

# 日志系统

对于任何运行在生产环境中的服务器应用程序来说，日志系统都扮演着至关重要的角色。



# 日志系统

**日志系统**是用来记录服务器运行状态、用户行为、请求和错误等事件的系统。

日志系统通过生成日志文件，将这些事件按照**时间顺序**保存下来，方便**监控、分析和排查问题**

# 日志系统的主要功能

- **记录访问日志：**记录客户端的请求信息，包括请求时间、IP 地址、请求 URL、请求方法（如 GET/POST）、响应状态码等。
- **错误日志：**记录服务器运行中出现的错误或异常，帮助开发者快速定位问题。
- **性能监控：**记录服务器性能数据，如请求处理时间、资源使用率等，便于优化性能。
- **安全审计：**记录用户登录、访问控制和安全相关事件，用于安全审查和异常行为监控。



# HP Insight Diagnostics



SmartStart

[Survey](#) [Diagnose](#) [Test](#) [Status](#) [Log](#) [Help](#)

## Test Status

[Reload](#) [About](#)

### Hardware Diagnosis

Testing Completed!

100%

[Retest](#)[Show All Results](#)

#### Device Diagnosis for Power Supply 1

Power Supply Serial Number: 5ANLF0CHL2S205

PassedAccumulated power-on time for this power supply: 667 days

No current failure conditions were detected on this power supply. If you feel you are experiencing power supply issues, refer to the *HP ProLiant Servers Troubleshooting Guide* for more information. For more information, contact HP customer support.

#### Device Diagnosis for Power Supply 2

Power Supply Serial Number: 5ANLE0CLL0WCC0

PassedAccumulated power-on time for this power supply: 457 days

No current failure conditions were detected on this power supply. If you feel you are experiencing power supply issues, refer to the *HP ProLiant Servers Troubleshooting Guide* for more information. For more information, contact HP customer support.

