CPlusPlus编程语言基础

[1. 变量与常量 1](#_128)

[1.1. 变量 1](#_129)

[1.1.1. 变量命名规范 1](#_130)

[1.1.2. 变量声明 1](#_131)

[1.1.3. 初始化 1](#_132)

[1.1.4. 存储信息的基本属性 1](#_133)

[1.2. 常量 1](#_126)

[1.2.1. 常量定义并初始化 1](#_157)

[1.2.2. 算数类型 2](#_156)

[1.2.3. 枚举 3](#_211)

[1.2.4. 符号常量 3](#_970)

[2. 数据类型 3](#_101)

[2.1. 类型 3](#_685)

[2.1.1. 类型 （程序开发语言） 3](#_686)

[2.2. 分类 3](#_1670)

[2.2.1. 内置类型 3](#_103)

[2.2.2. 自定义类型 9](#_102)

[2.2.3. void类型 9](#_656)

[2.3. 类型转换 9](#_730)

[2.3.1. 按数据类型分类 9](#_724)

[2.3.2. 按是否强制分类 10](#_172)

[2.3.3. 按是否隐式分类 10](#_861)

[2.3.4. 按语言分类 10](#_850)

[3. 表达式与语句 11](#_112)

[3.1. 表达式 11](#_237)

[3.1.1. 值 11](#_405)

[3.1.2. 运算符 12](#_701)

[3.1.3. 表达式分类 13](#_413)

[3.2. 语句 14](#_242)

[3.2.1. 复合语句{ } 14](#_260)

[4. 流程控制 14](#_327)

[4.1. 循环 14](#_113)

[4.1.1. 关键字for、while、do while 14](#_238)

[4.1.2. 递增运算符++ 14](#_247)

[4.1.3. 递减运算符-- 14](#_248)

[4.1.4. 注意 14](#_249)

[4.1.5. 基于范围for循环 14](#_267)

[4.2. 条件 14](#_322)

[4.2.1. 关键字if、if else、if else if else 14](#_323)

[4.2.2. 关键字switch 14](#_352)

[4.3. break和continue语句 14](#_358)

[4.3.1. 关键字continue与break 15](#_356)

[5. 函数 15](#_116)

[5.1. 基础概念 15](#_282)

[5.1.1. 函数定义 15](#_283)

[5.1.2. 函数参数 15](#_284)

[5.1.3. 函数分类 16](#_715)

[5.2. 深入 17](#_287)

[5.2.1. 函数重载（函数多态） 17](#_430)

[5.2.2. 函数模板 17](#_435)

[5.2.3. 可变参数模板 18](#_1655)

[5.2.4. 重载解析 18](#_448)

[5.2.5. 函数调用的实现 18](#_1035)

[6. 输入输出和文件 18](#_714)

[6.1. 基础 18](#_1511)

[6.1.1. 控制台 18](#_368)

[6.1.2. 文件 20](#_369)

[6.2. 深入 21](#_1510)

[6.2.1. 输入和输出概述 21](#_1517)

[6.2.2. 使用cout进行输出 22](#_1518)

[6.2.3. 使用cin进行输入 22](#_1519)

[6.2.4. 文件输入和输出 24](#_1520)

[6.2.5. 文件类型 26](#_1601)

[6.2.6. 内核格式化 26](#_1521)

[7. 内存模型 26](#_454)

[7.1. 单独编译 26](#_464)

[7.1.1. 文件与翻译单元 26](#_459)

[7.1.2. 头文件管理 26](#_455)

[7.1.3. 多个库的连接 27](#_460)

[7.2. 介绍 27](#_466)

[7.2.1. 分类：自动变量、寄存器变量（摒弃）、静态变量（包含3种）、动态变量 27](#_467)

[7.2.2. 实质：自动变量、静态变量、动态变量（动态存储） 27](#_587)

[7.3. 变量存储方式 27](#_465)

[7.3.1. 自动变量 27](#_531)

[7.3.2. 静态变量 27](#_484)

[7.3.3. 存储三特性 28](#_527)

[7.3.4. 变量5种存储方式（引入命名空间前）特性总结 29](#_534)

[7.4. 存储说明符 29](#_489)

[7.4.1. auto（C++11以后不再是说明符） 29](#_491)

[7.5. cv-限定符 29](#_490)

[7.5.1. 关键字const 29](#_497)

[7.5.2. 关键字volatile 29](#_498)

[7.5.3. 转换 29](#_1088)

[7.6. 链接性拓展 29](#_546)

[7.6.1. 变量和链接性 29](#_547)

[7.6.2. 函数和链接性 29](#_543)

[7.6.3. 语言和链接性 30](#_548)

[7.7. 存储方案和动态分配 30](#_551)

[7.7.1. 动态分配内存方式 30](#_555)

[7.7.2. 变量5种存储方式不适用于动态分配的内存（动态存储） 30](#_552)

[7.7.3. 存储方式仍然适用于用来跟踪动态内存的指针变量（自动或静态静态指针变量） 30](#_554)

[7.7.4. 编译器使用三块独立的内存 31](#_553)

[7.8. 类和动态内存分配 31](#_1672)

[8. 命名空间 31](#_244)

[8.1. 传统C++命名空间 31](#_566)

[8.1.1. 声明区域 31](#_567)

[8.1.2. 潜在作用域 31](#_568)

[8.1.3. 作用域 31](#_569)

[8.2. 新的命名空间特性 31](#_570)

[8.2.1. 分类 31](#_574)

[8.2.2. 特性 32](#_577)

[8.2.3. 访问命名空间中名称的方法 32](#_580)

[8.3. 使用命名空间的指导原则 32](#_586)

[9. 面相对象 33](#_363)

[9.1. 对象和类 33](#_588)

[9.1.1. 过程性编程和面向对象编程 33](#_590)

[9.1.2. 抽象和类 33](#_592)

[9.1.3. 类的构造函数和析构函数 35](#_624)

[9.1.4. this指针 36](#_659)

[9.1.5. 类作用域 36](#_662)

[9.1.6. 对象数组 37](#_660)

[9.1.7. 抽象数据类型（ADT） 37](#_589)

[9.2. 使用类 37](#_676)

[9.2.1. 运算符重载 37](#_692)

[9.2.2. 简单友元 37](#_693)

[9.2.3. 类的自动转换和强制类型转换 37](#_694)

[9.3. 类和动态内存分配 37](#_677)

[9.3.1. 动态内存和类 37](#_735)

[9.3.2. 改进后的新String类 39](#_736)

[9.3.3. 在构造函数中使用new时的注意事项 39](#_738)

[9.3.4. 有关返回对象的说明 40](#_737)

[9.3.5. 使用指向对象的指针 40](#_739)

[9.3.6. 模拟队列 40](#_784)

[9.4. 类继承 40](#_678)

[9.4.1. 一个简单的基类 40](#_792)

[9.4.2. 继承：is-a关系 41](#_793)

[9.4.3. 多态公有继承 42](#_794)

[9.4.4. 静态联编和动态联编 43](#_795)

[9.4.5. 访问控制：protected 44](#_796)

[9.4.6. 抽象基类 44](#_797)

[9.4.7. 继承和动态内存分配 45](#_798)

[9.4.8. 类设计回顾 45](#_799)

[10. C++中的代码重用 47](#_478)

[10.1. 包含对象成员的类 47](#_938)

[10.1.1. 包含（也称为：组合、层次化） 47](#_942)

[10.1.2. 对比 47](#_960)

[10.1.3. 接口：这里的接口是狭义上的接口，特指被public访问控制符包含的类成员，包括公有的数据成员和公有的函数成员。 47](#_981)

[10.1.4. 初始化被包含的对象（成员对象） 47](#_947)

