C++17新特性std::optional

Background

今天在写cs144的Lab的时候发现一个头文件里有 `std::optional` ，我在脑海里翻找暑假学cppreference的时候的记忆，结果发现完全没有这个东西的半点痕迹...😇

了解了一番后发现这个东西还挺实用的，将空值和正常含值的变量聚合到了一起。在我们约定了某些值(**Magic Value**，比如-1，空字符串"")表示"nothing"的场景，使用 `std::optional` 可以起到很好的替代作用。使用Magic Value的弊端是不言而喻的，我们会默认这些Magic Value是我们**不需要的**，抑或我们赋予了这些值“**有异常**”的意义，但这一切都是**口头约定**，并没有从**语法层面**加以约束。其次，对于某些类型，我们并不能找到类似的Magic Value，或者就算找到后，这些值也不能轻松地被创造出来，这都给我们在使用Magic Value时带来风险。

记录一下 `std::optional` 的用法：

std::optional

类模板 `std::optional` 管理一个可选 的容纳值，既可以存在也可以不存在的值。

一种常见的optional使用情况是一个可能失败的函数的返回值。与其他手段，如 `std::pair<T, bool>` 相比，optional良好地处理构造开销高昂的对象，并更加可读，因为它显式表达意图。

任何一个 `optional<T>` 的实例在给定时间点要么含值，要么不含值。

如果一个 `optional<T>` 含值，那么保证值作为 optional 对象所用空间的一部分分配，即不会发生动态内存分配。从而optional对象模拟一个对象，而非指针，尽管定义了 operator*() 和 operator->() 运算符。

当一个 `optional<T>` 对象被按语境转换到bool 时，对象含值的情况下转换返回 true，对象不含值的情况下返回 false。

optional 对象在下列条件下含值：

1. 对象被以 T 类型值或另一含值 的 optional 初始化/赋值。

对象在下列条件下不含值：

1. 对象被默认初始化。
2. 对象被以 std::nullopt_t 类型值或不含值 的 optional 对象初始化/赋值。
3. 调用了成员函数 reset()。

不存在引用的 optional：如果以引用类型实例化 optional，那么程序非良构。另外，如果以（可有 cv 限定的）标签类型 std::nullopt_t 或 std::in_place_t 实例化 optional，那么程序非良构。

构造函数

std::optional<T>::optional

`constexpr optional() noexcept;` `constexpr optional( std::nullopt_t ) noexcept;`	(1)	(C++17 起)
`constexpr optional( const optional& other );`	(2)	(C++17 起)
`constexpr optional( optional&& other ) noexcept(/* 见下文 */);`	(3)	(C++17 起)
`template < class U >` `optional( const optional<U>& other );`	(4)	(C++17 起) (C++20 前) (条件性 explicit)
`template < class U >` `constexpr optional( const optional<U>& other );`		(C++20 起) (条件性 explicit)
`template < class U >` `optional( optional<U>&& other );`	(5)	(C++17 起) (C++20 前) (条件性 explicit)
`template < class U >` `constexpr optional( optional<U>&& other );`		(C++20 起) (条件性 explicit)
`template< class... Args >` `constexpr explicit optional( std::in_place_t, Args&&... args );`	(6)	(C++17 起)

`template< class U, class... Args >`
`constexpr explicit optional( std::in_place_t,`
`std::initializer_list<U> ilist,`
`Args&&... args );`
	(7)	(C++17 起)

`template < class U = T >` `constexpr optional( U&& value );`	(8)	(C++17 起) (条件性 explicit)

(1) 构造不含值的对象

构造不含值的对象示例 C++

```cpp
std::optional<int> o1;
```

(2) 复制构造函数

复制构造函数示例 C++

```cpp
std::optional<int> o3(o1);
```

(3) 移动构造函数

移动构造函数示例 C++

```cpp
std::optional<int> o4(std::move(o1));
```

(6) 使用对应类型的构造函数

示例 C++

```cpp
std::optional<int> o5(std::in_place, 3, 'A');
```

（7）使用对应类型的 `std::initializer_list`

示例 C++

```cpp
std::optional<int> o6(std::in_place, {'a', 'b', 'c'});
```

赋值函数

重载了 `operator=` ，使用方法就是直接赋值

同时， `std::optional` 的对象op可以通过 `op = {}` 和 `op = nullopt` 变成空optional。

(`op = {}` 通过构造空的 `std::optional` 对象并将它赋值给op)

观察器

`operator->` & `operator*` ： 访问所含值

`operator bool` & `has_value` ： 检查对象是否含值

`value` ： 返回所含值

`value_or` ： 在所含值可用时返回它，否则返回另一个值

修改器

`swap` ： 交换内容

`reset` ： 销毁任何所含值

`emplace` ： 原位构造所含值

非成员函数

`make_optional` ： 创建一个optional对象

`std::swap` ： 特化 `std::swap` 算法