1

## 错误追踪与调试

日志提供了程序错误和异常的详细记录，能够帮助开发者快速定位问题，加速调试过程。

2

## 性能监控

通过日志监控关键功能的执行时间，可以对系统的性能进行分析，及时优化瓶颈。

3

## 运行状态记录

系统运行的每次请求和响应都能在日志中找到记录，为系统的稳定性和运维提供了重要数据支撑。

# 日志示例

Wed Feb 14 08:34:28 2024

[INFO] Registration failed for user: juanbing

Wed Feb 14 08:34:28 2024

[INFO] Response sent to client

Wed Feb 14 08:34:44 2024

[INFO] Handling connection for fd: 6

Wed Feb 14 08:34:44 2024

[INFO] Found SSL object for fd: 6 in map

Wed Feb 14 08:34:44 2024

[ERROR] SSL\_read failed for fd: 6 with SSL error: 1

# 预期效果

## 初始化日志

在服务端应用启动时，应该开始记录日志，例如使用LOG\_INFO宏记录服务器的启动状态。

## 记录连接

当新的客户端连接时，在日志中记录，帮助追踪活跃的连接和潜在的问题。

## 结束日志

在连接关闭时记录日志，这是确保每次连接都有完整日志记录的重要步骤。

# 预期使用

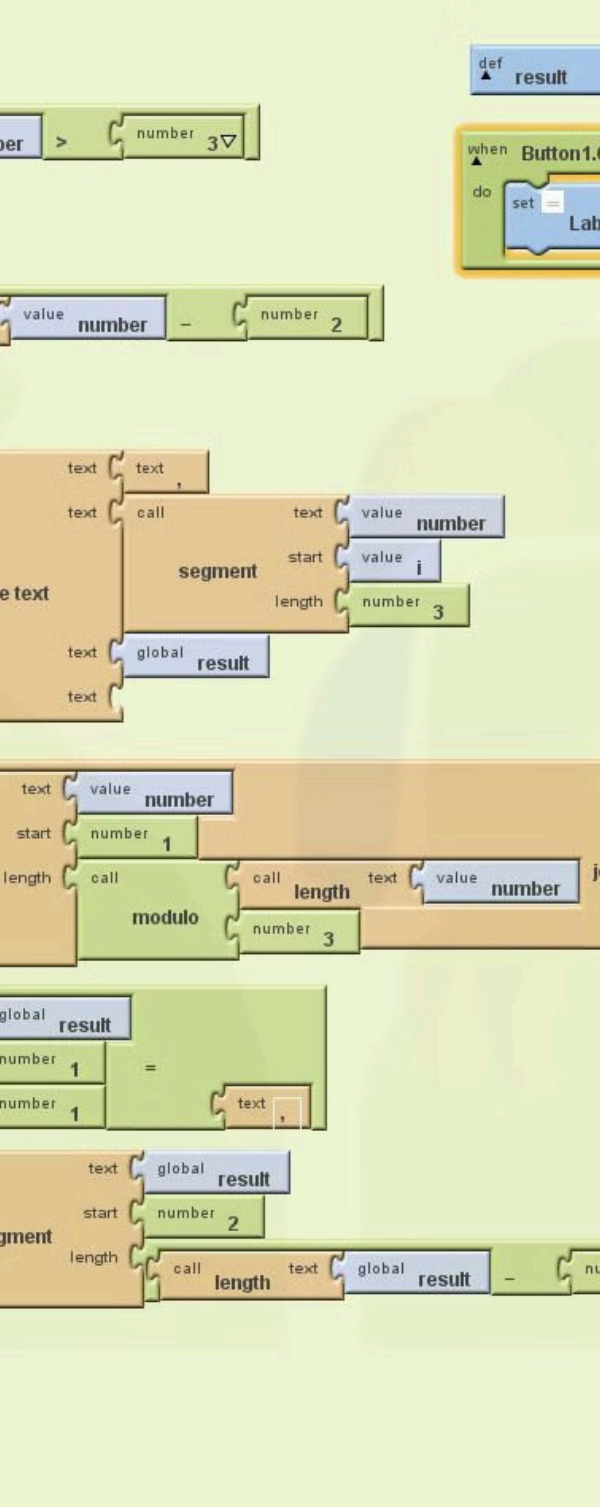
```
#include "Logger.h"

int main() {
    // 初始化和绑定socket的代码
    LOG_INFO("Server starting");

    while (true) {
        new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen);
        LOG_INFO("New connection accepted");

        // 处理请求和发送响应的代码
        LOG_INFO("Connection closed");
    }

    return 0;
}
```



# 创建Logger

1

## 定义日志级别

在Logger.h文件中定义日志级别，以确定消息记录的重要性和分类。

2

## 日志输出函数

根据不同的日志级别，日志内容可以被输出到不同的媒介，如控制台或文件。

3

## 时间戳与上下文

每条日志都应该包含时间戳、文件名和行号，以提供足够的信息以便回溯问题。



A stylized graphic of a hand with fingers spread, rendered in dark brown outlines against a light pink background, positioned at the top of the slide.

## 本节课涉及到的C++知识

M学长的考研Top帮

# 宏 (Macro):

## 预处理指令

宏是预处理器指令，主要用于代码替换，提高代码重用性和简化复杂性。

## 宏函数

宏函数扩展了宏的功能，允许用在需要参数传递的场景，提高代码的动态性和灵活性。

## 使用注意

宏进行简单的文本替换，不进行类型检查和作用域限制，需谨慎使用以避免副作用。

# 宏定义:

1

## 详细解释

宏定义是一种简洁的命名方式，通过预处理阶段的文本替换来使用。

2

## 编译时替换

编译时，PI 将在代码中替换成 3.14159265 的值，令数学计算更为直观。

```
#define PI 3.14159265
```

上述宏定义将在编译时将所有出现的PI替换为3.14159265。

# 宏的调用：

1

## 简化表达

在代码中直接使用宏，可以简化表达式，避免重复编写复杂的数学计算。在代码中使用宏的方式是直接使用宏标识符，例如：

```
double radius = 5.0;  
double area = PI * radius * radius;
```

# 宏的展开：

1

## 直接替换

宏的展开将在编译时完成，将表达式直接替换为宏定义的值。

```
double radius = 5.0;  
double area = PI * radius * radius;  
//上述代码在编译时等价于：  
double radius = 5.0;  
double area = 3.14159265 * radius * radius;
```