[10.1.5. 使用被包含对象的接口 47](#_950)

[10.2. 私有继承 47](#_939)

[10.2.1. 建立has-a关系 47](#_953)

[10.2.2. 初始化基类组件 48](#_961)

[10.2.3. 访问基类的方法 48](#_963)

[10.2.4. 访问基类的对象 48](#_965)

[10.2.5. 访问基类的友元函数 48](#_968)

[10.2.6. 保护继承 48](#_956)

[10.2.7. 使用using重新定义访问权限 48](#_958)

[10.3. 多重继承 48](#_940)

[10.4. 类模板 48](#_941)

[10.4.1. 定义模板类 48](#_972)

[10.4.2. 使用模板类 48](#_973)

[10.4.3. 数组模板示例和非类型参数 49](#_974)

[10.4.4. 模板多功能性 49](#_975)

[10.4.5. 模板具体化 49](#_993)

[10.4.6. 成员模板 49](#_998)

[10.4.7. 将模板用作参数 50](#_999)

[10.4.8. 模板和友元 50](#_1000)

[10.4.9. 模板别名 50](#_1001)

[11. 友元和异常 50](#_526)

[11.1. 友元 50](#_1002)

[11.1.1. 分类 50](#_1003)

[11.1.2. 其它友元关系 50](#_1006)

[11.1.3. 共同的友元 50](#_1007)

[11.2. 嵌套类 50](#_1014)

[11.2.1. 嵌套类和访问权限 50](#_1015)

[11.2.2. 模板中的嵌套 50](#_1016)

[11.3. 异常 51](#_1019)

[11.3.1. 调用abort( ) 51](#_1022)

[11.3.2. 返回错误码 51](#_1023)

[11.3.3. 异常机制 51](#_1024)

[11.3.4. 将对象用作异常类型 51](#_1026)

[11.3.5. 异常规范 51](#_1027)

[11.3.6. 栈解退 51](#_1029)

[11.3.7. 其它异常特性 51](#_1030)

[11.3.8. exception类 51](#_1031)

[11.3.9. 异常、类和继承 52](#_1057)

[11.3.10. 未捕获异常和意外异常 52](#_1058)

[11.3.11. 注意事项 53](#_1066)

[11.4. 运行阶段类型识别（RTTI） 53](#_1020)

[11.4.1. RTTI的用途 53](#_1070)

[11.4.2. RTTI的工作原理 53](#_1071)

[11.5. 类型转换运算符 53](#_1021)

[12. string类和标准模板 53](#_657)

[12.1. string类和STL库全面总结 53](#_1314)

[12.1.1. 附录F 模板类string 53](#_1133)

[12.1.2. 附录G 标准模板库方法和函数STL 53](#_1312)

[12.2. string类 53](#_1095)

[12.2.1. 构造字符串 54](#_1102)

[12.2.2. string类输入 54](#_1103)

[12.2.3. 使用字符串 54](#_1104)

[12.2.4. 其他功能 54](#_1105)

[12.2.5. 字符串种类 54](#_1106)

[12.2.6. 解读 54](#_1107)

[12.3. 智能指针模板类 54](#_1096)

[12.3.1. 使用智能指针 54](#_1117)

[12.3.2. 注意事项 54](#_1118)

[12.3.3. 选择智能指针 55](#_1119)

[12.4. 标准模板库 55](#_1097)

[12.4.1. 模板类vector 55](#_1130)

[12.4.2. 可对vector执行的操作 55](#_1131)

[12.4.3. 对vector可执行的其它操作 55](#_1139)

[12.4.4. 基于范围的for循环 55](#_1132)

[12.5. 泛型编程 55](#_1098)

[12.5.1. 标准模板库STL 55](#_1238)

[12.5.2. 迭代器 56](#_1135)

[12.5.3. 容器种类 57](#_1147)

[12.6. 函数对象（函数符） 59](#_1099)

[12.6.1. 函数符概念 59](#_1196)

[12.6.2. 预定义的函数符 59](#_1197)

[12.6.3. 包装器（适配器） 59](#_1653)

[12.6.4. 函数符替代品：Lambda表达式 60](#_1210)

[12.7. 算法 60](#_1100)

[12.7.1. 算法组 60](#_1211)

[12.7.2. 算法的通用特征 60](#_1212)

[12.7.3. STL和string类 61](#_1213)

[12.7.4. STL函数和容器方法 61](#_1214)

[12.7.5. 使用STL 61](#_1215)

[12.8. 容器的列表初始化 61](#_1101)

[12.8.1. 模板initializer\_list 61](#_1236)

[13. 补充专题 61](#_721)

[13.1. 关键字专题 61](#_745)

[13.1.1. 关键字const 61](#_1512)

[13.2. 统一建模语言（UML） 61](#_841)

[13.2.1. UML 教程 61](#_842)

[13.2.2. UML各种图总结-精华 61](#_843)

[13.3. 接口 61](#_599)

[13.3.1. 广义接口 62](#_598)

[13.3.2. 狭义接口 62](#_600)

[13.4. 操作系统 62](#_188)

[13.4.1. 内存管理 62](#_189)

[13.5. 计算机组成原理 62](#_134)

[13.5.1. 存储系统 62](#_135)

[13.5.2. 运算方法与运算器 62](#_137)

[13.6. 编译原理 63](#_502)

[13.6.1. 编译原理知识汇总 63](#_514)

[13.6.2. 程序编译过程 63](#_503)

[13.6.3. 例子：gcc编译C语言程序 63](#_513)

[14. 字符函数库 63](#_328)

[14.1. ctype库 63](#_329)

[14.1.1. 字母或数字？ 63](#_349)

[14.1.2. 用于字母 63](#_343)

[14.1.3. 用于数字 64](#_344)

[14.1.4. 用于空白 64](#_345)

[14.1.5. 用于标点 64](#_346)

[14.1.6. 用于控制符 64](#_347)

[14.1.7. 用于打印符 64](#_348)

[15. 源代码 64](#_504)

[15.1. 预编译 64](#_505)

[15.1.1. 预处理过程的代码 64](#_506)

[15.2. 编译 64](#_507)

[15.2.1. 汇编代码 64](#_509)

[15.3. 汇编 64](#_508)

[15.3.1. 目标代码 64](#_510)

[15.4. 连接 64](#_511)

[15.4.1. 可执行代码 65](#_512)

[16. C++开始的地方 65](#_562)

[16.1. C++百度百科 65](#_1678)

[16.1.1. 官网 65](#_1513)

[16.2. 本资源作者 袁宵 65](#_561)

[16.2.1. https://yuanxiaosc.github.io/ 65](#_1509)

[17. 附录F 模板类string 65](#_1242)

[17.1. 模板basic\_string定义了13种类型,供以后定义方法时使用 65](#_1244)

[17.1.1. 常量npos 65](#_1292)

[17.2. string的数据方法 65](#_1293)

[17.2.1. 迭代器 65](#_1294)

[17.2.2. 元数个数size( )、length( ) 65](#_1298)

[17.2.3. 容量capacity( ) 65](#_1299)

[17.2.4. 最大长度max\_size( ) 65](#_1300)

[17.2.5. 返回const charT\*指针data( )、c\_str( ) 65](#_1301)

[17.2.6. get\_alloctor( ) 66](#_1302)

[17.3. 11种字符串构造函数 66](#_1245)

[17.3.1. 默认构造函数 66](#_1303)

[17.3.2. C-风格字符串的构造函数、部分C-风格字符串的构造函数 66](#_1304)

[17.3.3. 左值引用的构造函数、右值引用的构造函数 66](#_1305)

[17.3.4. 一个字符的n个副本的构造函数 66](#_1306)

