CAPI接口(cpp)

C++接口必看！！！

在此鸣谢\xfgg/\xfgg/\xfgg/，看到这里的选手可以到选手群膜一膜！！！

除非特殊指明，以下代码均在 MSVC 19.28.29913 /std:c++17 与 g++ 10.2 for linux -std=c++17 两个平台下通过。

由于我们的比赛最终会运行在Linux平台上，因此程设课上学到的一些只适用于Windows的C++操作很可能并不能正确执行。此外，代码中使用了大量Modern C++中的新特性，可能会使选手在编程过程中遇到较大困难。因此，此处介绍一些比赛中使用C++接口必须了解的知识。

异步接口std::future的使用

本届比赛中，我们可能会看到类似 `std::future<bool>` 这样类型的接口返回值，这实际上是一个异步接口。在调用同步接口后，在接口内的函数未执行完之前，线程通常会阻塞住；但是异步接口的调用通常不会阻塞当前线程，而是会另外开启一个线程进行操作，当前线程则继续向下执行。当调用 `get()` 方法时，将返回异步接口的值，若此时异步接口内的函数依然未执行完，则会阻塞当前线程。

如果不需要返回值或没有返回值，但是希望接口内的函数执行完之后再进行下一步，即将接口当做常规的同步接口来调用，也可以调用 `wait()` 方法。

```cpp
#include <iostream>
#include <thread>
#include <future>
#include <chrono>

int f_sync()
{
    std::this_thread::sleep_for(std::chrono::seconds(1));
    return 8;
}

std::future<int> f_async()
{
    return std::async(std::launch::async, []()
                      { std::this_thread::sleep_for(std::chrono::seconds(1));
                        return 8; });
}

int main()
{
    auto start = std::chrono::system_clock::now();
    std::cout << std::chrono::duration_cast<std::chrono::duration<double,
std::milli>>(std::chrono::system_clock::now() - start).count() << std::endl;
    auto x = f_async();
    std::cout << std::chrono::duration_cast<std::chrono::duration<double,
std::milli>>(std::chrono::system_clock::now() - start).count() << std::endl;
    std::cout << x.get() << std::endl;
    std::cout << std::chrono::duration_cast<std::chrono::duration<double,
std::milli>>(std::chrono::system_clock::now() - start).count() << std::endl;
    auto y = f_sync();
    std::cout << std::chrono::duration_cast<std::chrono::duration<double,
std::milli>>(std::chrono::system_clock::now() - start).count() << std::endl;
    std::cout << y << std::endl;
    std::cout << std::chrono::duration_cast<std::chrono::duration<double,
std::milli>>(std::chrono::system_clock::now() - start).count() << std::endl;
}
```

计时相关

编写代码过程中，我们可能需要获取系统时间等一系列操作，C++ 标准库提供了这样的行为。尤其注意**不要**使用 Windows 平台上的 `GetTickCount` 或者 `GetTickCount64`！！！ 应当使用 `std::chrono`

头文件：`#include <chrono>`

可以用于获取时间戳，从而用于计时、例如计算某个操作花费的时间，或者协调队友间的合作。

```
#include <iostream>
#include <chrono>
int main()
{
    auto sec = std::chrono::duration_cast<std::chrono::seconds>
(std::chrono::system_clock::now().time_since_epoch()).count();
    auto msec = std::chrono::duration_cast<std::chrono::milliseconds>
(std::chrono::system_clock::now().time_since_epoch()).count();
    std::cout << "从 1970 年元旦到现在的：秒数" << sec << "；毫秒数：" << msec <<
std::endl;
    return 0;
}
```

线程睡眠

由于移动过程中会阻塞人物角色，因此玩家可能要在移动后让线程休眠一段时间，直到移动结束。C++ 标准库中使线程休眠需要包含头文件：`#include <thread>`。示例用法：

```
std::this_thread::sleep_for(std::chrono::milliseconds(20));     // 休眠 20 毫秒
std::this_thread::sleep_for(std::chrono::seconds(2));           // 休眠 2 秒

// 下面这个也能休眠 200 毫秒
std::this_thread::sleep_until(std::chrono::system_clock::now() +=
std::chrono::milliseconds(200));
```

休眠过程中，线程将被阻塞，而不继续进行，直到休眠时间结束方继续向下执行。

`auto`类型推导

C++11开始支持使用 `auto` 自动推导变量类型，废除了原有的作为 storage-class-specifier 的作用：

```cpp
int i = 4;
auto x = i; // auto 被推导为 int，x 是 int 类型
auto& y = i; // auto 仍被推导为 int，y 是 int& 类型
auto&& z = i; // auto 被推导为 int&，z 是 int&&&，被折叠为 int&，即 z 与 y 同类型
auto&& w = 4; // auto 被推导为 int，w 是 int&& 类型
```

