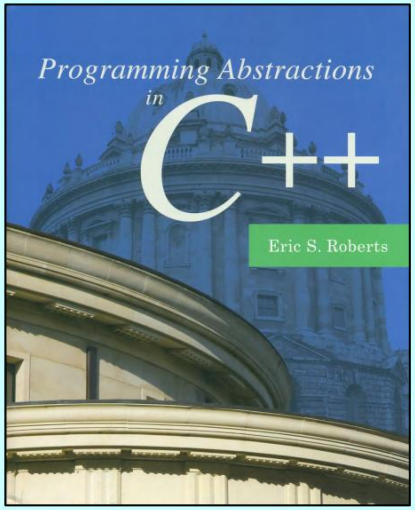
第2章

函数和库

你的图书馆就是你的天堂。

Desiderius Erasmus,费舍尔在鹿特丹的研究， 1524 年



图书馆

- 现代编程依赖于库的使用。当您创建一个典型的应用程序时,您只编写了一小部分代码。

- 在计算机科学中,图书馆是非易失性的集合
计算机程序使用的资源,通常用于开发软件。

- 库也是行为实现的集合,用一种语言编写,具有良好定义的接口,通过该接口调用行为。

- 如果您想成为一名高效的程序员,您至少需要花费与学习语言本身一样多的时间来学习库。

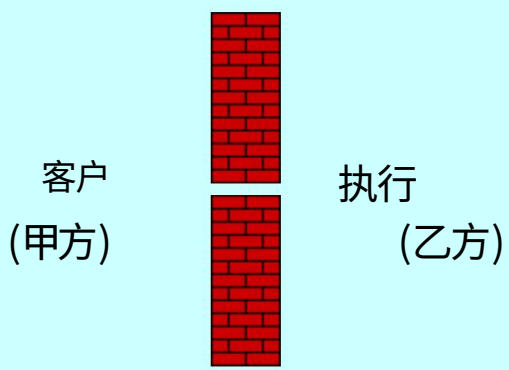
图书馆

- 可以从两个角度查看库。代码
  使用库称为客户端。库本身的代码称为实现。

- 客户端和实现的交汇点称为接口，它既是屏障又是沟通渠道：

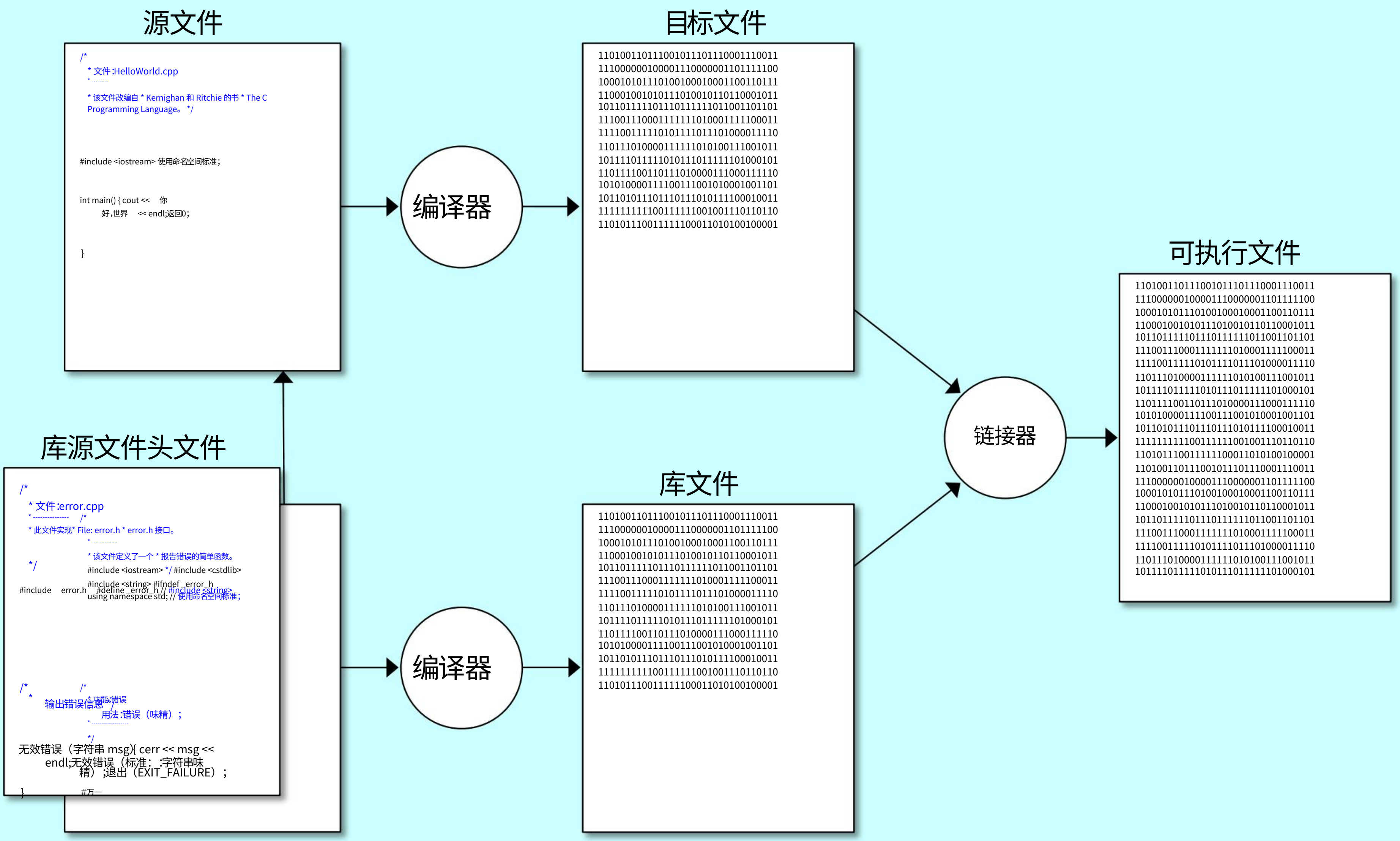
  – 共享和重用

  – 抽象和隐藏



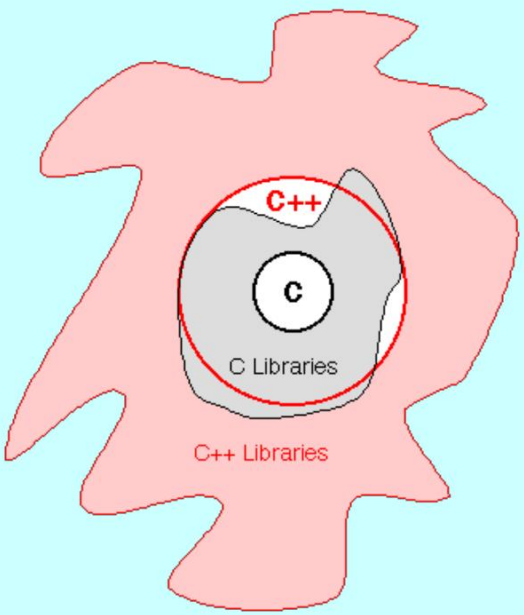
Pyramus 和 Thisbe

编译过程



C与C++的关系

- C++ 的基本设计原则之一是将C 作为子集包含在内,这也是C++成功的部分原因。这种设计策略使得逐步将应用程序从 C 转换为 C++ 成为可能。



- 这种策略的缺点是C++ 的设计不如其他方式一致和集成。

C++ 标准库简介

·类和函数的集合，它们是用核心语言编写的，也是C++ ISO 标准本身的一部分。
C++ 标准库的特性在std命名空间中声明

