78 How to store ANY data in C++

#将不同类型的数据存储在单变量中系列3/3

本节会讲如何在单个变量中存储任意类型的数据。
你可以用`void*`来处理，然而这里要讲的是一个更好、更安全的方式，它是C++17标准带来的全新处理方式。

1. std::any

它的用法很简单，我们主要要思考的是：为什么会有这个东西，什么时候要用到它，为什么要用`std::any`而不是`std::variant`，这些才是本节最精彩的内容。

```C++
#include <any>

int main()
{
    std::any data;
    data = 2;   // int
    data = "Cherno";   // const char
    data = std::string("Cherno"); // string

    // 如果想提取数据
    std::string string = std::any_cast<std::string>(data);

    std::cin.get();
}
```

如果`std::any`对象包含的值类型与`std::any_cast`的目标类型相匹配，那么`std::any_cast`将返回该值。否则，如果类型不匹配，`std::any_cast`将抛出`std::bad_any_cast`异常。

看起来和`std::variant`很像，它们确实有些类似，可以存储任何类型，有人可能觉得：`std::variant`要求我们列出所有的类型而很糟糕，而在`std::any`这里根本就不需要担心类型。但其实恰恰相反，这才是绝大多数情况下，`std::variant`比`std::any`更好的地方。

事实上，`std::variant`要求列出所有这些类型是很好的，会使得类型安全：

```C++
data = "Cherno";   // 比如忘记显式地写出这是一个字符串
// 然后下行代码会很合理，但试着提取时就遇到问题了：data 是一个 const char*
std::string string = std::any_cast<std::string>(data);

// 而用std::variant的话，就可以正常工作了，因为限定了类型

#include <variant>

std::variant<int, std::string> data;
data = 2;
data = "Cherno";
std::string string = std::any_cast<std::string>(data);
```

那除了`std::variant`需要列出类型之外，`std::any`和它相比还有什么区别吗?

2. std::any和std::variant的区别

当然有，这和它们的存储方式有关。

`std::variant`只是一个类型安全的`union`，意思是它把所有的数据都存储在一个联合体中。

那`std::any`的存储方式呢？让我们进入它的头文件：

> Storage这个似乎用来存储东西的变量实际上是一个Union
>
> ```
> struct _Storage_t {
>     union {
>         unsigned char _TrivialData[_Any_trivial_space_size];
>         _Small_storage_t _SmallStorage;
>         _Big_storage_t _BigStorage;
>     };
>     uintptr_t _TypeData;
> };
> static_assert(sizeof(_Storage_t) == _Any_trivial_space_size + sizeof(void*));
> static_assert(is_standard_layout_v<_Storage_t>);
>
> union {
>     _Storage_t _Storage{};
>     max_align_t _Dummy;
> };
> ```

> 最后深入，发现Small_storage是一个aligned_union_t，发现它是aligned_union的所有类型。
>
> ```
> template <size_t _Len, class... _Types>
> using aligned_union_t = typename aligned_union<_Len, _Types...>::type;
>
> // STRUCT TEMPLATE underlying_type
> template <class _Ty>
> struct underlying_type { // determine underlying type for enum
> ```

所以`std::any`到底做了什么？其实很聪明，对于 *small types* （小类型），它只是把它门存储为一个`union`，这意味着对于小类型来说，它的工作方式和`std::variant`完全相同。

而如果你有一个大类型，它会带你进入大存储空间的`void*`，在这种情况下它会动态分配内存，而动态分配内存不利于性能。

如果你需要更多而存储空间（这里可以看到是>32），`std:any`会动态分配，但是`std::variant`不会。

```
inline constexpr size_t _Any_small_space_size =
  (void*);

template <class _Ty>
inline constexpr bool _Any_is_
```

