& section, const std::string& key);**

# FAQ

## Q: Why I need a global configure object?

A: When a software module needs configuration, it usually means just a query of some configuration values by specified keys. My solution is to do the query on a global configure object, and the global configure object is initialized when the program starts up.

Software module need not to know where the configure object is initialized and where the configuration file locates. It just calls for the configuration value it needs.