– 容器：vector、queue、stack、map、set等。
– 通用：算法、函数、迭代器、内存等。
– 字符串
– 流和输入/输出：iostream、fstream、sstream 等。
– 本地化
- 语言支持
- 线程支持库
– 数值库
– C 标准库：cmath、cctype、cstring、cstdio、cstdlib 等。

<cmath>库中的有用函数

绝对值（x）	返回x的绝对值。
平方（x）	返回x的平方根。
楼层(x)	返回小于或等于x 的最大整数。
细胞(x)	返回大于或等于x 的最小整数。
exp(x)	返回x (ex)的指数函数。
日志（x）	返回x的自然对数（以 e 为底）。
日志10(x)	返回x的常用对数（以 10 为底）。
pow(x, y)	返回xy .
cos(θ) sin(θ)	返回弧度角theta 的三角余弦值。
tan(θ)	返回弧度角theta 的正弦值。
	返回弧度角theta 的正切值。
atan(x)	返回x的主反正切。
atan2(y, x)返回y除以x的反正切。	

http://www.cplusplus.com/reference/

http://www.cppreference.com/

斯坦福图书馆简介

- C++ 编程抽象附带了斯坦福大学开发的各种库包,使学生更容易学习基本概念。
  这些库包括但不限于以下内容:

console.h将标准输入	和输出重定向到控制台窗口。
错误.h	支持错误报告和恢复。
gwindow.h实现一个	简单的、可移植的、面向对象的图形模型。
simpio.h	Suoports 改进了输入操作的错误检查。
strlib.h扩展字符串操	作集。