# 宏函数（Macro Function）

1

## 宏函数定义

宏函数通过参数扩展了宏的能力，使得可以在代码中传递变量并计算。

2

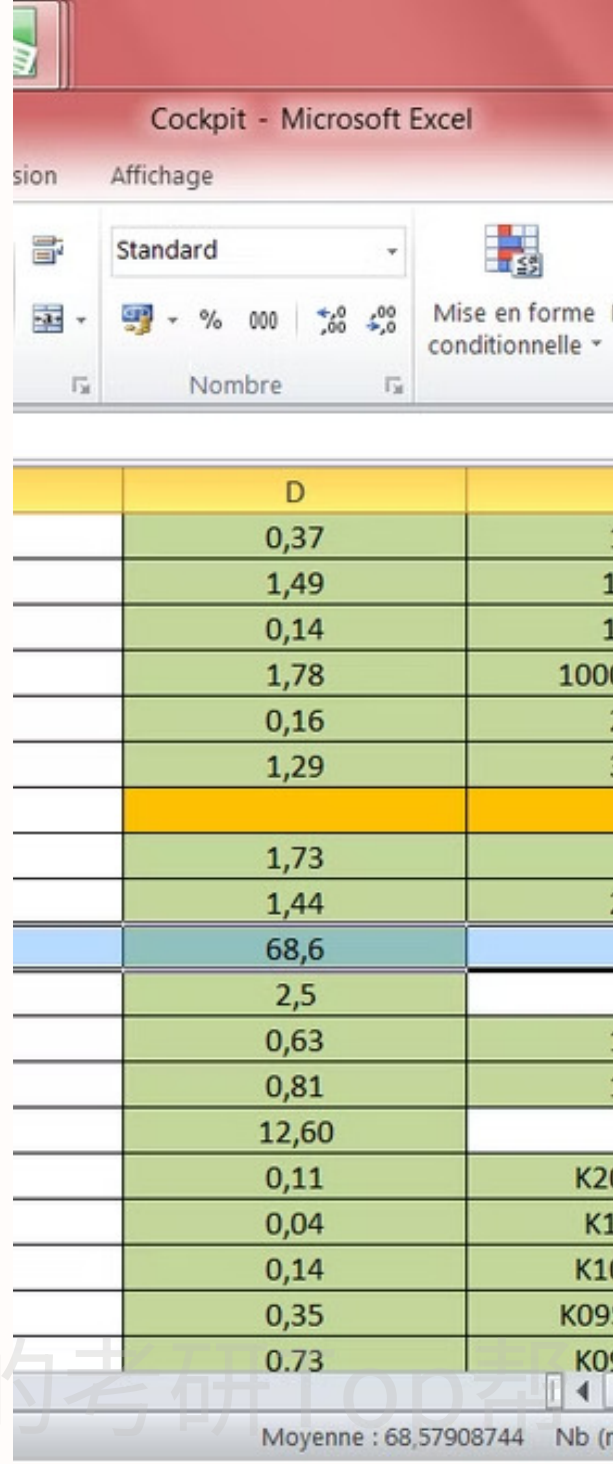
## 宏函数调用

宏函数的调用方法是将参数直接代入宏内完成运算，这样做可以避免函数调用带来的开销。

3

## 编译时展开

宏函数在编译时展开，将参数替换，执行宏定义中规定的运算。



The screenshot shows the Microsoft Excel interface with a table of data. The table has columns labeled 'D' and 'Nb (r)'. The data is as follows:

D	Nb (r)
0,37	
1,49	1
0,14	1
1,78	100
0,16	
1,29	
1,73	
1,44	
68,6	
2,5	
0,63	
0,81	
12,60	
0,11	K2
0,04	K1
0,14	K1
0,35	K09
0.73	K0

The Excel ribbon shows the 'Affichage' (Display) tab with the 'Standard' style selected. The 'Mise en forme conditionnelle' (Conditional Formatting) button is visible. The status bar at the bottom shows 'Moyenne : 68,57908744' and 'Nb (r) : 1'.

## 宏函数定义：

使用`#define`关键字定义一个带参数的宏函数，例如：

```
#define SQUARE(x) ((x) * (x))
```

上述宏函数定义表示可以将一个参数传递给宏函数，并计算其平方。

## 宏函数的调用：

在代码中使用宏函数的方式是将参数传递给宏函数，并使用宏函数进行计算，例如：

```
int result = SQUARE(5);
```



## 宏函数的展开：

在编译时，宏函数会将参数替换到宏定义的位置，并进行计算。上述代码在编译时等价于：

```
int result = ((5) * (5));
```

需要注意的是，宏和宏函数在编译时进行简单的文本替换，没有类型检查和错误检测，因此在使用时需要小心确保参数的正确性和安全性。此外，宏函数的展开可能会导致代码的可读性下降，因此在适当的情况下，应该优先考虑使用函数来代替宏函数。