STL相关

std::vector

头文件：`#include <vector>`，类似于可变长的数组，支持下标运算符 `[]` 访问其元素，此时与 C 风格数组用法相似。支持 `size` 成员函数获取其中的元素数量。

创建一个 `int` 型的 `vector` 对象：

```cpp
std::vector<int> v { 9, 1, 2, 3, 4 };   // 初始化 vector 有五个元素，v[0] = 9, ...
v.emplace_back(10);          // 向 v 尾部添加一个元素，该元素饿构造函数的参数为 10（对于int，只有一个语法意义上的构造函数，无真正的构造函数），即现在 v 有六个元素，v[5] 的值是10
v.pop_back();                // 把最后一个元素删除，现在 v 还是 { 9, 1, 2, 3, 4 }
```

遍历其中所有元素的方式：

```cpp
// std::vector<int> v;
for (int i = 0; i < (int)v.size(); ++i)
{
    /*可以通过 v[i] 对其进行访问*/
}

for (auto itr = v.begin(); itr != v.end(); ++itr)
{
    /*
    * itr 作为迭代器，可以通过其访问 vector 中的元素。其用法与指针几乎完全相同。
    * 可以通过 *itr 得到元素；以及 itr-> 的用法也是支持的
    * 实际上它内部就是封装了指向 vector 中元素的指针
    * 此外还有 v.cbegin()、v.rbegin()、v.crbegin() 等
    * v.begin()、v.end() 也可写为 begin(v)、end(v)
    */
}

for (auto&& elem : v)
{
    /*
    * elem 即是 v 中每个元素的引用，也可写成 auto& elem : v
    * 它完全等价于：
    * {
```

```
    *   auto&& __range = v;
    *   auto&& __begin = begin(v);
    *   auto&& __end = end(v);
    *   for (; __begin != __end; ++__begin)
    *   {
    *       auto&& elem = *__begin;
    *       // Some code
    *   }
    * }
    */
}
```

例如：

```
for (auto elem&& : v) { std::cout << elem << ' '; }
std::cout << std::endl;
```

作为 STL 的容器之一，其具有容器的通用接口。但是由于这比较复杂，在此难以一一展开。有兴趣的同学可以在下方提供的链接里进行查阅。

注：请千万不要试图使用 `std::vector<bool>`，若需使用，请用 `std::vector<char>` 代替！

更多用法参见（点击进入）：cppreference vector

std::array

头文件：`#include <array>`，C 风格数组的类封装版本。

用法与 C 风格的数组是基本相似的，例如：

```
std::array<double, 5> arr { 9.0, 8.0, 7.0, 6.0, 5.0 };
std::cout << arr[2] << std::endl;   // 输出 7.0
```

同时也支持各种容器操作：

```
double sum = 0.0;
for (auto itr = begin(arr); itr != end(arr); ++itr)
{
    sum += *itr;
}
// sum 结果是 35
```

更多用法参见（点击进入）：cppreference array。

接口解释

主动指令

移动

- `std::future<bool> Move(int64_t timeInMilliseconds, double angleInRadian)`:移动，`timeInMilliseconds` 为移动时间，单位毫秒；`angleInRadian` 表示移动方向，单位弧度，使用极坐标，**竖直向下方向为x轴，水平向右方向为y轴**
- `std::future<bool> MoveRight(uint32_t timeInMilliseconds)` 即向右移动,`MoveLeft`、`MoveDown`、`MoveUp` 同理

使用技能

- `std::future<bool> UseSkill(int32_t skillID)`:使用对应序号的主动技能

人物

- `std::future<bool> EndAllAction()`:可以使不处在不可行动状态中的玩家终止当前行动

攻击

- `std::future<bool> Attack(double angleInRadian)`:`angleInRadian` 为攻击方向

学习与毕业

- `std::future<bool> StartLearning()`:在教室里开始做作业
- `std::future<bool> StartOpenGate()`:开始开启校门
- `std::future<bool> Graduate()`:从开启的校门或隐藏校门毕业。

勉励与唤醒

- `std::future<bool> StartEncourageMate(int64_t mateID)`:勉励对应玩家ID的学生。
- `std::future<bool> StartRouseMate(int64_t mateID)`：唤醒对应玩家ID的沉迷的学生。