[17.3.5. 区间构造函数 66](#_1307)

[17.3.6. 初始化列表构造函数 66](#_1308)

[17.4. 内存操作 66](#_1246)

[17.4.1. risize( )、shrink\_to\_fit( )、clear( )、empty( ) 66](#_1309)

[17.5. 字符串存取 66](#_1247)

[17.5.1. [ ]、at( ) 66](#_1310)

[17.6. 字符串搜索 66](#_1248)

[17.6.1. find( ) 66](#_1252)

[17.6.2. find\_first\_of( ) 66](#_1254)

[17.6.3. find\_first\_not\_of( ) 66](#_1256)

[17.7. 字符串比较 66](#_1249)

[17.7.1. compare( ) 67](#_1258)

[17.7.2. 重载的关系运算符 67](#_1261)

[17.8. 字符串修改 67](#_1250)

[17.8.1. 追加和相加 67](#_1262)

[17.8.2. 赋值 67](#_1263)

[17.8.3. 插入 67](#_1266)

[17.8.4. 清除 67](#_1268)

[17.8.5. 替换 67](#_1270)

[17.8.6. 复制 67](#_1282)

[17.8.7. 交换 67](#_1284)

[17.9. 字符串输入和输出 67](#_1251)

[17.9.1. 输入 67](#_1286)

[17.9.2. 输出 68](#_1287)

[17.10. 参数规律 68](#_1272)

[17.10.1. 操作对象：basic\_string&、const charT\*、charT 68](#_1273)

[17.10.2. 操作区间 68](#_1274)

[18. 附录G 标准模板库方法和函数 68](#_1313)

[18.1. STL容器 68](#_1320)

[18.1.1. 大部分容器都有的成员 68](#_1330)

[18.1.2. 序列容器的其它成员 69](#_1322)

[18.1.3. 有序关联容器 72](#_1323)

[18.1.4. 无序关联容器 73](#_1324)

[18.2. STL函数 74](#_1315)

[18.2.1. 非修改式序列操作 74](#_1316)

[18.2.2. 修改式序列操作 75](#_1317)

[18.2.3. 排序和相关操作 76](#_1318)

[18.2.4. 通用数字运算 77](#_1319)

# 变量与常量

## 变量

### 变量命名规范

#### 标识符

##### 关键字typedef

### 变量声明

#### 定义声明（定义）

##### 单定义规则

#### 引用声明（引用）

##### 关键字extern

### 初始化

#### 列表初始化

##### 大括号初始化器：{ }

#### 关键字auto

### 存储信息的基本属性

#### 运算符sizeof

#### 取地址运算符&

## 常量

### 常量定义并初始化

#### 关键字const

### 算数类型

#### 整型字面值

##### 第一位是1~9的整数是十进制

##### 第一位是0第二位是1~7的整数是八进制

##### 前两位是0x或0X的整数是十六进制

##### 进制

##### 以l或L结尾的整数是long型

##### 以ll或LL结尾的整数是long long型

##### 以u或U结尾的整数是unsigned型

##### 类型

#### 字符型字面值

##### char字面值

##### 字符编码

##### 转义字符

#### bool型字面值：true、false

#### 浮点型字面值

##### 小数点表示法

##### 科学计数法

##### 表示方法

##### 以f或F结尾的浮点数是float型

##### 以L结尾的浮点数是long double型

##### 类型

### 枚举

#### 关键字enum

#### 作用域内枚举

#### 枚举量

##### 默认值

##### 显式赋值

### 符号常量

# 数据类型

## 类型

### 类型 （程序开发语言）

#### 指定基本类型完成了三项工作

#### 不同数据类型占用的字节数根系统有关

## 分类

### 内置类型

#### 基本类型

##### 整型

###### 5种整型关键字：char、int、long、long long

函数库

###### 无符号类型关键字：unsigned

###### 字符类型扩展：char -> wchar\_t、char16\_t、char32\_t

##### 浮点型

###### float、double、long double

##### 单位类型

###### size\_t

sizeof( )

###### size\_type

#### 复合类型

##### 数组

###### 数组声明

数组名与数组的地址

###### 运算符[ ]

###### 数组初始化规则

###### 二维数组

理解 arr[M][N]

char二维数组、字符串指针数组、string对象数组

###### 函数指针数组

###### 数组的替代品

静态数组：模板类array

动态数组：模板类vector

面相数值计算的数组：模板类valarray

对比

##### 字符串

###### 分类

string类

操作

拼接+

复制=

字符数size()、length()

C风格字符串

初始化方法

双引号法

数组法

字符串字面值

空字符：\0

C库函数<string.h>

拼接strcat()

复制strcpy()、strncpy()

大小strlen()

比较strcmp()

原始字符串

R"字符串"

其它形式

wchar\_t、char16\_t、char32\_t

u16string、u32string

##### 结构

###### 关键字struct

###### 成员运算符.

###### 结构数组

###### 结构中的位字段

##### 共用体

###### 关键字union

##### 指针

###### 解除引用\*

###### 分配内存

运算符 new

堆

运算符delete

内存泄漏

数据对象

管理数据内存的方式

动态存储

静态存储

自动存储

根据用于分配内存的方法

局部变量、栈

static、静态区

new、堆

###### 指针运算

递增（减）运算符和指针： \*++pt、++\*pt

###### 指针的应用

指针与数组

创建与删除

使用动态数组

遍历数组等价式

arr[i] == \*(arr + i)

&arr[i] == arr + i

指针与字符串

指针与结构

箭头成员运算符->

指针与类

###### 指针和关键字const

指向const对象的指针

const指针

指向const对象的const指针

###### 分类

函数指针

函数指针声明

函数名不等于函数指针

作用：调用函数和做函数的参数

C++实现多态性的虚函数表是通过函数指针实现

指向函数指针数组的指针

空指针

关键字nullptr

智能指针：帮助管理动态内存分配

悬挂指针（野指针）

广义指针

迭代器

##### 引用

###### 声明引用

###### 引用的特点

###### 主要适应对象

结构和类

###### 主要作用

作为函数参数

尽量使用const

从函数中返回左值

函数返回引用

###### 左值引用与右值引用

分类

左值引用&

拷贝语义

右值引用&&

移动语义和右值引用

强制移动

std::move( )

