BUtils Documentation

发布 *0.1.2-beta*

Ball Chang

2019 年 06 月 13 日

开始

CHAPTER 1

欢迎来到 BUtils 的文档。

小技巧: This is stable version. Need development version? Please click here .

Ball Chang 项目之基石。

警告: 本软件不提供任何担保。

1.1 安装

1.1.1 快速上手视频

下面的视频演示了如何构建和快速测试 BUtils。

1.1.2 快速开始

通过以下指令可以获得一个本项目的副本并且可以在你的本地计算机上运行，用于开发和测试。

前置要求

```
C++ Standard: c++11
Build tools: cmake make autoconf automake gcc
```

安装

```
$ git clone https://gitlab.com/zhangbolily/BUtils.git BUtils
$ cd BUtils
$ git submodule update --init --recursive
$ mkdir build
$ cd build
$ cmake ..
$ make
$ make install
```

运行测试

```
$ cd build
$ rm -rf ./*
$ cmake -DBUILD_TESTING=ON -DCODE_COVERAGE=ON -DCMAKE_BUILD_TYPE=Debug ..
$ make tests
$ bin/tests
```

1.2 BTimer

定义在头文件 <BUtils/BTimer> 中

1.2.1 概述

Class BTimer provides repetitive and single-shot timers with a minimum precision of 1 millisecond.

1.2.2 公有类型

enum	*BTimerStatus {Active, Stop}*

1.2.3 属性

bool	*singleShot*
std::chrono::milliseconds	*interval*
std::chrono::milliseconds	*timeout*

1.2.4 公有函数

	BTimer()
	~BTimer()
bool	*operator>(const BTimer& rtimer) const*
bool	*operator<(const BTimer& rtimer) const*
bool	operator=(const BTimer& rtimer) const
void	*start()*
void	*stop()*
bool	*isActive() const*
bool	*isSingleShot() const*
int32	*id() const*
uint32	*interval() const*
uint32	*timeout() const*
void	*reset()*
void	*setActive(bool)*
void	*callOnInterval(std::function timer_action)*
void	*callOnTimeout(std::function timer_action)*
void	*setInterval(uint32 _interval)*
void	*setInterval(std::chrono::milliseconds _interval)*
void	*setTimeout(uint32 _timeout)*
void	*setTimeout(std::chrono::milliseconds _timeout)*
void	*setSingleShot(bool singleshot)*

1.2.5 静态公有函数

uint	precision()
void	setPrecision(uint)

1.2.6 详细描述

Class BTimer provides repetitive and single-shot timers with a minimum precision of 1 millisecond.

BTimer 提供一种简单易用的编程界面，用于在您的应用中执行周期性任务。只需要创建一个定时器 对象，并设置相应的属性，然后启动它。你可以在任何时候改变定时器的属性值。

1 秒的定时器例子：

```
#include <BUtils/BTimer>
#include <unistd.h>
#include <iostream>

void timerAction() {
    std::cout << "I'm timer action." << std::endl;
}

int main() {
    BUtils::BTimer timer;
    timer.callOnTimeout(timerAction);
    timer.setTimeout(1000);
    timer.start();
    // If timer object is destroyed, the timer event do not exist as well.
    // Sleep to make the timeout event occur.
    sleep(2);
}
```

BTimer 的定时器事件系统被设计为可以在多线程环境下工作，但是 BTimer 对象本身并不能做到这一点。不要尝试在线程间共享 BTimer 对象，就在一个线程中创建并使用即可。

精确度和定时器精度

定时器的精确度取决于底层操作系统和硬件设备。在大多数系统平台上，系统时钟普遍可以达到 1 微秒的精确度。

大多数系统平台上，BTimer 支持 1 毫秒的最低精度。但是在定时器系统工作负载非常高（例如非常多的定时器事件）或者高 CPU 使用率（这将导致定时器事件循环不能立刻被操作系统唤醒）的情况下或导致精度不准（一个定时器循环时间大于 1 毫秒）。

1.2.7 成员类型文档

enum BTimerStatus

这个枚举类型被这些函数使用：*isActive() const* and *setActive(bool)* .

常量	值	描述
BTimer::Active	0	定时器被激活
BTimer::Stop	1	定时器停止

1.2.8 属性文档

bool **singleShot**

这个属性值表示是否要在间隔超时发生的时候触发间隔动作。

如果为真，间隔动作会在每个间隔周期后被触发，除非定时器超时。默认值为假。

访问函数:

bool	*isSingleShot() const*
void	*setSingleShot(bool singleshot)*

std::chrono::milliseconds **interval**

这个属性值表示该定时器的间隔周期。在每个间隔周期后，间隔动作会被触发。默认值是 0，代表没有间隔。

访问函数:

uint32	*interval() const*
void	*setInterval(std::chrono::milliseconds)*

std::chrono::milliseconds **timeout**

这个属性值表示该定时器的过期时间（超时时间）。超时之后，该定时器会从定时器系统中移除直到下一个 start 调用。

注解: 默认值是最大的无符号整型，代表该定时器”永不超时”。

访问函数:

uint32	*timeout() const*
void	*setTimeout(std::chrono::milliseconds)*

1.2.9 成员函数文档

explicit *BTimer*::**BTimer**() **noexcept**

构造一个 BTimer 对象。

BTimer::~BTimer()

析构一个 BTimer 对象。

bool **BTimer::operator>**(**const** BTimer &*rtimer*) **const**

如果 id 大于 rtimer’s id 则返回真。

bool **BTimer::operator<**(**const** BTimer &*rtimer*) **const**

如果 id 小于 rtimer’s id 则返回真。

bool **BTimer::operator==**(**const** BTimer &*rtimer*) **const**

如果 id 等于 rtimer’s id 则返回真。

void **BTimer::start**()