- 您可以从以下表格中了解有关这些库的更多信息
  教科书中的功能,来自基于网络的文档,或通过阅读界面。

- 最新代码和文档可在此处获得:
  https://web.stanford.edu/class/cs106b/library/documentation/

从客户到实施者

- 可以从两个角度查看库。使用库的代码称为客户端。库本身的代码称为实现。

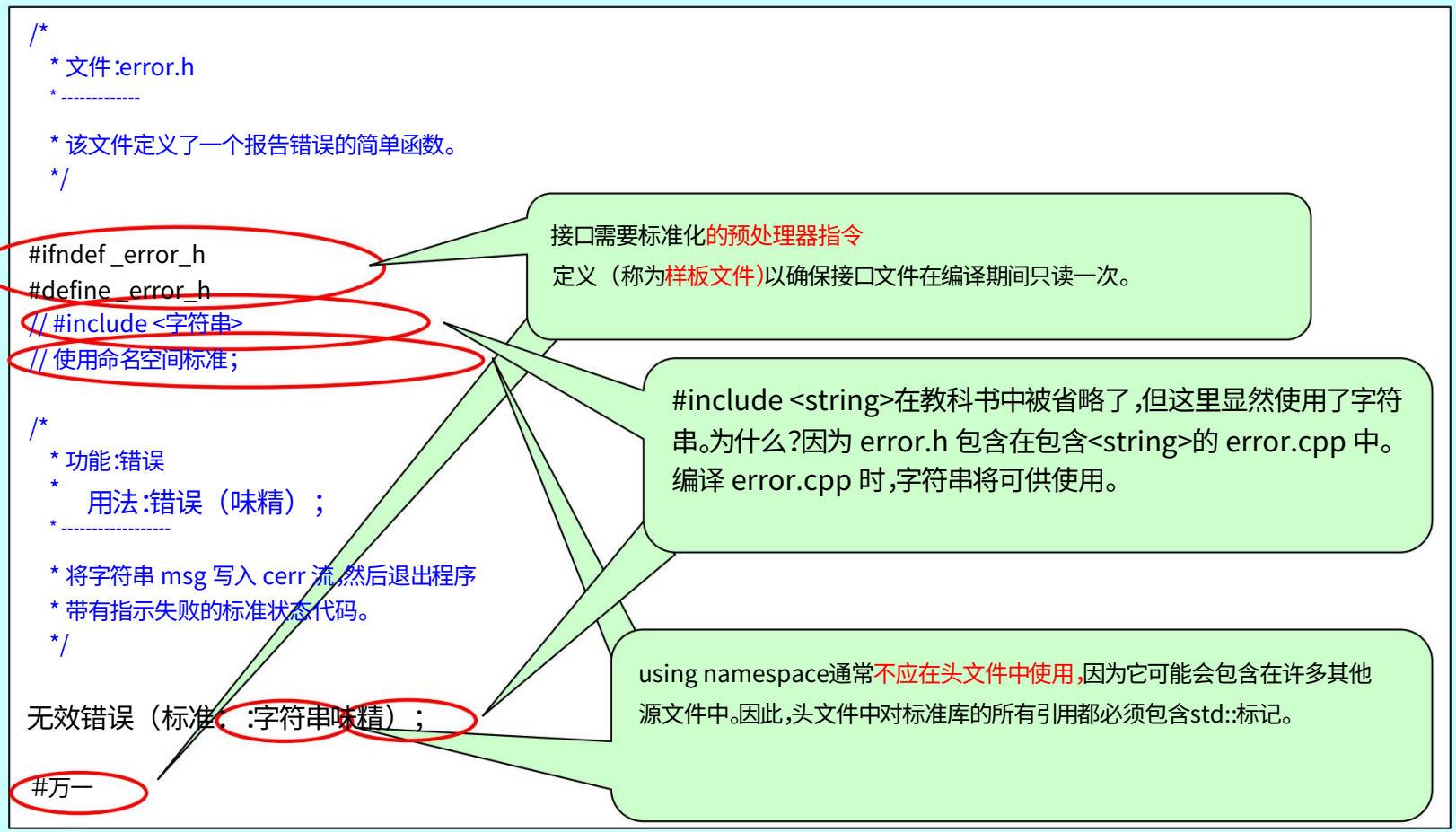
- 因此,调用库提供的函数的程序员也被称为该库的客户端。实现库的程序员称为实现者。 · 当我们阅读本书的各个章节时,您将有一个

  有机会从这两个角度查看几个库（例如,集合类）,首先作为客户端,后来作为实现者。

- 我们现在将从客户的角度切换到实施者的角度。

创建库接口

- 在 C++ 中,通过使用后缀为.h的接口文件向客户端提供库,该后缀表示头文件,如下面的error.h文件所示:



实现库接口

- 在 C++ 中,头文件仅包含
  导出的函数。这些函数的实现出现在相应的.cpp文件中：

```
/*
  * 文件:error.cpp
  * ----------
  * 该文件实现了error.h接口。
  */

#include <iostream> #include
<cstdlib> #include <string>
#include    error.h

使用命名空间标准；

/*
  * 此函数将错误消息写入 cerr 流并
  * 然后退出程序。 EXIT_FAILURE 常量定义在
  * <cstdlib> 表示标准故障代码。
  */

无效错误（字符串味精){
      cerr << msg << endl;
      退出（EXIT_FAILURE）；
}
```

