Modern C++ Camp

现代C++系统研发骨干特训营

李建忠 Boolan

变参模板与折叠表达式

- 变参模板（Variadic Template）：将模板参数定义为可接受任意数目、任意类型的实参；通过递归效应来实现编译时展开。
  - 变参模板完美转发
  - 变参表达式
  - 变参索引
  - 变参类模板： Tuple、 Variant;
  - 变参推导
  - 变参基类

- 折叠表达式：遍历参数包中所有元素，可简化变参模板的实现。

SFINAE 习语

- 替换错误不是失败 SFINAE (substitution failure is not an error)
- 模板编译过程，如果替换参数T 产生失败或无意义值，则发生替换错误，忽略失败（不是错误），不产生该模板参数对应的构造。
- 注意，替换过程 和 模板选择实例化的过程是不同的。
- 通过SFINAE，可以根据参数是否满足需求，来决定是否生成某些构造的重载。

enable_if 表达类型约束

- enable_if是个traits, 根据参数产生编译时true or false结果

- 如果表达式产生true：
  - 如果传递了第二个类型参数，那么产生相关类型;
  - 如果没有传递第二个类型参数，那么产生void；
- 如果表达式产生false，那么使用SFINAE忽略错误，不定义相关类型，相关模板定义直接不产生

- std::enable_if_t<> 可以忽略typename和type，更方便.
- 可使用using 别名简化使用

Boolan
高端IT咨询与教育平台

现代C++系统研发骨干特训营

模块五、泛型编程与C++20

C++ 20 概念

- C++概念是一种显式的类型接口合同，在编译时执行类型约束检查。是泛型编程的灵魂。

- 概念可以帮助更好的理解泛型组件之间的合约
- 易维护，易沟通、更友好的出错信息、
- 有概念约束的版本，可以参与重载辨析，相对于通用版本是一个特化版（更优先辨析）
- 概念是比Traits、编译时表达式都更强大的抽象。

概念定义


```
template<typename T>
concept MyConcept = ... ;
```

- 概念仅仅是约束，不是代码、没有类型、存储、声明周期、地址....
- 概念的本质是一组对类型参数T的编译时表达式求职 true 或false
- 参数T不可以有额外约束，不能本身再是概念（用概念来定义概念）
- 不可以在函数内定义概念。
- 概念可在下面使用：模板类型参数约束；auto；复合需求表达式

requires表达式约束

- requires表达式可以指定多个类型约束需求。多个约束没有顺序关系，可以是下列的组合：
  - 编译时boolean值表达式：类型判断式、编译时变量或函数
  - requires表达式：类型定义、有效表达式、表达式产生的类型需求
  - concept