左值、左值引用、右值、右值引用的比较

### 自定义类型

#### 类型 （技术名词）

#### 类

### void类型

#### 关键字void

## 类型转换

### 按数据类型分类

#### 标准转换

##### 基本数据类型之间的转换

##### 指针、引用、指向成员的指针派生类型的转换

#### 基本类型->自义定类型（类对象）

##### 方法1.构造函数（只能是单参数值）

##### 方法2.重载赋值运算符

##### 对比

#### 自义定类型（类对象）->基本类型

##### 转换函数（特殊的运算符函数）

##### 转换函数、友元函数、重载函数

### 按是否强制分类

#### 自动类型转换

##### 存在问题

#### 强制类型转换

##### 显式强制类型转换

###### 显式转换运算符

关键字explicit

##### 隐式强制类型转换

### 按是否隐式分类

#### 隐式转换

#### 显式转换

### 按语言分类

#### C语言中

#### C++中

##### dynamic\_cast

###### 指向基类的指针向下转型为指向派生类的指针

##### const\_cast

###### 限定符const 和volatile之间转换

##### static\_cast

###### 基类和派生类之间的显式转换、数值类型之间转换

##### reinterpret\_cast

###### 指针、引用、整数三种类型之间的转换

# 表达式与语句

## 表达式

### 值

#### 左值与右值

##### 分类

###### 左值

可修改左值

不可修改左值

特殊类左值：函数、数组

###### 右值

纯右值

将亡值

##### 转换

###### 左值->右值

运算符取地址 &

###### 右值->左值

运算符解引用 \*

#### 表达式的值

### 运算符

#### 运算符分类

#### 运算符性质

##### 运算符的优先级

##### 运算符的结合性：左、右结合性

##### 操作数个数：单、双、三目运算符

##### 可重载性

#### 运算符重载

##### 运算符函数

###### 关键字operator

###### 调用方法

函数表示法

运算符表示法

###### 作为类的成员函数还是非成员函数

##### 重载限制

### 表达式分类

#### 赋值表达式

##### 赋值运算符=

##### 组合赋值运算符

#### 算数表达式

##### 双目加减乘除取余运算符+-\*/%

##### 单目正运算符+、单目负运算符 -

#### 逻辑表达式

##### 逻辑运算符

###### 逻辑与运算符 &&、逻辑或运算符 ||、逻辑非运算符 !

#### 关系表达式

##### 大于运算符>、小于运算符<、大或等于运算符>=、小或等于运算符<=

##### 等于运算符==

###### 不等于运算符!=

#### 条件表达式

##### 条件运算符? :

#### 逗号表达式

##### 逗号运算符

###### 逗号分隔符（区分）

## 语句

### 复合语句{ }

# 流程控制

## 循环

### 关键字for、while、do while

### 递增运算符++

### 递减运算符--

### 注意

#### 副作用

#### 顺序点

#### 执行速度

### 基于范围for循环

## 条件

### 关键字if、if else、if else if else

### 关键字switch

#### 整数表达式

#### 标签

#### 整数、字符、枚举量

#### 关键字default

## break和continue语句

### 关键字continue与break

# 函数

## 基础概念

### 函数定义

#### 函数名

##### 函数地址

#### 函数声明（函数原型）

##### 默认参数

##### 函数声明与定义的关系

### 函数参数

#### 实参argument与形参parameter

#### 局部变量

#### 参数的类型

##### 一维数组名

###### 数组表示与指针表示

###### 显式传递数组大小

##### 二维数组名

##### C风格字符串

###### C风格字符串作为返回值

##### 结构

##### 对象

###### string

###### array

##### 函数

##### 引用

#### 重载引用参数

#### 参数传递方式

##### 分类

###### 按值传递

###### 按指针传递

###### 按引用传递

##### 指导原则

###### 使用传递的值而不做修改的函数

###### 修改调用函数中数据的函数

###### 经验

### 函数分类

#### 普通函数

#### 递归函数

#### 内联函数

##### 关键字：inline

#### 友元函数

##### 关键字friend

## 深入

### 函数重载（函数多态）

#### 方式：使用不同的参数列表完成相同的工作

#### 参数列表（函数特征标）

#### 返回值

### 函数模板

#### 定义

##### template<typename T>

##### 向下兼容

###### template<class T>

#### 函数模板重载

#### 模板的局限性

#### 实例化

##### 隐式实例化

##### 显式实例化

#### 显式具体化

##### template<>

#### 对比

##### 例子

#### 后置返回类型

##### 关键字decltype

### 可变参数模板

#### 参数包

##### 模板参数包和函数参数包

###### 元运算符...

#### 展开参数包

##### 在可变参数模板函数中使用递归

### 重载解析

#### 重载解析过程

#### 类型转换优先级

#### 完全匹配和最佳匹配

### 函数调用的实现

#### 栈

# 输入输出和文件

## 基础

### 控制台

#### 输入

##### 对象cin

###### istream类

###### cin >> 变量

###### 先输入数字再输入字符

###### 输入字符

读取一个字符

cin>>ch

cin.get()

cin.get(char对象名)

读取一行字符串

cin.getline(字符数组名, 长度);

cin.get(字符数组名, 长度)

getline(cin, string对象名)

###### cin状态标志

cin.eof()

cin.fail()

cin.good()

###### 输入错误

分类

类型不匹配

解决方案

条件判断

cin.clear()、cin.sync()和cin.ignor()

#### 输出

##### 对象cout

###### ostream类

###### 输出不同进制的整数

###### 输出一个字符

cout.put(char)

###### 控制输出的格式

cout.setf()

输出布尔值true、false

###### 输出char和wchar\_t类型字符

###### 控制符endl

### 文件

#### 逻辑划分

##### 文本文件

###### 读取

ifstream类

###### 写入

ofstream类

##### 二进制文件

##### 区别

#### 文件结尾EOF

#### 回车与换行

##### \r与\n与\n\r

## 深入

### 输入和输出概述

#### 流和缓冲区

##### streambuf类

#### iostream文件

##### ios\_base类

###### ios类

ostream类

istream类

iostream类

##### 自动创建对象

###### 用于窄字符流

cin、cout、cerr、clog

###### 用于宽字符流

wcin、wcout、wcerr、wclog

#### 重定向

##### 标准输入和输出流通常是指键盘和屏幕

##### 可使用重定向改变流连接对象

###### 输入重定向<

###### 输出重定向>

### 使用cout进行输出

#### 输出基本类型和字符串

##### 插入运算符<<

#### 输出字符put( )

##### 输出字符串write( )

#### 刷新输出缓冲区

##### flush

###### endl

#### 输出格式化

##### 格式常量

###### 标准控制符

##### 浮点数的精度

##### cout成员函数precison( )、width( )、fill( )

##### 头文件iomanip

###### 控制符setprecision( )、setw( )、setfill( )

### 使用cin进行输入

#### 输入基本类型和字符串

##### 格式化抽取方法

###### 抽取运算符>>

#### cin>>检查输入

##### 跳过空白（空格、换行、制表符）

#### 流状态

##### 流状态成员

###### 到达文件尾

eofbit

eof( )

###### 流被破坏

badbit

bad( )

###### 与预期不符

failbit

fail( )

###### 正常状态

goodbit

good( )

###### 操作

返回流状态rdstate( )

设置状态

clear( )

setstate( )

##### 流状态的影响

###### I/O和异常

异常

exceptions( )

exceptions(isostate ex)

##### 主动检测流状态

#### 其它istream类方法

##### 非格式化输入函数

###### 单字符输入

cin.get(char&)

cin.get( )

对比

###### 字符串输入：getline( )、get( )、ignore( )

##### 意外字符输入：文件尾、流被破坏、无输入、输入到达或超过指定最大字符数

##### read( )、peek( )、gcount( )、putback( )

### 文件输入和输出

#### fstream族与iostream族

##### ifstream继承自istream

##### ofstream继承自ostream

##### fstream继承自iostream

#### 简单的文件I/O

#### 流状态检查

##### is\_open( )

#### 打开多个文件

#### 命令行处理技术

##### int main(int argc, char \*argv[ ])

#### 文件模式

##### 文件模式常量

##### C语言模式字符串："r"、"w"、"a"、"r+"、"w+"

##### 对应关系

#### 随机存取

##### 指针移动

###### 输入指针移动

seekg( )

###### 输出指针移动

seekp( )

##### 获取指针当前位置

###### 对于输入流

tellg( )

###### 对于输出流

tellp( )