# 宏函数的弊端

1

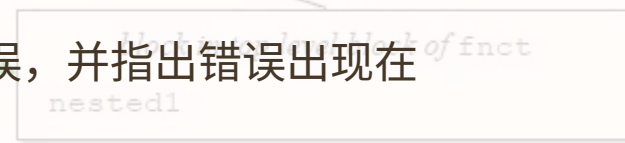
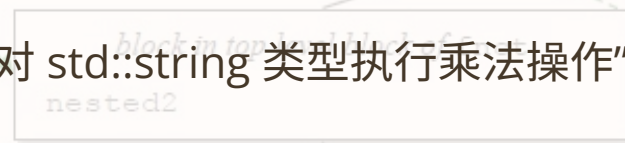
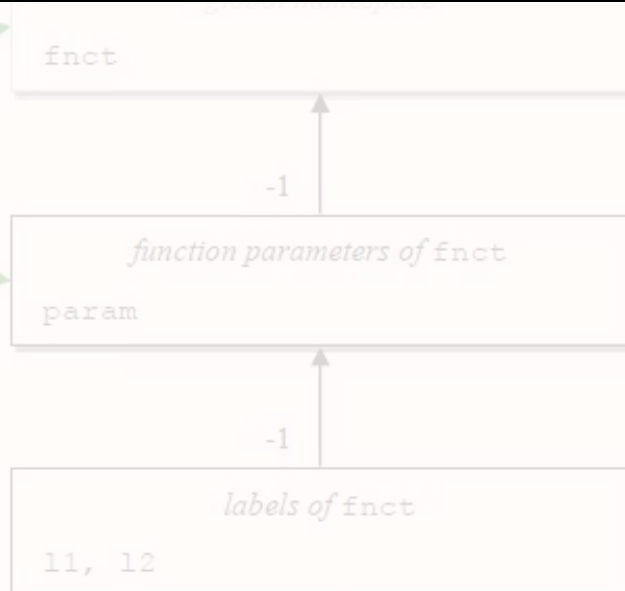
```
#define SQUARE(x) ((x) * (x))
```

//然后一个程序员在一个不适用的地方误用了它：

```
std::string str = "Hello";
```

```
SQUARE(str); // 错误：字符串类型不能进行乘法运算
```

编译器可能报告类似于“无法对 `std::string` 类型执行乘法操作”的错误，并指出错误出现在 `SQUARE(str)` 这一行



# 宏与内联函数与Lambda表达式

## 宏(Macro)

宏通过预处理文本替换增加了编译速度，但可能导致代码副作用。

## Lambda表达式

Lambda表达式支持运行时匿名函数创建，增强了代码的表达力和灵活性。

## 内联函数

内联函数通过预编译时展开优化了性能，消除了函数调用开销。



# Lambda 表达式 (Lambda Expression)

1

## 运行时功能

Lambda表达式允许在运行时创建匿名函数对象，并具备捕获外部状态的能力。

2

## 代码简洁性

Lambda表达式特别适合在需要精简代码书写及强调灵活性的场合使用。

M学长的考研Top帮

# 内联函数 (Inline Function)

1

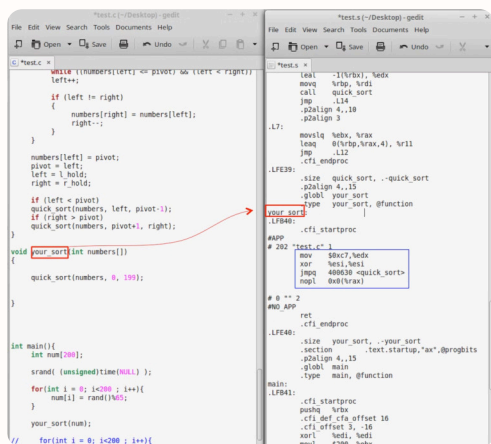
## 优化机制

内联函数作为编译时优化机制，通过减少函数调用的开销来提升程序效率。

2

## 潜在问题

虽然内联函数可以增加性能，但它们可能导致可执行文件增大，以及代码的可读性降低。



# 内联函数实例

C++中通过 `inline` 关键字或隐式由编译器决定是否将函数内联化。

```
inline int square(int x) {  
    return x * x;  
}  
  
int main() {  
    int num = 5;  
    int result = square(num); // 编译器可能将其内联展开，消除函数调用成本  
    return 0;  
}
```

内联函数的主要优点在于减少了函数调用的开销，提高了性能。但缺点是可能会导致可执行文件大小增大，而且过多的内联也可能降低代码的可读性和维护性，同时编译器并不一定会接受程序员的内联建议。



