C++ 内存管理库 <new>

C++ 是一种功能强大的编程语言，它提供了丰富的标准库来帮助开发者更高效地编写代码。

在 C++ 中，<new> 是一个非常重要的头文件，它包含了用于动态内存分配的函数和异常类型。

动态内存分配允许程序在运行时请求内存，这在处理不确定大小的数据结构时非常有用。

<new> 头文件定义了以下几个关键组件：

- new 运算符：用于动态分配内存。
- delete 运算符：用于释放动态分配的内存。
- nothrow 运算符：用于在内存分配失败时不抛出异常。
- std::bad_alloc 异常：当内存分配失败时抛出。

语法

使用 new 运算符

new 运算符用于在堆上分配内存。其基本语法如下：

<code class="language-cpp">pointer new (type [, initializer]);</code>

- pointer 是指向分配的内存的指针。
- type 是要分配的对象的类型。
- initializer 是一个可选的初始化表达式。

使用 delete 运算符

delete 运算符用于释放之前使用 new 分配的内存。其基本语法如下：

<code class="language-cpp">delete pointer;</code>

- pointer 是之前使用 new 分配的内存的指针。

实例

动态分配单个对象:

实例

```
#include <iostream>
#include <new> // 包含 <new> 头文件

class MyClass {
public:
    int value;
```

```
    MyClass() : value(0) {}
};

int main() {
    MyClass* myObject = new MyClass; // 分配一个 MyClass 对象
    myObject->value = 10; // 使用点操作符访问成员
    std::cout << "Value: " << myObject->value << std::endl;

    delete myObject; // 释放内存
    return 0;
}
```

输出结果:

```
Value: 10
```

动态分配数组:

实例

```
#include <iostream>
#include <new>

int main() {
    int* myArray = new int[10]; // 分配一个包含10个整数的数组
    for (int i = 0; i < 10; ++i) {
        myArray[i] = i * 2; // 初始化数组
    }

    for (int i = 0; i < 10; ++i) {
        std::cout << "Array[" << i << "]: " << myArray[i] << std::endl;
    }

    delete[] myArray; // 释放数组内存
    return 0;
}
```

输出结果:

```
Array[0]: 0
Array[1]: 2
Array[2]: 4
Array[3]: 6
Array[4]: 8
Array[5]: 10
Array[6]: 12
Array[7]: 14
Array[8]: 16
Array[9]: 18
```

使用 nothrow 避免异常:

实例

```cpp
#include <iostream>
#include <new>

int main() {
    int* myArray = new(std::nothrow) int[10000000]; // 尝试分配一个大数组
    if (!myArray) {
        std::cout << "Memory allocation failed." << std::endl;
    } else {
        std::cout << "Memory allocation succeeded." << std::endl;
        delete[] myArray; // 释放内存
    }
    return 0;
}
```

输出结果：

```
Memory allocation failed. // 或者 Memory allocation succeeded. 取决于系统内存情况
```

异常处理

当使用 new 运算符分配内存失败时，C++ 会抛出一个 std::bad_alloc 异常。

开发者可以通过 try-catch 块来捕获并处理这个异常。

实例

```cpp
#include <iostream>
#include <new>

int main() {
    try {
        int* myArray = new int[10000000]; // 尝试分配一个大数组
        std::cout << "Memory allocation succeeded." << std::endl;
        delete[] myArray; // 释放内存
    } catch (const std::bad_alloc& e) {
        std::cout << "Exception caught: " << e.what() << std::endl;
    }
    return 0;
}
```

输出结果：

```
Exception caught: std::bad_alloc // 如果内存分配失败
```