###### 注意

##### 使用临时文件

###### tmpnam( )

### 文件类型

#### 文本文件

##### 插入运算符<<写入、抽取运算符>>和get( )读取

#### 二进制文件

##### write( )写入、read( )读取

###### 例子

#### 对比

### 内核格式化

#### sstream族

##### ostringstream继承自ostream

###### str( )成员函数

##### istringstream继承自istream

##### stringstream继承自iostream

#### sstream族给格式化的文本提供了缓冲区

# 内存模型

## 单独编译

### 文件与翻译单元

### 头文件管理

#### 包含头文件命令#incude

##### "filename" 与<filename>

#### 头文件内容

#### 预处理器编译命令

##### #ifndef

##### #define、#endif

### 多个库的连接

#### 名称修饰

#### 链接错误

##### 重新编译源代码

## 介绍

### 分类：自动变量、寄存器变量（摒弃）、静态变量（包含3种）、动态变量

### 实质：自动变量、静态变量、动态变量（动态存储）

## 变量存储方式

### 自动变量

#### 自动变量的初始化

#### 自动变量和栈

### 静态变量

#### 初始化

##### 零初始化

##### 常量表达式初始化

##### 动态初始化

#### 特性

##### 静态持续性、外部链接性

###### 全局变量（外部变量）

作用域解析运算符::

##### 静态持续性、内部链接性

###### 局部变量（内部变量）

关键字static

关键字extern

##### 静态存储持续性、无链接性

###### 代码块和函数中

关键字static

### 存储三特性

#### 持续性->变量在内存保留（持续）时间

##### 自动存储持续性

##### 静态存储持续性

##### 线程存储持续性

##### 动态存储持续性

#### 作用域->变量在文件的多大范围内可见（可被程序使用）

##### 代码块

##### 文件

#### 链接性->变量在哪些文件之间共享

##### 无链接性（只能在当前函数或代码块中访问）

##### 内部链接性（只在当前文件中访问）

##### 外部链接性（可在其他文件中访问）

### 变量5种存储方式（引入命名空间前）特性总结

## 存储说明符

### auto（C++11以后不再是说明符）

#### register

##### static

###### extern

thread\_local

mutable

## cv-限定符

### 关键字const

#### const全局变量链接性：内部

### 关键字volatile

### 转换

## 链接性拓展

### 变量和链接性

### 函数和链接性

#### 持续性：静态

#### 链接性：默认外部链接性

##### 内部链接性：关键字static

#### 非内联函数的单定义规则

##### 内联函数特殊性

#### C++在哪里查找函数

### 语言和链接性

#### C语言链接规范和C++语言链接规范

## 存储方案和动态分配

### 动态分配内存方式

#### C++运算符new

##### new：运算符、函数和替换函数

##### new失败时

###### 引发异常std::bad\_alloc

##### 定位new运算符

###### 内置类型与定位new运算符

###### 对象与定位new运算符

显式调用析构函数

#### C函数malloc( )

### 变量5种存储方式不适用于动态分配的内存（动态存储）

### 存储方式仍然适用于用来跟踪动态内存的指针变量（自动或静态静态指针变量）

### 编译器使用三块独立的内存

#### 一块用于静态变量（可再细分）

#### 一块用于自动变量

#### 一块用于动态存储

## 类和动态内存分配

# 命名空间

## 传统C++命名空间

### 声明区域

### 潜在作用域

### 作用域

#### 分类

##### 全局（文件）作用域

##### 局部（代码块）作用域

##### 类作用域

#### 访问作用域内成员方法

##### 作用域解析运算符::

##### 直接成员运算符.

##### 间接成员运算符->

## 新的命名空间特性

### 分类

#### 全局命名空间

##### 全局变量

#### 自定义命名空间

##### 关键字namespace

### 特性

#### 外部链接性（默认）

#### 解决命名冲突

#### 开放性

#### 传递性

#### 可以嵌套

##### 可别名

#### 无名命名空间

### 访问命名空间中名称的方法

#### 直接指定标识符

##### 作用域解析运算符::

#### 关键字using

##### using声明

##### using编译指令

##### 对比

## 使用命名空间的指导原则

# 面相对象

## 对象和类

### 过程性编程和面向对象编程

#### 面相对象编程特性

### 抽象和类

#### 抽象是什么

##### 抽象(Abstraction)是简化复杂的现实问题的途径

##### 包括

###### 过程抽象

###### 数据抽象

##### 抽象和接口关系

#### 类是什么

##### 对象是什么

###### 类和对象的关系

##### 类如何实现抽象、数据隐藏和封装

##### 类与结构的区别、类与模板的区别

#### 定义类

##### 目标：使得使用类与使用基本的内置类型（如int）尽可能相同

##### 关键字class

##### 类规范

###### 类声明（蓝图）

数据成员

成员函数

###### 类方法定义（细节）

实现类成员函数

##### 类设计步骤

###### 1.提供类声明

访问控制

关键字private

默认private

数据成员常放在private部分

数据隐藏

关键字public

public和抽象

公共接口

函数成员常放在public部分

关键字protected

存储控制

关键字static

静态成员变量

静态成员函数

###### 2.实现类成员函数

类成员函数特点

类作用域

作用域解析运算符::

限定名

非限定名

类方法可以访问类的私有成员

内联函数

内部链接性

共享一组成员函数

const成员函数

#### 使用简单的类

##### 创建对象（类的实例）

###### 将类名视为类型名

##### 使用类函数（公有接口）

###### 成员运算符句点.

###### 客户-服务器模型

### 类的构造函数和析构函数

#### 构造函数

##### 作用：初始化对象

###### 初始化与赋值

成员初始化列表

类内初始化

等价

##### 声明和定义构造函数

###### 声明构造函数

默认参数

###### 定义构造函数

##### 默认构造函数

###### 默认构造函数初始化

调用方法：显式调用、隐式调用

#### 析构函数

##### 作用：完成清理工作

##### 通常由编译器决定调用析构函数的时机

##### 默认析构函数

#### 构造函数和析构函数

##### 都没有返回类型（连void都没有）

### this指针

### 类作用域

#### 作用域为类的常量

##### 在类中声明一个枚举（整数类型）

###### 状态成员

##### 使用关键字static

#### 作用域内枚举

### 对象数组

#### 对象数组初始化与默认构造函数

### 抽象数据类型（ADT）

#### 类适合用于描述ADT

## 使用类

### 运算符重载

### 简单友元

#### 友元函数

#### 常用的有元

##### 重载<<运算符

#### 派生类通过强制转换为基类类型来使用基类的友元

### 类的自动转换和强制类型转换

## 类和动态内存分配

### 动态内存和类

#### 类的静态成员

##### 静态成员变量

###### 关键字static

###### 整型const、枚举型const静态成员可以在类声明中初始化

##### 静态类成员函数

###### 关键字static

#### 特殊成员函数

##### 分类

###### 默认构造函数

默认的默认构造函数

###### 复制构造函数

默认的复制构造函数

###### 赋值运算符

默认的赋值运算符

重载赋值运算符

###### 注意

“浅复制”与“深复制”