// cerr, endl
// 退出,EXIT_FAILURE
// 细绳

实现文件通常包括它们自己的接口。这也是 error.h 不包含<string>但仍然有效的原因。

using namespace应始终在包含所有库和头文件之后出现,因为它会影响其后的所有代码。

导出类型

- 让我们创建一个导出枚举类型之一的接口,例如用于对四个标准罗盘点进行编码的Direction类型:

```
/*
  * 文件:direction.h
  * ----------------
  * 此接口导出一个名为 Direction 的枚举类型,其 * 元素是四个罗盘点:NORTH、EAST、SOUTH 和 WEST。 */



#ifndef _direction_h
#define _direction_h

#include <字符串>

/**
      类型:方向
  * ---------------
  * 此枚举类型用于表示四个罗盘方向。 */


枚举方向{北,东,南,西};
```

```
/*
   * 函数:leftFrom
   * 用法:方向 newdir = leftFrom(dir);
   * -------------------------
   * 返回参数左侧的方向。
   * 例如,leftFrom(NORTH) 返回 WEST。 */

方向 leftFrom(方向 dir);

/*
   * 功能:rightFrom
   * 用法:方向 newdir = rightFrom(dir);
   * ----------------------------------------
   * 返回参数右侧的方向。
   * 例如,rightFrom(NORTH) 返回 EAST。 */

方向 rightFrom(方向 dir);

/*
   * 函数:方向字符串
   * 用法:string str = directionToString(dir);
   * ------------------------------
   * 以字符串形式返回方向的名称。 */

std::string directionToString(Direction dir);
```

```
/*
 *      操作员 :<<
 *      用法 :cout << 目录 ;
 * ------------------
 * 重载 << 运算符 ,使其能够显示方向值。
 */

std::ostream & operator<<(std::ostream & os, Direction dir);

/*
 *      运算符 :++
 *      用法 :  ++目录
 * ------------
 * 重载 ++ 运算符的前缀版本以使用 Direction
 * 值。
 */

方向运算符++(Direction & dir);

/*
 *      运算符 :++
 *      用法 :  dir++
 * ------------
 * 重载 ++ 运算符的后缀版本以使用 Direction
 * 值 ,以支持 ,例如 ,成语
 *
 *
 *              for (方向 dir = NORTH; dir <= WEST; dir++) 。
 */

方向运算符++(Direction & dir, int);

#万一
```

导出类型

- 方向库的实施

```
/*
 * 文件:direction.cpp
 * ----------
 * 该文件实现了direction.h 接口。 */


#include <string> #include
   direction.h     using namespace
std;

/*
 * 实现笔记:leftFrom、rightFrom
 * ----------

 * 这些函数使用余数运算符循环遍历枚举类型的内部值。请注意,leftFrom * 函数不能从方向减去 1,因为结果 * 可能是负数;
 添加 3 可以达到相同的效果,但 * 确保值保持为正。 */




方向 leftFrom(Direction dir) { return Direction((dir + 3) % 4);

}
方向 rightFrom(Direction dir) { return Direction((dir + 1) % 4);

}
```

```
/*
  * 实现说明:directionToString
  * ----------------------------------------
  * 大多数 C++ 编译器都需要默认子句以确保此 * 函数始终返回一个字符串,即使方向不是合法值之一。 */



string directionToString(Direction dir) { switch (dir) { case NORTH:
       return    NORTH   ;案例东:返回“东”;案例南:返回“南”;案例西:
         返回“西”;默认值:返回“？？？”； }




}

/*
  * 实施说明:<<
  * ------------
  * 此运算符必须在打印 * 值后通过引用返回流。运算符 << 返回此流,因此函数 * 可以实现为单行。 */



std::ostream & operator<<(std::ostream & os, Direction dir) { return os << directionToString(dir);

}
```

```
/*
  * 实施说明:++
  * ----------
  *此运算符的签名中的 int 参数是C++ 编译器用来标识运算符后缀形式的标记。注意 * 增加包含 WEST 的变量后的值将 * 超出方向范围。
  如果 * 此运算符仅用于为其定义的 for 循环习惯用法,则该事实不会引起问题。 */

方向运算符++(Direction & dir) { dir = Direction(dir + 1);返回目录;


}

方向运算符++(Direction & dir, int) {
        方向老= dir;目录 = 方向（目录 + 1）；
        回归旧;

}
```

如果你想让dir在WEST之后回到NORTH ,你应该如何修改代码?

导出常量

```
/*
 * File: gmath.h
 * -------------
 * This file exports the constant PI along with a few degree-based
 * trigonometric functions, which are typically easier to use.
 */

#ifndef _gmath_h
#define _gmath_h

/* Constants */

extern const double PI;                    stant pi */

/*
 * Function: sinDegrees
 * Usage: double sine = sinDegrees(angle);
 * ---------------------------------------
 * Returns the trigonometric sine of angle expressed in degrees.
 */
```

在 C++ 中,以这种形式编写的常量对于包含它们的源文件是私有的,不能通过接口导出。

要导出常量PI,您需要将关键字extern添加到其定义和接口中的原型声明中。

```
double sinDegrees(double angle);

/*
 * Function: cosDegrees
 * Usage: double cosine = cosDegrees(angle);
 * ----------------------------------------
 * Returns the trigonometric cosine of angle expressed in degrees.
 */

double cosDegrees(double angle);

/*
 * Function: toDegrees
 * Usage: double degrees = toDegrees(radians);
 * ------------------------------------------
 * Converts an angle from radians to degrees.
 */

double toDegrees(double radians);

/*
 * Function: toRadians
 * Usage: double radians = toRadians(degrees);
 * ------------------------------------------
 * Converts an angle from degrees to radians.
 */

double toRadians(double degrees);

#endif
```

