模板可以关联到约束,它指定了对模板实参的一些要求,这些要求可以被用于选择最恰当的函数重载和模板特化.这种要求的具名集合被称作为概念(concept)(C++20起).每个概念都是谓词,在编译时求值,并在自己被用作约束时成为模板接口的一部分.

概念

概念定义拥有以下格式

```
template <模板实参列表>
concept 概念名 属性(可选) = 约束表达式;
```

定义一个概念Numeric,约束T的类型为整形或浮点型:

```
template<typename T>
concept Numeric = std::integral<T> || std::floating_point<T>;
```

概念不能被显式实例化,显式特化或部分特化

概念可以在标识表达式中命名.该标识表达式的值在满足约束表达式时是 true，否则是 false

```
auto x = Numeric<decltype(1)>;
std::print("{}\n", x);
```

输出结果:

```
true
```

概念在作为以下内容的一部分时也可以在类型约束中被命名

- 类型模板形参声明

```
template<Numeric T1>
T1 demo(T1 x, T1 y) {
    return x + y;
}
```

- 占位类型说明符

```
Numeric auto a = 1;
```

- 复合要求

合取

两个约束的合取是通过在约束表达式中使用 `&&` 运算符来构成的：

两个约束的合取只有在两个约束都被满足时才会得到满足。合取从左到右短路求值（如果不满足左侧的约束，那么就不会尝试对右侧的约束进行模板实参替换：这样就会防止出现立即语境外的替换所导致的失败）

```
template<typename T>
concept demo1=std::integral<T> && sizeof(T)>1;
```

析取

两个约束的析取，是通过在约束表达式中使用 `||` 运算符来构成的：

如果其中一个约束得到满足，那么两个约束的析取的到满足。析取从左到右短路求值（如果满足左侧约束，那么就不会尝试对右侧约束进行模板实参替换）。

```
template<typename T>
concept demo2 = std::integral<T> || std::floating_point<T>;
```

requires 子句

关键词 requires 用来引入 *requires 子句*，它指定对各模板实参，或对函数声明的约束。

```
template<typename T>
requires Numeric<T>
constexpr T foo(T a, T b) {
    return a + b;
}
```

```
template<typename T>
constexpr T foo(T a, T b) requires Numeric<T> {
    return a + b;
}
```

这种情况下，关键词 *requires* 必须后随某个常量表达式（因此可以写成 requires true），但这是为了使用一个具名概念（如上例），具名概念的一条合取/析取，或者一个 *requires 表达式*。