复制与赋值的异同

连续赋值问题

###### 析构函数

默认的析构函数

###### 移动构造函数

默认的移动构造函数

###### 移动赋值运算符

默认的移动赋值运算符

###### 注意

使用时机

##### 注意事项

###### 编译器自动提供的函数

自动定义

存在隐患

显式自定义

解决隐患

###### 启用和禁用成员函数

启用默认的方法

关键字default

禁用方法

关键字delete

伪私有方法

### 改进后的新String类

### 在构造函数中使用new时的注意事项

#### 使用new时的推荐做法

##### new的三个“统一”

##### 空指针：NULL、0、nullptr

##### 应当定义一个复制构造函数

##### 应当定义一个赋值运算符

#### 包含类成员的类的逐成员复制

### 有关返回对象的说明

#### 返回指向const对象的引用

#### 返回指向非const对象的引用

#### 返回对象

#### 返回const对象

#### 规律

### 使用指向对象的指针

#### 析构函数调用时机

##### 自动变量

##### 静态变量

##### 动态变量

#### 指针和对象小结

##### 指针和对象

##### 使用new创建对象具体步骤

### 模拟队列

#### 嵌套结构和类

## 类继承

### 一个简单的基类

#### 派生一个类

##### 继承的内容

###### 不能继承的内容

##### 需要增添的内容

###### 构造函数

##### calss 派生类名: 访问控制符 基类名

#### 派生类构造函数

##### 访问权限的考虑

###### 派生类不能直接访问基类的私有成员

##### 是否使用初始化列表

###### 使用基类默认构造函数

###### 显式调用基类构造函数

#### 派生类和基类之间的特殊关系

### 继承：is-a关系

#### 继承方式分类

##### 公有继承

###### 能建立

is-a

has-a

is-implement-as-a

uses-a

导致问题

###### 不能建立

is-like-a

##### 保护继承

##### 私有继承

#### 类之间关系分类

##### is-a

###### 使用公有继承来处理

is-a关系的进一步抽象

##### has-a

###### 使用包含、私有继承和保护继承来处理

##### is-like-a

###### 设计共有特征的类来处理

##### is-implement-as-a

###### 使用隐藏数据成员来处理

##### uses-a

###### 使用友元函数或类来处理

### 多态公有继承

#### 实现多态公有继承机制

##### 在派生类中重新定义基类的方法

##### 使用虚方法

###### 关键字virtual

#### 多态中确定调用哪个类的方法

##### 通过限定名

###### 类名::方法名

类名决定

##### 通过引用或指针

###### 不使用关键字virtual

引用类型或指针类型决定

###### 使用关键字virtual

引用或指针指向的对象类型决定

### 静态联编和动态联编

#### 函数名联编

##### 分类

###### 静态联编（早期联编）

###### 动态联编（晚期联编）

#### 指针和引用类型的兼容性

##### 向上强制转换

###### 传递性

###### 允许隐式向上类型转换

###### 本质

is-a关系

##### 向下强制转换

###### 只允许显式向下强制类型转换

运算符static\_cast

#### 虚成员函数和动态联编

##### 为什么有两种类型联编以及默认为静态联编

###### 效率

###### 概念模型

##### 虚函数

###### 虚函数工作原理

虚函数表

###### 虚函数的作用

##### 有关虚函数注意事项

###### 构造函数

###### 析构函数

###### 友元

###### 没有重新定义继承的方法

###### 重新定义继承的方法

注意与函数重载区别

经验规则

### 访问控制：protected

#### 对外部世界：保护成员的行为与私有成员相似

#### 对派生类：保护成员的行为与公有成员相似

### 抽象基类

#### 抽象基类（abstract base class, ABC）

##### 纯虚函数

###### =0

##### 具体类

#### 应用ABC概念

#### ABC理念

### 继承和动态内存分配

#### 分类

##### 派生类不使用new

##### 派生类使用new

###### 必须自定义

复制构造函数

赋值运算符

析构函数

#### 使用动态内存分配和友元的继承示例

### 类设计回顾

#### 编译器生成的公有成员函数

##### 默认构造函数

##### 复制构造函数

##### 赋值运算符

#### 其它类方法

##### 构造函数

##### 析构函数

##### 类型转换

##### 按值传递对象与传递引用

##### 返回对象和返回引用

##### 使用const

#### 公有继承的考虑因素

##### is-a关系

##### 为什么不能被继承

##### 赋值运算符

##### 私有成员与保护成员

##### 虚方法

##### 析构函数

##### 友元函数

##### 有关使用基类方法的说明

#### 类函数小结

##### 成员函数属性

###### 能否继承

###### 成员还是友元

###### 能否默认生成

###### 能否为虚函数

###### 是否可以有返回类型

# C++中的代码重用

## 包含对象成员的类

### 包含（也称为：组合、层次化）

#### 建立has-a关系

##### A对象中的B对象：A包含B

### 对比

#### 包含=A类获得了其成员B对象的实现，但不继承接口

#### 公有继承=获得实现（若有）+继承接口

#### 接口与实现

### 接口：这里的接口是狭义上的接口，特指被public访问控制符包含的类成员，包括公有的数据成员和公有的函数成员。

### 初始化被包含的对象（成员对象）

#### 构造函数使用其成员名

##### 初始化顺序

### 使用被包含对象的接口

#### 被包含对象的接口不是公有的，但可以在类方法中使用它。

#### 对象名.数据成员 / 对象名.函数成员

## 私有继承

### 建立has-a关系

#### 使用包含还是私有继承

### 初始化基类组件

#### 构造函数使用基类的类名

### 访问基类的方法

#### 基类名::方法名

### 访问基类的对象

#### 使用强制类型转换

##### (const 基类名&) \*this

### 访问基类的友元函数

#### 使用强制类型转换

### 保护继承

#### 各种继承方式对比

### 使用using重新定义访问权限

#### 只适用于继承关系

## 多重继承

## 类模板

### 定义模板类

#### template<typename Type>

#### template<class Type>

#### 泛型标识符 Type

##### 类型参数

### 使用模板类

#### 必须显式地指出模板类的具体类型

### 数组模板示例和非类型参数

#### 指定数组大小的数组模板

##### 方案1：在类中使用动态数组和构造函数来提供数目

##### 方案2：使用模板参数来提供常规数据的大小

##### 对比

#### 非类型参数（表达式参数）

##### array模板类

### 模板多功能性

#### 递归使用模板

##### Array< Array<int, 5>, 10> twodee;

#### 使用多个类型参数

##### 预定义模板类pair<class T, class U>

#### 默认类型模板参数

### 模板具体化

#### 隐式实例化

#### 显式实例化

#### 显式具体化

##### 部分具体化

### 成员模板

### 将模板用作参数