(global variable) inline c
in namespace std
value: 32

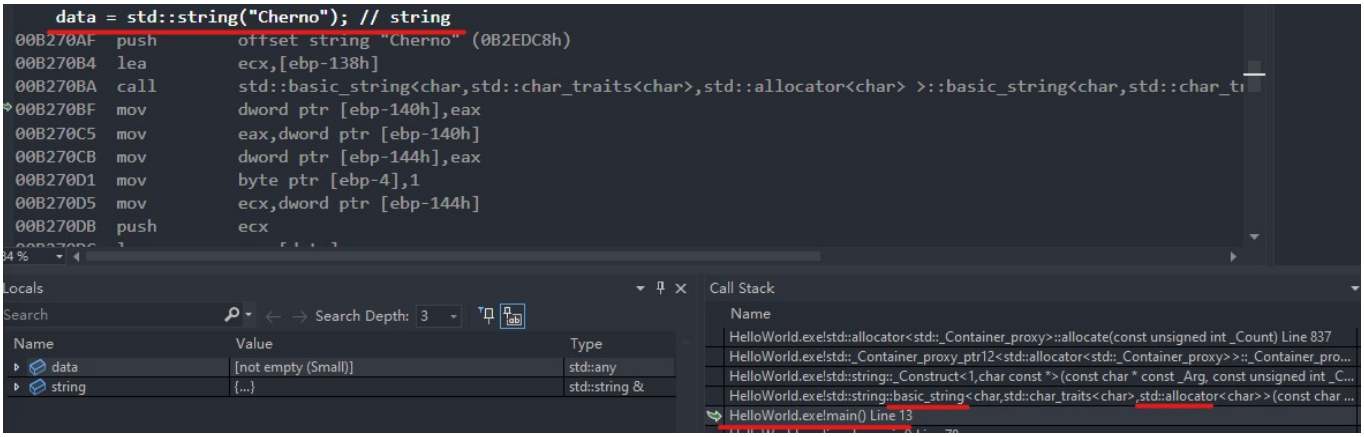
所以换句话说，除了更加类型安全和有一点限制性，`std::variant`在处理较大数据时也会执行得更快，或是你的数据又需要避免动态内存分配。

另一个提高性能的技巧是不要复制数据，之前代码中有字符串的复制，可以换成通过引用返回：

```C++
// 在std::get()中可以直接加一个引用符，但std::any_cast中需要把引用插入到模板参数中，才能很好地优化
std::string& string = std::any_cast<std::string&>(data); // 注意这里模板参数中也要用引用
```

目前的情况下，我们不会得到除了那个string的内存分配，可以简单测试一下：

```C++
void* operator new(size_t size) // 这不是new操作符的很好的实现，因此不要在任何生产代码中使用它
{
    return malloc(size); // 打上断点
}
```

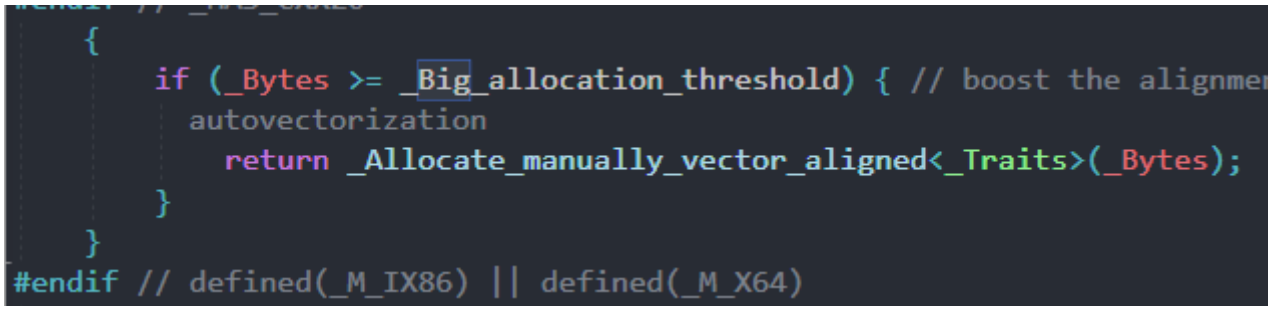


> 可以看到我们确实得到了一个内存分配，来源是std::string，std::basic_string进入allocator分配器。any实际上只做了一个_Construct_in_place转发所有这些参数，basic_string在做再一次的内存分配，它就是这个程序中全部的内存分配了，这个情况下std::any实际没有分配任何东西。

```C++
struct CustomClass
{
    std::string s0, s1;  // 一个是28(x86)
};

int main()
{
    std::any data;
    data = 2;   // int
    data = CustomClass();
//  std::string& string = std::any_cast<std::string&>(data);

    std::cin.get();
}
```



> 在Emplace的过程中，你会看到这是一个大的存储过程，所以会调用new，所以要知道std::any会在需要时进行分配。在这种情况下（>32Bytes），它会调用new和动态内存分配，这里可能编译器把过程抽象了，没得到和视频中一样的效果。

3. any和variant分别什么时候用?

这是一个比较棘手的问题。

如果希望在一个变量中存储多个数据类型，请使用`std::variant`，因为它基本上是`std::any`的类型安全版本，这意味着你不能随意设置它的类型。而且它还不会动态分配内存，这样性能会更好。

如果你需要能够在单个变量中存储任何数据类型，也许你需要重新考虑程序的设计，Cherno的意思是他真的想不出来一个有效用例（就不该用到它），如果你有像缓冲区的数据，不知道这些数据的类型，只是想指向它们，那你可以用`void*`。就目前的情况，Cherno不认为它和`std::optional`以及`std::variant`一样有用，你应该明确意识到`std::any`可能不是你应该经常使用的东西。（观点发表于视频录制的2019年）。