# Ether设计文档

## 模块图



图1-1 整体模块图

CoreEngine模块是Ether引擎的核心工作模块，App模块需要调用CoreEngine模块提供的各种接口来实现用户的各种需求。

Base模块提供各种基础功能，包括程序入口，智能指针封装各种必备的宏定义等等。

Log模块对第三方日志库Spdlog进行封装与日志输出格式配置，提供TRACE/INFO/WARN/DEBUG/ERROR/CRITICAL等六个级别的日志输出。

Application模块负责实现主题循环与创建App程序对象实例，是CoreEngine的核心模块之一。

## 类图

### 2.1 Application类图



图2-1-1 Application模块类图



图2-1-2 Application类成员(2023-03-27)

App类继承自Application类，是被实际使用的程序实例的类型，需要在主循环开启之前被创建，创建App实例的接口被定义在CoreEngine模块中，由App模块实现。

Application类中的Run()方法是引擎程序的主循环，该方法被Base模块的程序入口调用。

Log类型提供了各个级别的日志接口，共App类和Application类调用。

Application类负责分发处理窗体模块传递过来的各种事件。

### 2.2 阻塞事件系统类图

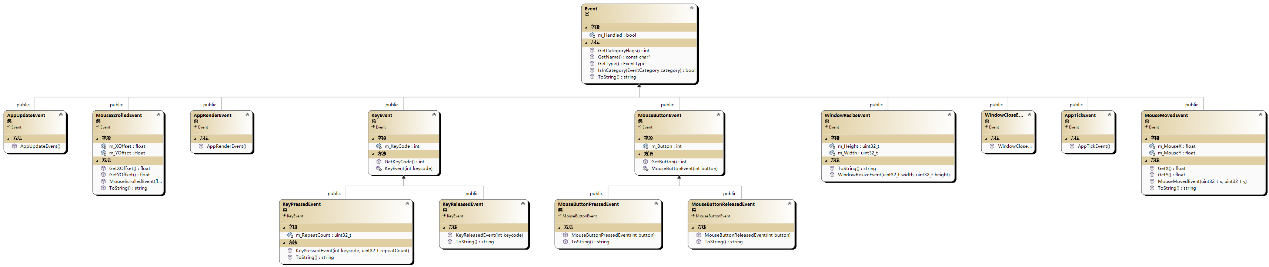


图2-2-1 阻塞事件系统类图

所有的事件系统都继承于Event类，该类是个抽象类，被多种具体事件类型继承。

### 2.3 窗体类图



图2-3-1 窗体模块类图(2023-03-27)

Window类是一个抽象类，包含了几个纯虚函数；Create函数负责判断平台特性然后选定平台特有窗体类进行具体的构建，比如在Windows操作系统上就创建WindowsWindow类的窗体。

WindowsWindow是Windows操作系统特有的针对Windows平台下的GLFW库的封装，保存了窗体的信息（包括宽高以及事件回调函数），实现了Window抽象类提供的所有的纯虚函数，负责捕捉事件并发送给Application类。

## 流程



图3-1 App主循环开启流程

如图3-1所示是主循环开启流程，整个流程由Base模块负责执行，首先初始化并加载日志模块，包括日志的命名、日志跟踪级别的设置（默认为TRACE）以及日志输出格式的设置；接下来调用Application模块的实例创造接口创建出App类的实例对象；对用调用App类的实例对象的Run函数开启主循环。



图3-2 事件处理流程

图3-2为事件处理流程，Application类包含Window类型的数据成员，Application类在构造函数中创建特定平台的Window子类对象并指定自己的OnEvent成员函数为事件分发函数，Window子类在自身的构造函数中设置事件回调函数，并在回调函数中构建事件类的对象，将对象作为参数传递给OnEvent事件分发器，事件分发器根据事件对象的类型选择不同的事件处理函数。