### 模板和友元

### 模板别名

# 友元和异常

## 友元

### 分类

#### 友元函数

#### 友元类

##### 例子：电视机类和遥控器类

#### 友元成员函数

##### 向前声明

### 其它友元关系

#### 互为友元类

### 共同的友元

#### 函数需要访问两个类的私有数据

## 嵌套类

### 嵌套类和访问权限

#### 嵌套类作用域

#### 访问控制

### 模板中的嵌套

## 异常

### 调用abort( )

### 返回错误码

### 异常机制

#### 关键字try、关键字catch、关键字throw

### 将对象用作异常类型

### 异常规范

#### 关键字noexcept、关键字throw

### 栈解退

#### 普通函数的调用返回机制

#### 栈解退特性

#### 对比

##### throw和return之间的区别

### 其它异常特性

#### throw-catch机制

#### 临时拷贝机制

### exception类

#### what( )函数

#### 头文件stdexcept

##### logic\_error类

###### domain\_error类

###### invalid\_argument类

###### length\_eroor类

###### out\_of\_bounds类

##### runtime\_error类

###### range\_error类

###### overflow\_error类

###### underflow\_error类

#### 头文件new

##### bad\_new类

###### std::bad\_alloc 异常

#### 空指针和new

##### std::nothrow

### 异常、类和继承

### 未捕获异常和意外异常

#### 未捕获异常

##### terminate( )

###### 默认调用abort( )

###### set\_terminate( )

#### 意外异常

##### unexpected( )

###### set\_unexpected( )

### 注意事项

#### 内存动态分配和异常

##### 内存泄漏问题

###### 智能指针模板

## 运行阶段类型识别（RTTI）

### RTTI的用途

### RTTI的工作原理

#### 运算符dynamic\_cast

##### bad\_cast异常

#### 运算符typeid

##### 重载了==和!=运算符

##### bad\_typeid异常

#### type\_info类

##### name( )成员函数

## 类型转换运算符

# string类和标准模板

## string类和STL库全面总结

### 附录F 模板类string

### 附录G 标准模板库方法和函数STL

## string类

### 构造字符串

#### 9种构造函数

### string类输入

#### getline(cin, str)

#### cin >> str

### 使用字符串

### 其他功能

#### 返回C-风格字符串str.c\_str( )

### 字符串种类

#### 本质：模板类basic\_string的具体化，然后typedef取的别名

#### string、wstring、u16string、u32string

### 解读

#### 成员函数和运算符被多次重载

##### 参数是string对象、C-风格字符串、char值

## 智能指针模板类

### 使用智能指针

#### 智能指针模板类auto\_ptr、unique\_ptr、shared\_ptr

#### 所有的智能指针类都是explicit构造函数

#### 智能指针只能指向动态分配的堆内存

### 注意事项

#### auto\_ptr导致的多次删除同一对象

### 选择智能指针

#### 要使用多个指向同一对象的指针

##### 选择shared\_ptr

#### 不需要使用多个指向同一对象的指针

##### 选择unique\_ptr

## 标准模板库

### 模板类vector

#### 分配器

### 可对vector执行的操作

#### size( )、begin( )、end( )、push\_back( )、erase( )、insert( )

### 对vector可执行的其它操作

#### STL函数：for\_each( )、random\_shuffle( )、sort( )

#### sort( )与全排序、完整弱排序

### 基于范围的for循环

#### for(auto 元素 : 容器)

##### 对比for\_each( )

## 泛型编程

### 标准模板库STL

#### STL术语

##### 概念：描述一系列的要求

###### 模板参数与概念

模板参数与迭代器概念

模板参数与函数符概念

###### 注意编译器不直接检查概念

##### 改进：表示概念上的继承

##### 模型：概念的具体实现

#### STL组成：容器(containers)、迭代器(iterators)、空间配置器(allocator)、适配器(adapters)、算法(algorithms)、函数符(functors)六个部分

### 迭代器

#### 为何使用迭代器

##### 迭代器是STL算法的接口

###### 模板使得算法独立于存储的数据类型

###### 迭代器使得算法独立于使用的容器类型

##### 基于算法的要求设计迭代器特征和容器特征

##### 超尾标记

#### 迭代器类型

##### 分类

###### 输入迭代器、输出迭代器、正向迭代器、双向迭代器、随机访问迭代器

##### 共性：可以执行解除引用操作、可以进行比较、递增、同一个类级typedef名称：iterator

#### 迭代器层次结构

##### 5种迭代器功能具有层次递增的包含关系

#### 将指针用作迭代器

##### 适配器

##### 迭代器适配器

###### ostream\_iterator模板使ostream输出用作迭代器接口

###### istream\_iterator模板使istream输入用作迭代器接口

#### 其他有用的迭代器

##### reverse\_iterator

##### back\_insert\_iterator、front\_insert\_iterator、insert\_iterator

### 容器种类

#### 容器

##### 容器概念

###### 容器的要求

##### 容器类型

###### vector、set、map等15种容器类型

#### 序列容器

##### 序列的要求

###### 序列的可选要求

##### 分类

###### vector

###### deque

###### list

forward\_list

###### 适配器类

queue

priority\_queue

stack

###### 非STL容器

array

#### 关联容器

##### 有序关联容器

###### set

multiset

###### map

multimap

###### 模板类pair<class T, class U>

##### 无序关联容器

###### unordered set、unordered multiset,、unorderedmap、unordered multimap

## 函数对象（函数符）

### 函数符概念

#### 函数对象是重载了( )运算符的类

#### 分类

##### 生成器、一元函数、二元函数

##### 谓词、二元谓词

### 预定义的函数符

#### 头文件function

##### 运算符和对应的函数符

### 包装器（适配器）

#### 自适应函数符和函数适配器

##### 自适应函数符

##### 函数适配器

###### 函数适配器类

binder1st、binder2nd

###### 函数适配器函数

bind1st( )、bind2nd( )