导出常量

```
/*
 * File: gmath.cpp
 * ---------------
 * This file implements the gmath.h interface.  In all cases, the
 * implementation for each function requires only one line of code,
 * which makes detailed documentation unnecessary.
 */

#include <cmath>
#include "gmath.h"

extern const double PI = 3.14159265358979323846;

double sinDegrees(double angle) {
   return sin(toRadians(angle));
}

double cosDegrees(double angle) {
   return cos(toRadians(angle));
}

double toDegrees(double radians) {
   return radians * 180 / PI;
}

double toRadians(double degrees) {
   return degrees * PI / 180;
}
```

在 C++ 中,以这种形式编写的常量对于包含它们的源文件是私有的,不能通过接口导出。

要导出常量PI,您需要将关键字extern添加到其定义和接口中的原型声明中。

界面设计原理

·统一。每个库都应该定义一个具有明确统一主题的一致抽象。如果某个功能不适合该主题,则它不应成为界面的一部分。

·简单。界面设计应该为客户简化事情。
就底层实现本身的复杂性而言,接口必须设法隐藏这种复杂性。

·足够。对于客户采用库,它必须提供满足他们需求的功能。如果缺少某些关键操作,客户可能会决定放弃它并开发自己的工具。 ·一般。一个设计良好的库应该足够通用,以满足许多不同客户的需求。为一个客户端提供狭义定义的操作的库不如可以在许多不同情况下使用的库那么有用。

·稳定。由库导出的类中定义的函数
即使图书馆在发展,也应该保持完全相同的结构和效果。改变图书馆的行为会迫使客户改变他们的程序,这会降低它的效用。

设计一个随机数库

- 非确定性行为在计算机上很难实现。计算机以精确、可预测的方式执行其指令。如果你给一个计算机程序相同的输入，它每次都会产生相同的输出，这不是你想要的非确定性程序。

- 鉴于在计算机中很难实现真正的非确定性，本章中描述的random.h接口等库必须通过执行满足以下标准的确定性过程来模拟随机性：

  1. 该过程产生的价值应该是困难的供人类观察者预测。
  2. 这些值应该看起来是随机的，因为它们应该通过随机性的统计测试。

从 C++11 开始，C++ 标准中有一个<random>库。但是这里我们只使用<cstdlib>来实现一个简化版本。

- 因为该过程不是真正随机的，所以由random.h接口生成的值被称为伪随机。

设计一个随机数库

- <cstdlib>中的函数rand没有参数,并且
  随机返回 0 到RAND_MAX之间的整数。这对客户来说是不够的。

- 选择正确的功能集

  ü选择指定范围内的随机整数

  ü在指定范围内随机选择一个实数

  ü模拟具有特定概率的随机事件

- C++ 库（以及更早的 C 库)的设计者决定rand应该在每次运行
  程序时返回相同的随机序列,以便可以使用rand

  以一种确定的方式来支持调试。

  ü设置随机数的初始值（种子）
  生成过程,例如程序运行的时间

random.h接口_

```
/*
 * 文件:random.h
 * ----------
 * 此文件导出用于生成伪随机数的函数。 */


#ifndef _random_h
#define _random_h

/*
 * 函数:随机整数
 * 用法:int n = randomInteger(low, high);
 * ----------
 * 返回一个从低到高（含)范围内的随机整数。 */


整数随机整数（整数低,整数高）；
```

random.h接口_

```
/**
 文件：randomReal.h * 函数：
 randomReal
 * ----------
 用法：double d = randomReal(low, high);
 *
 * 此文件导出用于生成伪随机数的函数。 */ * 返回半开区间 [low, high) 中的随机数。

 * 半开区间包括第一个端点，但不包括#ifndef _random_h * 秒。 #define _random_h

 */

/*
double randomReal(double low, double high);
 * 函数：随机整数 /*
 *
 用法：int n = randomInteger(low, high);
 * 功能：随机机会 * ----------
 *
 用法：if (randomChance(p)) . . .
 * 返回一个从低到高（包含）范围内的随机整数。 */ * 以 p 表示的概率返回真。这
 * ----------

 *
 参数 p 必须是介于 0 之间的浮点数（从不） int randomInteger(int low, int high); * 和 1（总是）。 */
```

布尔随机机会（双 p）；

random.h接口_

```
/**/
  函数:randomReal()函数:randomBool 用
  *   用法:double d = randomReal(low, high);
  * ------------------- * ----- ----------------------

  * 返回50%的概率返回[low, high)中的一个随机数true 或 false。*/ * 半开区间包括第一个端点,但不包括第二个端点。

布尔随机布尔（）；
  */ /
*
double randomReal(double low, double high);
  * 功能:设置随机种子
  *
    用法： setRandomSeed(seed);
/** ---------------
  * 功能:randomChance * 将内部随机数种子设置
  *为指定值。
    用法： if (randomChance(p)) ...
  * 您可以使用此功能设置特定的起点
  * -------------------
  * 用于伪随机序列或确保程序 * 以 p 指示的概率返回 true。 * 行为在调试阶段是可重复的。
  *
    参数 p 必须是介于 0（从不)*/ * 和 1（总是)之间的浮点数。 */

无效 setRandomSeed(int 种子);

布尔随机机会（双 p）； #万一
```