地图互动

- `std::future<bool> OpenDoor()`:开门
- `std::future<bool> CloseDoor()`:关门
- `std::future<bool> SkipWindow()`:翻窗
- `std::future<bool> StartOpenChest()`:开箱

道具

- `bool PickProp(THUAI6::PropType prop)` 捡起与自己处于同一个格子（cell）的道具。
- `bool UseProp(THUAI6::PropType prop)` 使用对应类型的道具
- `bool ThrowProp(THUAI6::PropType prop)` 将对应类型的道具扔在原地

信息获取

队内信息

- `std::future<bool> SendMessage(int64_t, std::string)`：给同队的队友发送消息。第一个参数指定发送的对象，第二个参数指定发送的内容，不得超过256字节。
- `bool HaveMessage()`:是否有队友发来的尚未接收的信息。
- `std::pair<int64_t, std::string> GetMessage()`:从玩家ID为第一个参数的队友获取信息。

查询可视范围内的信息

- `std::vector<std::shared_ptr<const THUAI6::Student>> GetStudents() const`：返回所有可视学生的信息。
- `std::vector<std::shared_ptr<const THUAI6::Tricker>> GetTrickers() const`：返回所有可视捣蛋鬼的信息。
- `std::vector<std::shared_ptr<const THUAI6::Prop>> GetProps() const`：返回所有可视道具的信息。
- `std::vector<std::shared_ptr<const THUAI6::Bullet>> GetBullets() const`：返回所有可视子弹（攻击）的信息。

查询特定位置物体的信息

下面的 CellX 和 CellY 指的是地图格数，而非绝对坐标。

- `THUAI6::PlaceType GetPlaceType(int32_t cellX, int32_t cellY)`：返回某一位置场地种类信息。场地种类详见 structure.h 。
- `bool IsDoorOpen(int32_t cellX, int32_t cellY) const`:查询特定位置门是否开启
- `int32_t GetChestProgress(int32_t cellX, int32_t cellY) const`:查询特定位置箱子开启进度
- `int32_t GetGateProgress(int32_t cellX, int32_t cellY) const`:查询特定位置校门开启进度
- `int32_t GetClassroomProgress(int32_t cellX, int32_t cellY) const`:查询特定位置教室作业完成进度
- `THUAI6::HiddenGateState GetHiddenGateState(int32_t cellX, int32_t cellY) const`：:查询特定位置隐藏校门状态
- `int32_t GetDoorProgress(int32_t cellX, int32_t cellY) const`:查询特定位置门开启状态

其他

- `std::shared_ptr<const THUAI6::GameInfo> GetGameInfo() const`:查询当前游戏状态\
- `std::vector<int64_t> GetPlayerGUIDs() const`:获取所有玩家的GUID\
- `int GetFrameCount() const`:获取目前所进行的帧数\
- `std::shared_ptr<const THUAI6::Tricker> GetSelfInfo() const`或`std::shared_ptr<const THUAI6::Student> GetSelfInfo() const`：获取自己的信息

- `std::vector<std::vector<THUAI6::PlaceType>> GetFullMap() const`：返回整张地图的地形信息。

辅助函数

`static inline int CellToGrid(int cell) noexcept`:将地图格数 cell 转换为绝对坐标grid。

`static inline int GridToCell(int grid) noexcept`:将绝对坐标 grid 转换为地图格数cell。

下面为用于DEBUG的输出函数，选手仅在开启Debug模式的情况下可以使用

```
    void Print(std::string str) const;
    void PrintStudent() const;
    void PrintTricker() const;
    void PrintProp() const;
    void PrintSelfInfo() const;
```