# 总结

1

## 宏的特性

宏通过预处理期间的文本替换实现代码简化，但它不具备类型安全和作用域限制。

2

## 内联函数的特性

内联函数可能在编译时被展开，提供了类型安全性和作用域规则的遵守，但不保证一定内联。

3

## Lambda表达式的特性

Lambda表达式在运行时创造了函数对象，提供了对外部状态的捕获功能，支持更强的编程抽象。

# 日志中宏定义的使用

## 简化操作

宏定义在日志系统中用于简化记录操作，  
提高编码效率。

1

## 处理可变参数

可变参数宏允许编写更复杂的日志信息，  
通过...和va\_list处理不同情况下的日志记  
录。

3

## 捕获上下文

示例宏中捕获信息的文件和行号，增加了  
日志的详细度和追踪方便性。

2



# 可变参数函数

1

## 定义和功能

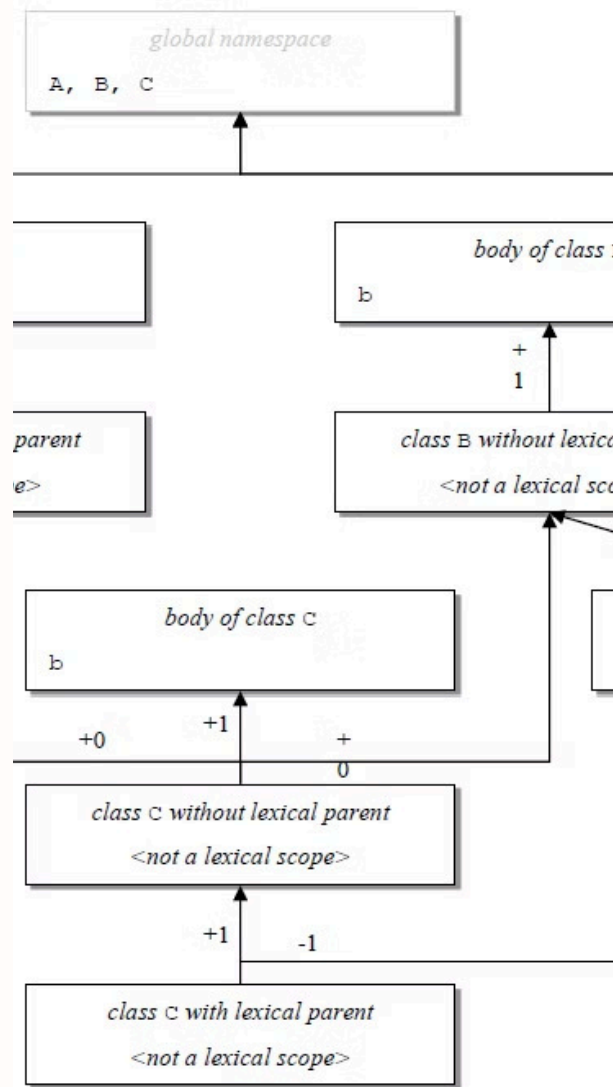
可变参数函数在C++中允许接受不确定数量的参数，为编程带来更大的灵活性。

2

## 可变参数处理

借助于标准库提供的宏，可变参数函数可处理未知数量和类型的参数。

*mbol Table ... Class Per*



# 原理

## 参数传递

函数调用时，不同数量的参数被压入堆栈中，等待被提取。

## 宏助手

`va_list`、`va_start`、`va_arg` 和 `va_end` 这些宏帮助我们管理和解析变量参数列表。

## 安全访问

这些宏确保了我们能够安全地访问可变参数列表中的每一个参数，并适当地解析它们。

# 示例

1

## 可变参数函数

printVarArgs 函数展示了如何使用 vprintf 函数来格式化输出可变参数列表中的参数。

```
#include <cstdarg>
#include <iostream>

void printVarArgs(const char* format, ...) {
    va_list args;
    va_start(args, format);
    vprintf(format, args); // 使用vprintf函数处理可变参数
    va_end(args);
}

int main() {
    printVarArgs("%d %f %s", 10, 3.14, "Hello, World!"); // 调用可变参数函数
    return 0;
}
```

# 可变参数宏详解

## **va\_list**

va\_list 用于存储指向可变参数列表的指针，作为管理变量参数的起点。

## **va\_start**

va\_start 用于初始化 va\_list 变量，使其指向函数行参列表中的第一个可变参数。

## **va\_arg**

va\_arg 用于依次提取可变参数列表中的参数，并将它们转换为适当的类型。

## **va\_end**

va\_end 清理之前由 va\_start 初始化的 va\_list 变量，此步骤在处理完所有参数后必须执行。

# Logger实现

## Logger.h文件

Logger类包含记录日志消息的静态方法，并提供了不同日志级别的枚举定义。

## 日志记录

日志记录器方法负责将时间、日志级别和实际的消息记录到日志文件中。

## 宏定义

Logger.h文件也定义了宏，来简化日志记录过程，方便开发者使用。



# 代码实操

M学长的考研Top帮



**谢谢大家**

M学长的考研Top帮