random.cpp实现_

TUTORIAL

```
/*
 * 文件:random.cpp
 * ----------------
 * 该文件实现了 random.h 接口。 */


#include <cstdlib> #include
<cmath> #include <ctime>
#include random.h using
namespace std;


/* 私有函数原型 */

无效初始化随机种子（）；
```

random.cpp实现_

```
/**
 * 文件：randomInteger.cpp * 实现说明：
 * ---------- * ---------- ---
 * 这个实现了 random.h 的代码分两步生成数字： */ * 1. 在 [0 .. 1) 范围内生成随机实数 d。 #include <cstdlib> * 2. 将数
 * 字缩放到范围 [0 .. N)。 #include <cmath> * 3. 翻译数字,使范围从低开始。 #include <ctime> * 4. 将结果截断为下
 一个较小的整数。 #include random.h 使用命名空间标准； * 由于 * 表达式 RAND_MAX + 1 和表达式 high -
low + 1 都可以 /* 私有函数原型 */ * 溢出整数范围,因此实现变得复杂。 */
 *
静态无效初始化随机种子（）； int randomInteger(int
low, int high) { initRandomSeed();双 d = rand() / (双 (RAND_MAX)
    + 1);双 s = d * (双(高) - 低 + 1);返回int（地板（低+ s））；
}
```

random.cpp实现_

```
/**
 * 实现说明：randomReal
 * ---------- * ----------
 * randomReal的代码与randomInteger的代码类似，
 * 没有最后的转换步骤。 */ * 1. 在 [0 .. 1) 范围内生成一个随机实数
 d。
 * 2. 将数字缩放到 [0 .. N) 范围内。 double randomReal(double low, double
high){ * 的整数。 双 d = rand() / (双 (RAND_MAX) + 1);双 s = d * (高 - 低);返回低 + s;
    得复杂。 * 表达式 RAND_MAX + 1 和表达式 high - low + 1 可以} * 溢出整数范围。 */
 *

int randomInteger(int low, int high) { initRandomSeed();双 d =
    rand() / (双 (RAND_MAX) + 1);双 s = d * (双(高) - 低 + 1);返回int
    （地板（低+ s））；

}
```

random.cpp实现_

```
/** /
  * 实施说明:randomReal randomChance
  * ------------------------------- * ---------------- ------------------
  * 没有最后的转换步骤类似结果是否小于请求的概率。*/ * /randomChance 的代码调用 randomReal(0, 1) 然后检查

bool randomChance(double p) { double high) { initRandomSeed();初始化随机种子
    * (高 ;双 d;返回低 + s;(双 (RAND_MAX) + 1);返回随机实数(0, 1) < p;双 s = 0.5);

}

}
```