接口一览

```
    // 指挥本角色进行移动，`timeInMilliseconds` 为移动时间，单位为毫秒；`angleInRadian` 表示移动的方向，单位是弧度，使用极坐标--竖直向下方向为 x 轴，水平向右方向为 y 轴
    virtual std::future<bool> Move(int64_t timeInMilliseconds, double angleInRadian) = 0;

    // 向特定方向移动
    virtual std::future<bool> MoveRight(int64_t timeInMilliseconds) = 0;
    virtual std::future<bool> MoveUp(int64_t timeInMilliseconds) = 0;
    virtual std::future<bool> MoveLeft(int64_t timeInMilliseconds) = 0;
    virtual std::future<bool> MoveDown(int64_t timeInMilliseconds) = 0;

    // 捡道具、使用技能
    virtual std::future<bool> PickProp(THUAI6::PropType prop) = 0;
    virtual std::future<bool> UseProp(THUAI6::PropType prop) = 0;
    virtual std::future<bool> UseSkill(int32_t skillID) = 0;
    virtual std::future<bool> Attack(double angleInRadian) = 0;

    virtual std::future<bool> OpenDoor() = 0;
    virtual std::future<bool> CloseDoor() = 0;
    virtual std::future<bool> SkipWindow() = 0;
    virtual std::future<bool> StartOpenGate() = 0;
    virtual std::future<bool> StartOpenChest() = 0;
    virtual std::future<bool> EndAllAction() = 0;

    // 发送信息、接受信息，注意收消息时无消息则返回nullopt
    virtual std::future<bool> SendMessage(int64_t, std::string) = 0;
    [[nodiscard]] virtual bool HaveMessage() = 0;
    [[nodiscard]] virtual std::pair<int64_t, std::string> GetMessage() = 0;

    // 等待下一帧
    virtual std::future<bool> Wait() = 0;

```

```cpp
    // 获取视野内可见的学生/捣蛋鬼的信息
    [[nodiscard]] virtual std::vector<std::shared_ptr<const THUAI6::Student>>
GetStudents() const = 0;
    [[nodiscard]] virtual std::vector<std::shared_ptr<const THUAI6::Tricker>>
GetTrickers() const = 0;

    // 获取视野内可见的道具信息
    [[nodiscard]] virtual std::vector<std::shared_ptr<const THUAI6::Prop>>
GetProps() const = 0;

    // 获取地图信息，视野外的地图统一为Land
    [[nodiscard]] virtual std::vector<std::vector<THUAI6::PlaceType>>
GetFullMap() const = 0;
    [[nodiscard]] virtual THUAI6::PlaceType GetPlaceType(int32_t cellX, int32_t
cellY) const = 0;

    [[nodiscard]] virtual bool IsDoorOpen(int32_t cellX, int32_t cellY) const =
0;
    [[nodiscard]] virtual int32_t GetChestProgress(int32_t cellX, int32_t
cellY) const = 0;
    [[nodiscard]] virtual int32_t GetGateProgress(int32_t cellX, int32_t cellY)
const = 0;
    [[nodiscard]] virtual int32_t GetClassroomProgress(int32_t cellX, int32_t
cellY) const = 0;
    [[nodiscard]] virtual THUAI6::HiddenGateState GetHiddenGateState(int32_t
cellX, int32_t cellY) const = 0;
    [[nodiscard]] virtual int32_t GetDoorProgress(int32_t cellX, int32_t cellY)
const = 0;

    [[nodiscard]] virtual std::shared_ptr<const THUAI6::GameInfo> GetGameInfo()
const = 0;

    // 获取所有玩家的GUID
    [[nodiscard]] virtual std::vector<int64_t> GetPlayerGUIDs() const = 0;

    // 获取游戏目前所进行的帧数
    [[nodiscard]] virtual int GetFrameCount() const = 0;

    /*****选手可能用的辅助函数*****/

    // 获取指定格子中心的坐标
    [[nodiscard]] static inline int CellToGrid(int cell) noexcept
    {
        return cell * numOfGridPerCell + numOfGridPerCell / 2;
    }

    // 获取指定坐标点所位于的格子的 x 序号
    [[nodiscard]] static inline int GridToCell(int grid) noexcept
    {
        return grid / numOfGridPerCell;
    }

```

```
    // 用于DEBUG的输出函数，选手仅在开启Debug模式的情况下可以使用

    virtual void Print(std::string str) const = 0;
    virtual void PrintStudent() const = 0;
    virtual void PrintTricker() const = 0;
    virtual void PrintProp() const = 0;
    virtual void PrintSelfInfo() const = 0;
};

class IStudentAPI : public IAPI
{
public:
    /*****学生阵营的特定函数*****/

    virtual std::future<bool> StartLearning() = 0;
    virtual std::future<bool> StartEncourageMate(int64_t mateID) = 0;
    virtual std::future<bool> StartRouseMate(int64_t mateID) = 0;
    virtual std::future<bool> Graduate() = 0;
    [[nodiscard]] virtual std::shared_ptr<const THUAI6::Student> GetSelfInfo()
const = 0;
};

class ITrickerAPI : public IAPI
{
public:
    /*****捣蛋鬼阵营的特定函数*****/

    [[nodiscard]] virtual std::shared_ptr<const THUAI6::Tricker> GetSelfInfo()
const = 0;
};
```