#### function类模板

##### 可调用类型：函数名、函数指针、函数对象、有名称的Lambda表达式

### 函数符替代品：Lambda表达式

#### 比较函数指针、函数符和Lambda函数

#### 有名的Lambda表达式

#### 额外功能：访问作用域内变量

##### 例子：使用Lambda时机

## 算法

### 算法组

#### 非修改式序列操作

#### 修改式序列操作

#### 排序和相关操作

#### 头文件algorithm

#### 通用数字运算

##### 头文件numerica

### 算法的通用特征

#### 模板函数参数标识符的作用

##### 标识符指出算法需要的模型对应的概念

#### 按算法结果放置位置分类

##### 就地算法

##### 复制算法

###### 以\_copy结尾

#### 根据将函数应用于容器元素得到的结果来执行操作的算法

##### 以\_if结尾

### STL和string类

### STL函数和容器方法

#### 优先选择容器方法

#### 对于容器：STL函数更通用，容器方法更合适

### 使用STL

## 容器的列表初始化

### 模板initializer\_list

#### 使用initializer\_list对象

##### 与参数包的区别与联系

# 补充专题

## 关键字专题

### 关键字const

#### const用于函数

## 统一建模语言（UML）

### UML 教程

### UML各种图总结-精华

## 接口

### 广义接口

### 狭义接口

#### 软件接口

#### 硬件接口

## 操作系统

### 内存管理

#### 堆、栈、静态区、常量区、代码区

#### 自动存储、静态存储、动态存储

## 计算机组成原理

### 存储系统

#### 位、字节、字

### 运算方法与运算器

#### 运算方法

##### 数的机器码表示

###### 整数的表示

原码、反码、补码

###### 浮点数的表示

深入浅出浮点数

浮点数的表示和运算

###### 在线演示

##### 非数值数据的表示

###### Unicode

国际化策略：字符编码问题

编码类型：ASCII、UTF-8、UTF-16、UTF-32

###### ASCII编码

#### 运算器

##### 定点数四则运算

## 编译原理

### 编译原理知识汇总

### 程序编译过程

### 例子：gcc编译C语言程序

# 字符函数库

## ctype库

### 字母或数字？

#### isalnum( )

### 用于字母

#### isalpha( )

#### islower( )

#### isupper( )

#### tolower( )

#### toupper( )

### 用于数字

#### isdigit( )

#### isxdigit( )

### 用于空白

#### isspace( )

### 用于标点

#### ispunct( )

### 用于控制符

#### iscntrl( )

### 用于打印符

#### isgraph( )

#### isprint( )

# 源代码

## 预编译

### 预处理过程的代码

## 编译

### 汇编代码

## 汇编

### 目标代码

## 连接

### 可执行代码

# C++开始的地方

## C++百度百科

### 官网

#### 参考手册

##### 教程

## 本资源作者 袁宵

### https://yuanxiaosc.github.io/

# 附录F 模板类string

## 模板basic\_string定义了13种类型,供以后定义方法时使用

### 常量npos

## string的数据方法

### 迭代器

#### begin( )、end( )

#### rbegin( )、rend( )

#### 说明

### 元数个数size( )、length( )

### 容量capacity( )

### 最大长度max\_size( )

### 返回const charT\*指针data( )、c\_str( )

### get\_alloctor( )

## 11种字符串构造函数

### 默认构造函数

### C-风格字符串的构造函数、部分C-风格字符串的构造函数

### 左值引用的构造函数、右值引用的构造函数

### 一个字符的n个副本的构造函数

### 区间构造函数

### 初始化列表构造函数

## 内存操作

### risize( )、shrink\_to\_fit( )、clear( )、empty( )

## 字符串存取

### [ ]、at( )

#### front( )、back( )

## 字符串搜索

### find( )

#### rfind( )

### find\_first\_of( )

#### find\_last\_of( )

### find\_first\_not\_of( )

#### find\_last\_not\_of( )

## 字符串比较

### compare( )

### 重载的关系运算符

## 字符串修改

### 追加和相加

#### append( )、push\_back( )

#### 重载的+、+=运算符

### 赋值

#### assign( )

#### 重载的=运算符

### 插入

#### insert( )

### 清除

#### erase( )、pop\_back( )

### 替换

#### replace( )

### 复制

#### copy( )

### 交换

#### swap( )

## 字符串输入和输出

### 输入

#### getline( )函数

#### 重载的>>运算符

### 输出

#### 重载的<<运算符

## 参数规律

### 操作对象：basic\_string&、const charT\*、charT

### 操作区间

#### 使用计数

#### 使用位置+使用计数

#### 使用迭代器区间

# 附录G 标准模板库方法和函数

## STL容器

### 大部分容器都有的成员

#### 为所有容器定义的类型

##### X::value\_type

##### X::reference、X::const\_reference

##### X::iterator、X::const\_iterator

##### X::different\_type

##### X::size\_type

#### 为所有容器定义的操作

##### X u、X( )、X(a)、X u(a)、X u = a

##### r = a

##### (&a)->X( )

##### begin( )、end( )、cbegin( )、cend( )

##### size( )、maxsize( )

##### empty( )

##### swap( )

##### ==、！=

#### 可反转容器定义的类型和操作

##### 可反转容器：vector、list、deque、array、set、map

##### X::reverse\_iterator( )、X::const\_reverse\_iterator( )

##### a.rbegin( )、a.rend( )、a.crbegin( )、a.crend( )

#### 有序容器操作

##### 除无序set和无序map容器外都需要支持的操作

##### <、>、<=、>=

### 序列容器的其它成员

#### 为序列容器定义的其它操作

##### 序列容器：vector、forward\_list、list、deque、array

##### X(n, t)、X a(n, t)

###### X(i, j)、X a(i, j)

##### X(初始化列表对象)

###### a = 初始化列表对象

##### a.cmplace(p, args)

##### a.insert(p, t)、a.insert(p, n, t)

###### a.insert(p, i, j)

a.insert(p, 初始化列表对象)

##### a.resize(n)、a.resize(n, t)

##### a.assign(i, j)、a.ssign(n, t)、a.ssign(初始化列表)

##### a.erase(q)、a.erase(q1, q2)

##### a.clear( )

##### a.front( )

#### 为某些序列容器定义的操作

##### a.back( )

###### 部分序列容器：vector、list、deque

##### a.push\_back(t)

###### 部分序列容器：vector、list、deque

##### a.pop\_back( )

###### 部分序列容器：vector、list、deque

##### a.emplace\_back(args)

###### 部分序列容器：vector、list、deque

##### a.push\_front(t)

###### 部分序列容器：forward\_list、list、deque

##### a.emplace\_front()

###### 部分序列容器：forward\_list、list、deque

##### a.pop\_front( )

###### 部分序列容器：forward\_list、list

##### a[n]、a.at(n)

###### 部分序列容器：vector、deque、array

#### vector的其它操作

##### a.capacity( )

##### a.reserve(n)

#### list的其它操作

##### a.splice(p, b)、a.splice(p, b, i)、a.splice(p, b, i, j)

##### a.remove(const T& t)

##### a.unique( )、a.unique(BinaryPredicate bin\_pred)

##### a.merge(b)、a.merge(b, Compare comp)

##### a.sort( )、a.sort(Compare comp)

##### a.revese( )

#### forward\_list操作

##### insert\_after( )、erase\_after( )、splice\_after( )

###### 其它操作同list

### 有序关联容器

#### 有序关联容器：set、multiset、map、multimap

#### 为有序关联容器定义的类型

##### X::key\_type

##### X::key\_compare

##### X::value\_compare

##### X::mapped\_type

###### 仅限于容器map、multimap

#### 为有序关联容器定义的操作

##### X(i, j, c)、X a(i, j, c)、X(i, j)、X a(i, j)

##### X(初始化列表对象)

##### a = 初始化列表对象

##### a.key\_comp( )

##### a.value\_comp( )

##### a\_uniq.insert(t)

###### a\_eq.insert(t)

##### a.insert(p, t)

##### a.insert(初始列表对象)

##### a\_uniq.emplace(args)、a\_eq.emplace(args)

##### a.emplace\_hint(args)

##### a.erase(迭代器)

###### e.erase(q1, q2)

##### a.clear( )

##### 键值k相关操作

###### a.erase(k)

a.find(k)

a.count(k)

###### a.lower\_bound(k)

a.upper\_bound(k)

###### a.equal\_range(k)

###### a.operator[ ](k)

仅限于map

### 无序关联容器

#### 无序关联容器：unordered\_set、unordered\_multiset、unordered\_map、unordered\_multimap

#### 为无序关联容器定义的类型

##### X::key\_type

##### X::key\_equal

##### X::hasher

##### X::local\_iterator

###### X::const\_local\_iterator

##### X::mapped\_type

#### 为无序关联容器定义的操作

##### X(n, hf, eq)、X a(n, hf, eq)、X(i, j, n, hf, eq)、X a(i, j, n, hf, eq)

##### b.hash\_function( )

##### b.key\_eq( )

##### b.bucket\_count( )

###### b.max\_bucket\_count( )

##### b.bucket(键值)

##### b.bucket\_size(n)

##### b.begin(n)、b.end(n)、b.cbegin(n)、b.cend(n)

##### b.load\_factor( )

###### b.max\_load\_factor( )、b.max\_load\_factor(z)

##### a.rehash(n)

##### a.reserve(n)

## STL函数

### 非修改式序列操作

#### all\_of( )、any\_of( )、none\_of( )

#### for\_each( )

#### find( )、find\_if( )、find\_if\_not( )

##### find\_end( )

###### find\_first\_of( )

#### adjacent\_find( )

#### count( )、count\_if( )

#### mismatch( )

#### equal( )

#### is\_permutation( )

#### search( )

##### search\_n( )

### 修改式序列操作

#### copy( )、copy\_n( )、copy\_if( )、copy\_backward( )

#### move( )、move\_backward( )

#### swap( )、swap\_ranges( )

##### iter\_swap( )

#### transform( )

#### replace( )、repalce\_if( )、replace\_if( )、replace\_copy( )、replace\_copy\_if( )

#### fill( )、fill(n)

#### generate( )、generate\_n( )

#### remove( )、remove\_if( )、remove\_copy( )、remove\_copy\_if( )

#### unique( )、unique\_copy( )

#### reverse( )、reverse\_copy( )

#### rotate( )、rotate\_copy( )

#### shuffle( )

##### random\_shuffle( )

#### partition( )

##### stable\_partition( )

#### partition\_copy( )

#### partition\_point( )

### 排序和相关操作

#### sort( )、stable\_sort( )、partial\_sort( )、partial\_sort\_copy( )

#### is\_sorted( )

##### is\_sorted\_until( )

#### nth\_element( )

#### lower\_bound( )

##### upper\_bound( )

#### equal\_range( )

#### binary\_search( )

#### merge( )

##### implace\_merge( )

#### includes( )

#### set\_union( )

##### set\_intersection( )

#### set\_defference( )

##### set\_symmetric\_difference( )

#### make\_heap( )

##### push\_heap( )

###### pop\_heap( )

sort\_heap( )

#### is\_heap( )

##### is\_heap\_until( )

#### min( )

##### max( )

###### minmax( )

#### min\_element( )

##### max\_element( )

###### minmax\_element( )

#### lexicographic\_compare( )

#### next\_permutation( )

##### previous\_permutation( )

### 通用数字运算

#### accumulate( )

#### inner\_product( )

#### partial\_sum( )

#### adjacent\_difference( )

#### iota( )