开启该定时器；如果超时时间设置为 0 则该函数不做任何事情。

void **BTimer::stop**()

停止该定时器；如果定时器已经过期则该函数不做任何事情。

bool **BTimer::isActive**() **const**

如果定时器正在运行则返回真。

bool **BTimer::isSingleShot**() **const**

如果间隔事件只触发一次则返回真。

int32 **BTimer::id**() **const**

返回定时器的 id。

uint32 **BTimer::interval**() **const**

返回定时器超时间隔周期，以毫秒为单位。

uint32 **BTimer::timeout**() **const**

返回定时器超时时间，以毫秒为单位。

void **BTimer::reset**()

重置该定时器的所有属性值（除了定时器 id）为默认值并停止该定时器。

void **BTimer::setActive**(bool *_active*)

用户调用则该函数不做任何事情。

void **BTimer::callOnInterval**(std::function<void)
> *timer_action*

设置超时间隔之后要触发的动作。

void **BTimer::callOnTimeout**(std::function<void)
> *timer_action*

设置超时之后要触发的动作。

void **BTimer::setInterval**(uint32 *_interval*)

void **BTimer::setInterval**(std::chrono::milliseconds *_interval*)

设置超时间隔，以毫秒为单位。默认值是 0。

void **BTimer::setTimeout**(uint32 *_timeout*)

void **BTimer::setTimeout**(std::chrono::milliseconds *_timeout*)

设置超时时间，以毫秒为单位。默认值是最大的无符号整型。

void **BTimer::setSingleShot**(bool *singleshot*)

如果 singleshot 为真，则间隔动作只会被触发一次。

static uint **BTimer::precision**()

返回定时器的精度，以毫秒为单位。默认值是 1 毫秒。

static void **BTimer::setPrecision**(uint)

设置定时器的精度，以毫秒为单位。

1.3 BTiming

定义在头文件 <BUtils/BTiming> 中

1.3.1 概述

BTiming 类提供一个最小精度为 1 微秒的计时系统用于记录时间。

1.3.2 公有类型

enum	*BTimingStatus {CPUTiming, Timing, Stop}*

1.3.3 公有函数

	BTiming() noexcept
	~BTiming()
void	*start()*
void	*stop()*
void	*startCPUTiming()*
void	*stopCPUTiming()*
bool	*isActive()*
int64	*time() const*
int64	*CPUTime() const*

1.3.4 详细描述

BTiming 类提供一个最小精度为 1 微秒的计时系统用于记录时间。

BTiming 能够以 1 微秒的精度记录*真实时间* 和*CPU 时间*。

1 秒钟计时的例子：

```
#include <BUtils/BTiming>
#include <unistd.h>
#include <iostream>

int main() {
    BUtils::BTiming timing;
    BUtils::BTiming CPUTiming;

    timing.start();
    CPUTiming.startCPUTiming();
    sleep(1);
    timing.stop();
    CPUTiming.stopCPUTiming();

    std::cout << "Real time is: " << timing.time() << " us" << std::endl;
    std::cout << "CPU time is: " << CPUTiming.CPUTime() << " us" << std::endl;
}
```

1.3.5 成员类型文档

enum BTimingStatus

常量	值	描述
BTiming::CPUTiming	0	代表正在记录 CPU 时间
BTiming::Timing	1	代表正在记录真实时间
BTiming::Stop	2	停止计时

1.3.6 成员函数文档

BTiming::**BTiming() noexcept**

构造一个 BTiming 对象。

BTiming::~BTiming()

析构一个 BTiming 对象。

void **BTiming::start()**

开始记录真实时间。如果 startCPUTiming 被调用，则调用此函数不做任何事。

void **BTiming::stop()**

停止记录真实时间。如果 startCPUTiming 被调用，则调用此函数不做任何事。

void **BTiming::startCPUTiming()**

开始记录 CPU 时间。如果 start 被调用，则调用此函数不做任何事。

void **BTiming::stopCPUTiming()**

停止记录 CPU 时间。如果 start 被调用，则调用此函数不做任何事。

bool **BTiming::isActive()**

如果正在计时则返回真。

int64 **BTiming::time() const**

以毫秒为单位返回通过 start 和 stop 函数调用记录的真实时间，其他情况下返回 0。

int64 **BTiming::CPUTime() const**

以毫秒为单位返回通过 startCPUTiming 和 stopCPUTiming 函数调用记录的 CPU 时间，其他情况下返回 0。

1.4 BUtils

定义在头文件 <BUtils/BUtils> 中

1.4.1 概述

名称空间 BUtils 提供了非常多的实用函数。

1.4.2 公有类型

enum	None

1.4.3 公有函数

std::string	*generateUUID4()*
bool	*isUUID4(const std::string &_uuid)*

1.4.4 详细描述

名称空间 BUtils 提供了非常多的实用函数。

1.4.5 成员函数文档

std::string **generateUUID4()**

返回一个版本 4 的 UUID（基于随机值生成）。

bool **isUUID4(const** std::string &*_uuid*)

如果输入的 UUID 是有效的则返回真。

1.5 术语表

真实时间 代表真实世界的时间，例如 24 小时或 1 天。

CPU 时间 操作系统分配的用于计算的时间。这个时间可以大于（在多线程程序中）、等于或小于真实时间。

计时 记录时间。

定时器 一种可以根据特定配置来触发事件的特殊时钟。

CHAPTER 2

索引

- genindex
- modindex
- search

索引

B

C

G

I

S

T