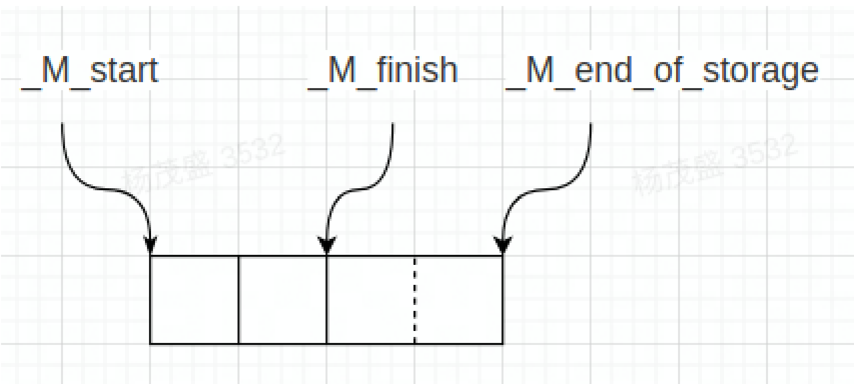
vector扩容机制（基于gcc 9.4.0源码）

vector的数据结构

```
template<typename _Tp, typename _Alloc>
struct _Vector_base
{
  typedef typename __gnu_cxx::__alloc_traits<_Tp_alloc_type>::pointer
    pointer;

  struct _Vector_impl_data
  {
    pointer _M_start;
    pointer _M_finish;
    pointer _M_end_of_storage;

    _Vector_impl_data() _GLIBCXX_NOEXCEPT
    : _M_start(), _M_finish(), _M_end_of_storage()
    {}
  }
}

template<typename _Tp, typename _Alloc = std::allocator<_Tp> >
class vector : protected _Vector_base<_Tp, _Alloc>
{
}
```

vector继承至_Vector_base，具体的数据成员有 _M_start、 _M_finish、 _M_end_of_storage，分别标识着内存地址开始、结束、分配内存末尾；

_M_finish - _M_start 为vector的size； _M_end_of_storage - _M_start为vector的capacity；



vector的resize过程

源码：

```
void resize(size_type __new_size)
{
    if (__new_size > size())
```

```
        _M_default_append(__new_size - size());
    else if (__new_size < size())
        _M_erase_at_end(this->_M_impl._M_start + __new_size);
}


template<typename _Tp, typename _Alloc>
void vector<_Tp, _Alloc>::_M_default_append(size_type __n)
{
    if (__n != 0)
    {
        const size_type __size = size();
        size_type __navail = size_type(this->_M_impl._M_end_of_storage
                                       - this->_M_impl._M_finish); // 还有多少空间
可以使用
        if (__navail >= __n){
            // 还有足够空间存储，不进行扩容
        }
        else{
            const size_type __len =
                _M_check_len(__n, "vector::_M_default_append"); // 求出要扩容至多大
            pointer __new_start(this->_M_allocate(__len)); // 重新申请空间，扩容
            /** 中间代码省略**/
            this->_M_impl._M_start = __new_start;
            this->_M_impl._M_finish = __new_start + __size + __n;
            this->_M_impl._M_end_of_storage = __new_start + __len;
        }
    }
}

size_type _M_check_len(size_type __n, const char* __s) const
{
    if (max_size() - size() < __n)
        __throw_length_error(__N(__s));

    // 下面两行就是扩容的策略，当前大小 + (当前大小与要增加容量的最大值)
    const size_type __len = size() + (std::max)(size(), __n);
    return (__len < size() || __len > max_size()) ? max_size() : __len; // 这个判
断下要扩容的大小是否超过系统最大申请内存大小，如果大于，使用系统最大，如果没有则返回计算的大小
}
```

对照例子进行分析

```
#include <vector>
#include <iostream>
#include <cmath>
#include <algorithm>

using namespace std;
int main() {
    vector<double> a{10.0,11.0};
    std::cout << "0 capacity:" << a.capacity() << std::endl;
    a.push_back(2.0);
```

```
    std::cout << "1 capacity:" << a.capacity() << std::endl;
    a.resize(5);
    std::cout << "2 capacity:" << a.capacity() << std::endl;
    vector<double> b;
    b.resize(5);
    std::cout << "3 capacity:" << b.capacity() << std::endl;
    return 0;
}
```

输出：

> 0 capacity:2
> 1 capacity:4
> 2 capacity:6
> 3 capacity:5

```
vector<double> a{10.0,11.0};
```

创建vector，并以{10.0,11.0}进行构造，此时size为2，capacity为2;

```
a.push_back(2.0);
```

向vector里放置数据，此时需要append数据的个数为1，剩余的空间为0, 那么需要扩容的len = size() + (std::max)(size(), __n); // len = 2 +std::max(2, 1) = 4;

```
a.resize(5);
```

将vector resize到5个元素，此时需要append的数据的个数为2（5-3），剩余的空间为1，那么需要扩容len = size() + (std::max)(size(), __n); // len = 3+std::max(3, 2) = 6;

```
    vector<double> b;
    b.resize(5);
```

创建一个空的vector，它的size和capacity都为0；此时你resize(5), 此时需要append的数据个数为5，剩余空间为0，那么需要扩容len = size() + (std::max)(size(), __n); // len = 0+std::max(0, 5) = 5;