- requires表达式执行的是编译时检查，对运行时代码没有任何影响，没有性能损失。

- 多个约束之间可以&&（与）、||(或），一个约束可以指定多个参数

概念使用

- 概念可在下面使用：
  - 模板类型参数约束；
  - auto 参数；
  - 复合需求表达式
  - 概念可以归并其他概念

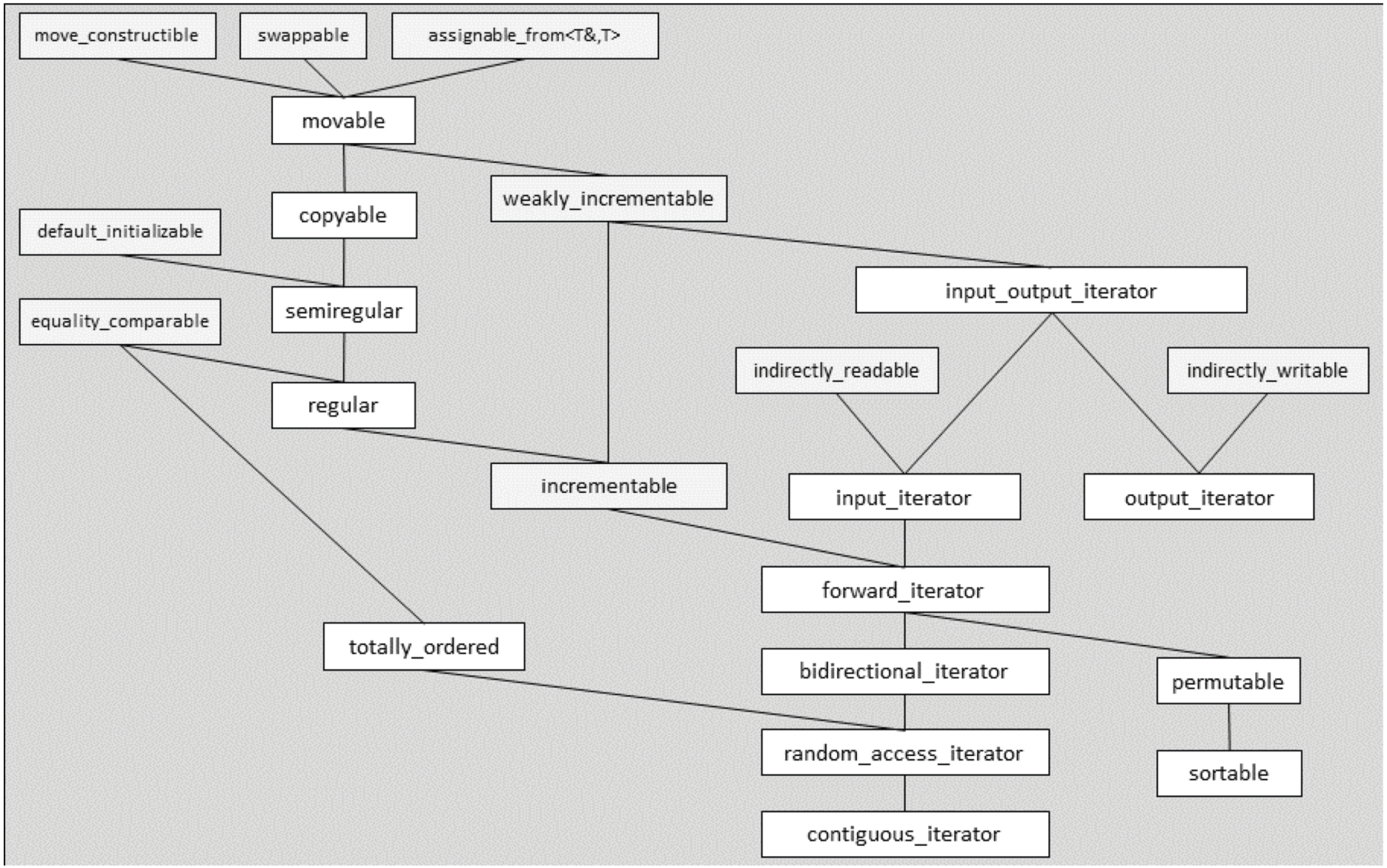
- 概念可以用来约束：
  - 模板类参数
  - 模板函数参数
  - 返回值
  - 约束可调用构造：函数指针、函数对象、lambda
  - 约束非类型参数

标准库预定义concept

- 算数概念：整数、浮点、有无符号....
- 对象概念：拷贝、移动、缺省初始化...
- 类型概念：相同、可转换、继承、构造、赋值...
- 比较概念：相等、比较、排序、大于、小于、不等....

- 迭代器
- 范围
- 仿指针
- .....

标准库概念关系


move_constructible
swappable
assignable_from<T&,T>
movable
copyable
weakly_incrementable
default_initializable
semiregular
input_output_iterator
equality_comparable
regular
indirectly_readable
indirectly_writable
incrementable
input_iterator
output_iterator
forward_iterator
totally_ordered
bidirectional_iterator
permutable
random_access_iterator
sortable
contiguous_iterator

auto 与decltype

- 当变量有合适的初始化器时，可以直接使用auto。
- 但有时候既希望编译器自动推断类型，又不希望、或者无法定义初始化变量，就应该使用decltype
  - 返回值类型依赖于形参类型的函数模板
  - decltype（auto）可以从初始化表达式中推导出来类型。

- decltype(expr) 推断的结果时expr的声明类型。注意 当类型为T的左值表达式，decltype的推断类型为 T&