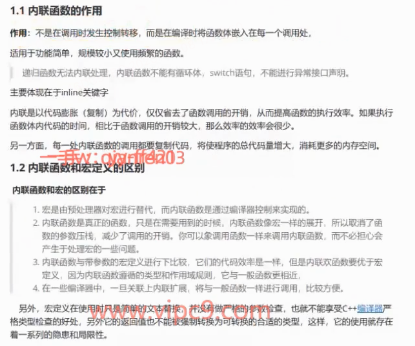
Endl是换行，否则没有\n

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成



内联函数用于频繁使用且代码短的函数，像宏一样，减少入栈出栈操作。

内联函数取址没有意义。

重载

实参一定写在前面，默认参数一定写在后面

参数没有变量名，意味在函数中用不到int=0

参数个数一样，类型不一样都算重载

构造函数与析构函数

Malloc需要确定大小。New不用，用delete释放内存。



图片包含 图形用户界面

描述已自动生成

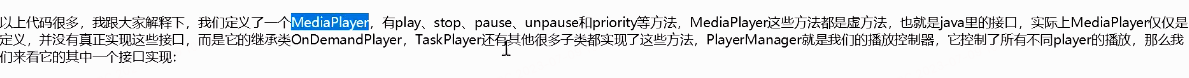
多态：定义一个接口，用不同的函数实现。同一种消息作用到不同的对象带来不一样的行为。

图示

描述已自动生成

原先playmanager依赖taskplayer和ondemandplayer，现在以来mediaplayer。变得单纯了

有助于扩展。



指针运算符重载，C++中有智能指针std::shared\_ptr<Person> p1(new Person(“name”))

赋值运算符重载

[]运算符重载

重载

文本

低可信度描述已自动生成

继承,C++可以多继承



派生的方式中，private继承和public继承一样，能访问public和protected，不能访问private

在类外访问中，只有public继承才能访问父类public成员，protect和private都不能访问任何父类成员。

子类构造父类的构造方法，用:Parent()，相当于java中super()

Virtual修饰函数，未实现的类，相当于java中abstract

虚函数（必须有函数体，必须子类实现）：

Virtual void func(){

//声明函数体

}

纯虚函数：

Virtual int func1()=0;

继承本质：子类copy父类

函数模板

Template<typename/class T>

String s(“hello”);

s.c\_str();//string转化为char\*

音视频C++：

1. 头文件include，源文件src/source
2. .gitignore忽略文件的文本编写

只保留.c .h .txt文件，并忽略其他文件的Git仓库，需要在项目根目录下创建一个名为.gitgnore的文本文件，添加以下规则：

# ignore all files

\*

# allow specific file types

!\*.c

!\*.h

!\*.txt

含义是：

\*忽略所有问价

！\*.c不忽略.c文件

！\*.h不忽略.h文件

！\*.txt不忽略.txt文件

需要单独忽略某路径下所有文件，要写具体路径

Git忽略规则是递归的，如果某个目录下的文件或文件夹匹配了忽略规则，那么该目录下所有子目录和子文件都会被忽略

图形用户界面, 文本

描述已自动生成

1. Git配置中的CRLF、LF、CR

CRLF：Carrriage-Return Line-Feed回车换行，即\r\n

LF：Line-Feed \n

CR：Carriage-Return \r



1. .gitattributes文件是一个文本文件，文件中一行定义一个路径的若干属性

（1）格式：要匹配的文件模式 属性1 属性2 …

（2）4种状态：

* 设置text
* 不设置-text
* 设置值text=string
* 未声明，不出现该属性即可，但为了覆盖其他文件中的声明，可以!text

表格

中度可信度描述已自动生成

第1行，对任何文件，设置text=auto，表示文件的行尾自动转换。如果是文本文件，则在文件入Git库时，行尾自动转换为LF。如果已经在入Git库中的文件的行尾为CRLF，则该文件在入Git库时，不再转换为LF。

第2行，对于txt文件，标记为文本文件，并进行行尾规范化。

第3行，对于jpg文件，标记为非文本文件，不进行任何的行尾转换。

第4行，对于vcproj文件，标记为文本文件，在文件入Git库时进行规范化，即行尾为LF。但是在检出到工作目录时，行尾自动转换为CRLF。

第5行，对于sh文件，标记为文本文件，在文件入Git库时进行规范化，即行尾为LF。在检出到工作目录时，行尾也不会转换为CRLF（即保持LF）。

第6行，对于py文件，只针对工作目录中的文件，行尾为LF。

（3）在一个Git库中可以有多个gitattributes文件，也可以为所有Git库设置统一的gitattributes文件

（4）属性

* text控制行尾的规范性
* eol设置行末字符
* ident
* filter
* diff
* merge
* whitespace
* export-ignore, export-subst
* delta
* encoding

1. std::thread

（1）std::thread构造函数

* 默认构造函数 thread() noexcept;创建一个空的std::thread执行对象
* 初始化构造函数template <class Fn, class… Args>创建一个std::thread对象，该对象可以被joinable，新产生的线程会调用fn函数，args是该函数的参数
* 拷贝构造函数thread(const thread&)=delete;
* Move构造函数 thread(thread&& x) noexcept;调用成功后x不代表任何std::thread执行对象

可被joinable的std::thread对象必须在他们销毁之前被主线程join或者将其设置为detached

（2）get\_id()

（3）joinable()检查线程是否可被join

（4）detach()将当前线程对象所代表的执行实例与该线程对象分离，使得线程的执行可以单独进行。一旦线程执行完毕，它所分配的资源将会被释放。‘

如果需要等待线程完成才能继续，就使用join等待。如果需要立刻返回继续其他操作，就调用detach()来脱离对线程的管理。两者必须选一个。

（5）yield()当前线程放弃执行，操作系统调度另一线程继续执行

（6）sleep\_until()线程休眠至某个时刻time point，该线程才被唤醒

（7）sleep\_for()线程休眠某个指定的时间片time span，该线程才被唤醒

1. 对象声明周期的管理
2. new和delete对象
3. 智能指针

避免内存泄漏的技术。采用RAII特性，在对象构造时获取资源，接着控制对资源的访问使之在对象声明周期内始终保持有效，最后在对象析构的时候释放资源。

不需要显示地释放资源。对象所需的资源在其生命周期内始终保持有效。

C++的智能指针在#include <memory>头文件中

Std::auto\_ptr

Std::unique\_ptr防拷贝

Std::shared\_ptr

1. 项目中路径名管理
2. 用户运行时传入
3. 相对路径
4. CMakeList.txt中使用cmake提供的宏，适配用户编译环境下的路径配置信息