random.cpp实现_

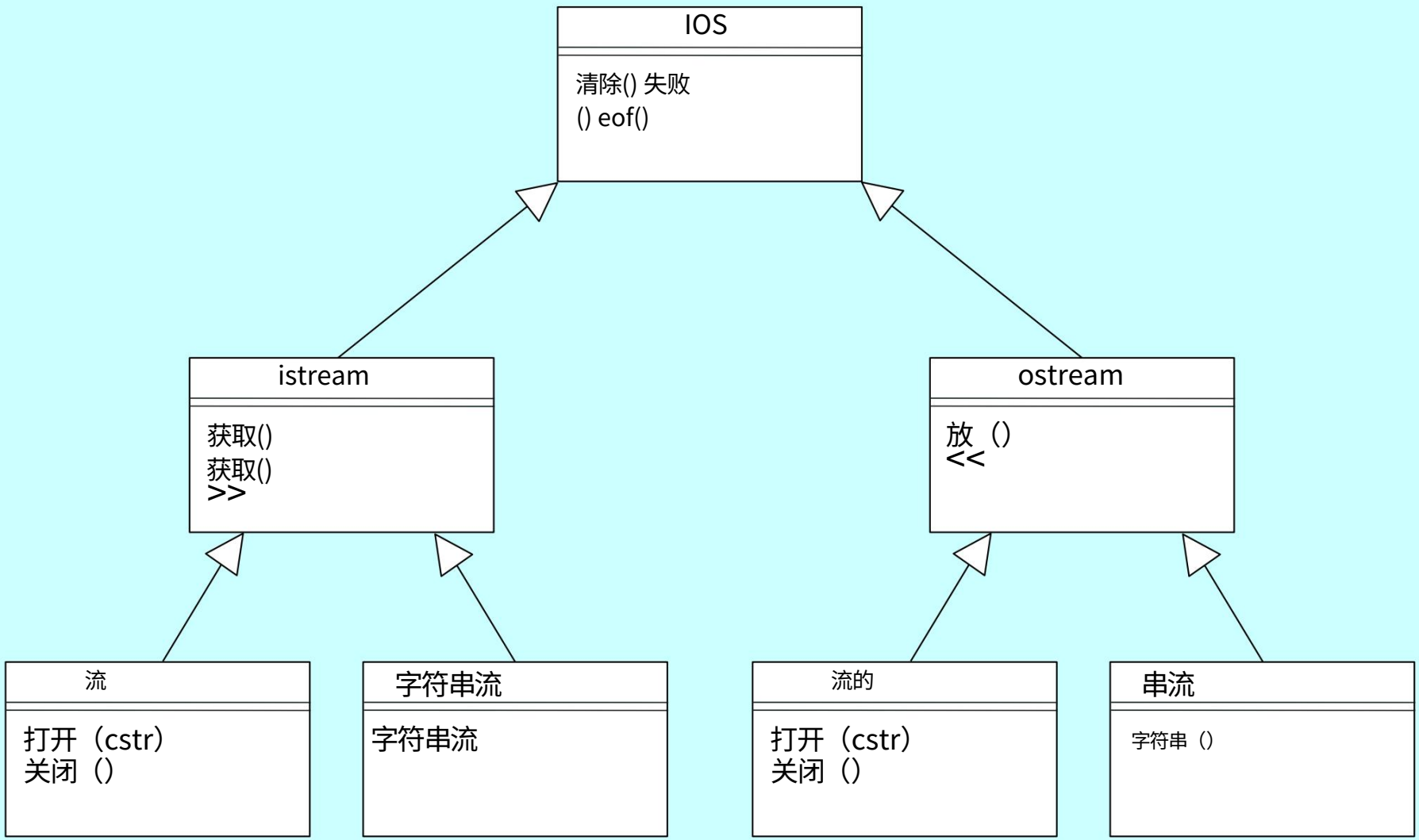
```
/** /
  * 实施说明：setRandomSeed / randomChance
  * ---------------------------------- * -------------- ---------------------
  * srand 需要调用 initRandomSeed 函数只是将它的参数设置为结果是否小于请求的概率，代码调用 randomReal(0, 1) 然后检查 */

bool randomChance(double p) { initRandomSeed();
{ initRandomSeed(); 无效 setRandomSeed(int 返回随机实数(0, 1) < p; srand
      （种子）；

} } bool randomBool() { return
      randomChance(0.5);
}
```

random.cpp实现_

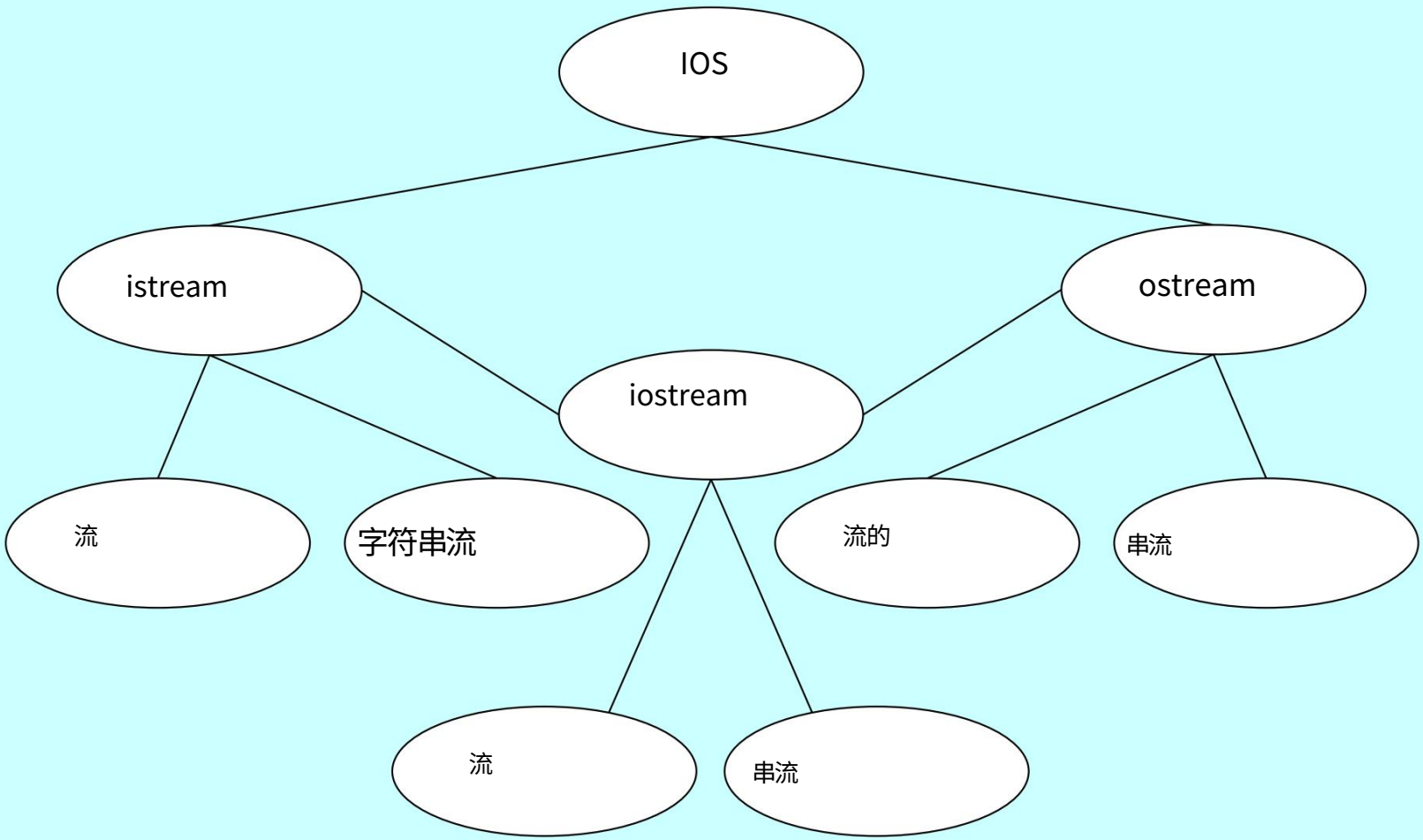
```
/** /
 * 实现说明:initRandomSeed
 * ------------------------------------ * ------------- ----------------------
 * initRandomSeed 将种子设置为当前时间。setRandomSeed 函数第一次调用时 initialized 为 false,
 为 false,*/ * 所以种子设置为当前时间。 */

无效 setRandomSeed(int 种子) { initRandomSeed();
初始化=假;void initRandomSeed() { if (!initialized) { srand(int(time(NULL))); 静态布尔
        初始化=真;
}

        }
}
```

