

第30章 alignas和alignof


《现代C++语言核心特性解析》 谢丙堃

不可忽视的数据对齐问题

- 问题代码

```
struct A {  
    char a1;  
    int a2;  
    double a3;  
};  
struct B {  
    short b1;  
    bool b2;  
    double b3;  
};  
int main() {  
    std::cout << "sizeof(A::a1) + sizeof(A::a2) + sizeof(A::a3) = "  
        << sizeof(A::a1) + sizeof(A::a2) + sizeof(A::a3) << std::endl;  
    std::cout << "sizeof(B::b1) + sizeof(B::b2) + sizeof(B::b3) = "  
        << sizeof(B::b1) + sizeof(B::b2) + sizeof(B::b3) << std::endl;  
    std::cout << "sizeof(A) = " << sizeof(A) << std::endl;  
    std::cout << "sizeof(B) = " << sizeof(B) << std::endl;  
}
```

$\text{sizeof}(A::a1) + \text{sizeof}(A::a2) + \text{sizeof}(A::a3) = 13$
 $\text{sizeof}(B::b1) + \text{sizeof}(B::b2) + \text{sizeof}(B::b3) = 11$
 $\text{sizeof}(A) = 16$
 $\text{sizeof}(B) = 16$



内存布局

```
struct A
{
    char a1;
    char a1_pad[3];
    int a2;
    double a3;
};
```

offset	element
0x0000	a1
0x0001	a1_pad[3]
0x0004	a2
0x0008	a3

```
struct B
{
    short b1;
    bool b2;
    char b2_pad[5];
    double b3;
};
```

offset	element
0x0000	b1
0x0002	b2
0x0003	b2_pad[5]
0x0008	b3

C++11标准之前控制数据对齐的方法

- 使用宏获取对齐长度

```
#include <cstddef>
#define ALIGNOF(type, result) \
    struct type##_alignof_trick { \
        char c; \
        type member; \
    }; \
    result = offsetof(type##_alignof_trick, member)

int main() {
    int x1 = 0;
    ALIGNOF(short, x1);
}
```

C++11标准之前控制数据对齐的方法

- 使用模板获取对齐长度

```
#include <cstddef>
template <class T>
struct alignof_trick {
    char c;
    T member;
};
#define ALIGNOF(type) offsetof(alignof_trick<type>, member)

auto x2 = ALIGNOF(void(*)());
```

C++11标准之前控制数据对齐的方法

- 厂商编译器提供获取对齐长度，例如__alignof和__alignof__

```
// MSVC
```

```
auto x1 = __alignof(int);
```

```
auto x2 = __alignof(void(*)());
```

```
// GCC
```

```
auto x3 = __alignof__(int);
```

```
auto x4 = __alignof__(void(*)());
```

C++11标准之前控制数据对齐的方法

- 厂商编译器提供设置对齐长度的方法，例如__declspec(align(x))和__attribute__((aligned(x)))

```
// MSVC
```

```
short x1;
```

```
__declspec(align(8)) short x2;
```

```
std::cout << "x1 = " << __alignof(x1) << std::endl;
```

```
std::cout << "x2 = " << __alignof(x2) << std::endl;
```

```
// GCC
```

```
short x3;
```

```
__attribute__((aligned(8))) short x4;
```

```
std::cout << "x3 = " << __alignof__(x3) << std::endl;
```

```
std::cout << "x4 = " << __alignof__(x4) << std::endl;
```

alignof运算符

- 使用alignof代替__alignof和__alignof__

```
auto x1 = alignof(int);  
auto x2 = alignof(void(*)());
```

- 不能代替__alignof和__alignof__的情况

```
int a = 0;  
auto x3 = alignof(a); // C++标准不支持这种用法
```

- 使用decltype获得类型也不准确

alignas说明符

- 接受类型或者常量表达式

```
struct X {  
    char a1;  
    int a2;  
    double a3;  
};  
  
struct X1 {  
    alignas(16) char a1;  
    alignas(double) int a2;  
    double a3;  
};
```

```
struct alignas(16) X2 {  
    char a1;  
    int a2;  
    double a3;  
};  
  
struct alignas(16) X3 {  
    alignas(8) char a1;  
    alignas(double) int a2;  
    double a3;  
};  
  
struct alignas(4) X4 {  
    alignas(8) char a1;  
    alignas(double) int a2;  
    double a3;  
};
```

其他关于对齐字节长度的支持

- `std::alignment_of`
 - 获取类型的对齐字节长度
- `std::aligned_storage`
 - 分配一块指定对齐字节长度和大小的内存
- `std::aligned_union`
 - 获取参数类型中对齐字节长度最严格的（对齐字节数最大）作为分配内存的对齐字节长度

使用new分配指定对齐字节长度的对象

- new运算符接受一个std::align_val_t类型的参数来获得分配对象需要的对齐字节长度

```
void* operator new(std::size_t, std::align_val_t);  
void* operator new[](std::size_t, std::align_val_t);
```



感谢您的观看
欢迎关注