静态变量的生命周期从程序流第一次遇到声明时开始,并在程序终止时结束。编译器只分配一个initialized的副本,它只初始化一次,然后由所有对 initRandomSeed 的调用共享。这确保了初始化步骤必须执行一次且仅执行一次。

流层次结构的简化视图

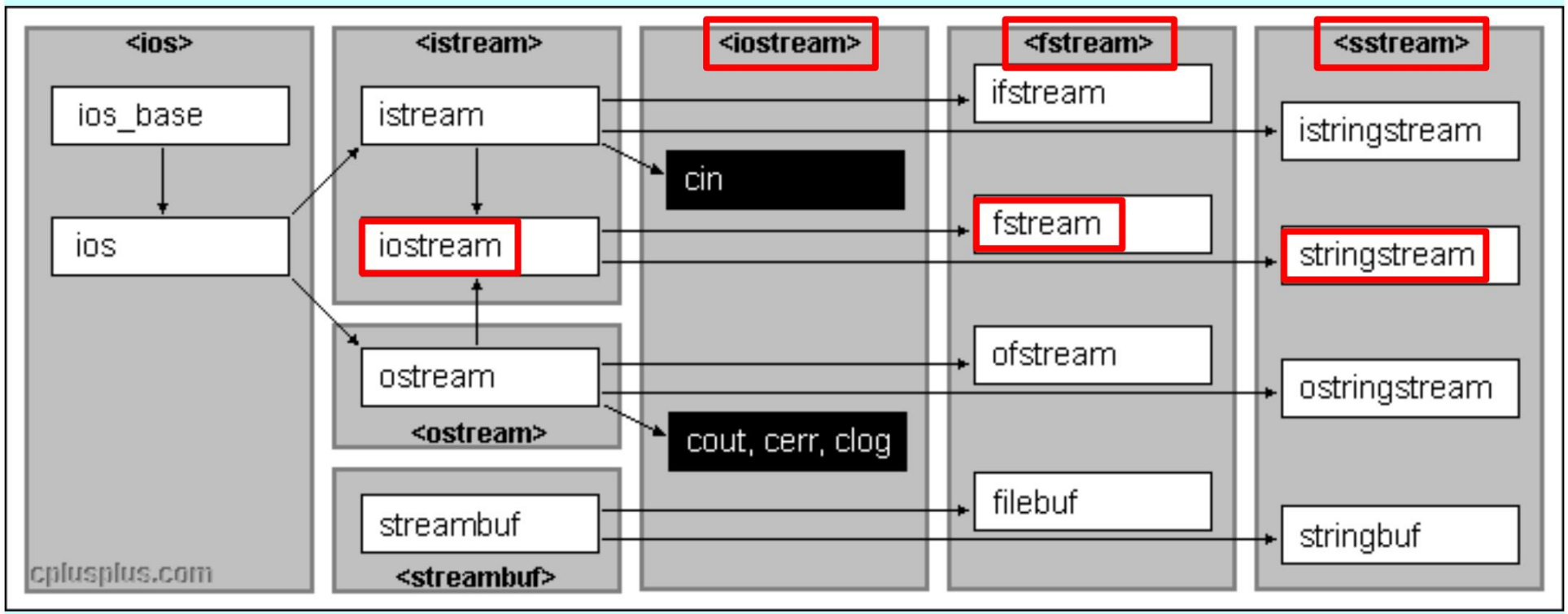


流层次结构中的选定类



库与类层次结构

- 库与类层次结构:有趣的是,C++ 流库不是基于类层次结构进行组织的。



- 组织图书馆以方便用户,并设计数据完整性和执行效率的类。

The End