C++ Matrix Library

12111224
贾禹帆

需求

写一个安全，的类cv::Mat_类,同时实现类cv::Mat_<_T>模板类,并尽可能满足STL标准。

简介

exception

使用模板+constexpr在编译期生成各个异常类

```
template <char const* c>
   class RM_Exception final : public std::exception {
   public:
       virtual const char* what() const noexcept override {
           return c;
       }
   };
```

interator

可以适用所有情况进行随机读写的迭代器。考虑到ROI操作和切分通道操作造成的内存不连续，需要足够智能的迭代器来遍历变量，同时为了满足标准库的排序， `for_each` 等函数，需要随机访问和输入输出迭代器

```
    template <typename _T>
    struct Mat_Iterator {
        _T* data;

    private:
        size_t step, jump, colPos, cols, ld;
```

```
    bool forward;

    static void movePtr(long long _n, Mat_Iterator& src);
    static Mat_Iterator createMoved(long long _n, Mat_Iterator& src);

public:
    using difference_type = std::ptrdiff_t;
    using value_type = _T;
    using pointer = _T*;
    using reference = _T&;
    using iterator_category = std::random_access_iterator_tag;
    Mat_Iterator(_T* data, size_t _channel, size_t _jump, size_t _cols,
    Mat_Iterator(const Mat_Iterator& src) = default;
    Mat_Iterator(Mat_Iterator&& src) = default;
    Mat_Iterator& operator=(const Mat_Iterator& rhs) = default;
    Mat_Iterator& operator=(Mat_Iterator&& rhs) = default;
    _T& operator*();
    Mat_Iterator operator+(long long _n);
    Mat_Iterator& operator+=(long long _n);
    Mat_Iterator operator-(long long _n);
    Mat_Iterator& operator-=(long long _n);
    Mat_Iterator& operator++();
    Mat_Iterator operator++(int);
    Mat_Iterator& operator--();
    Mat_Iterator operator--(int);
    _T operator-(const Mat_Iterator& rhs) const;
    _T& operator[](long long _n);
    bool operator==(const Mat_Iterator& rhs) const;
    bool operator!=(const Mat_Iterator& rhs) const;
    bool operator<(const Mat_Iterator& rhs) const;
    bool operator>(const Mat_Iterator& rhs) const;
    bool operator<=(const Mat_Iterator& rhs) const;
    bool operator>=(const Mat_Iterator& rhs) const;
};

template <typename _T>
struct Mat_ConstIterator {
    const _T* data;

private:
    size_t step, jump, colPos, cols, ld;
    bool forward;

    static void movePtr(long long _n, Mat_ConstIterator& src);
    static Mat_ConstIterator createMoved(long long _n, Mat_ConstIterato

public:
    using difference_type = std::ptrdiff_t;
```

```
        using value_type = const _T;
        using pointer = const _T*;
        using reference = const _T&;
        using iterator_category = std::random_access_iterator_tag;
        Mat_ConstIterator(const _T* data, size_t _channel, size_t _jump, si
        Mat_ConstIterator(const Mat_ConstIterator& src) = default;
        Mat_ConstIterator(Mat_ConstIterator&& src) = default;
        Mat_ConstIterator& operator=(const Mat_ConstIterator& rhs) = defaul
        Mat_ConstIterator& operator=(Mat_ConstIterator&& rhs) = default;
        const _T& operator*() const;
        Mat_ConstIterator operator+(long long _n) const;
        Mat_ConstIterator& operator+=(long long _n);
        Mat_ConstIterator operator-(long long _n) const;
        Mat_ConstIterator& operator-=(long long _n);
        Mat_ConstIterator& operator++();
        Mat_ConstIterator operator++(int);
        Mat_ConstIterator& operator--();
        Mat_ConstIterator operator--(int);
        _T operator-(const Mat_ConstIterator& rhs) const;
        const _T& operator[](long long _n);
        bool operator!=(const Mat_ConstIterator& rhs) const;
        bool operator<(const Mat_ConstIterator& rhs) const;
        bool operator>(const Mat_ConstIterator& rhs) const;
        bool operator<=(const Mat_ConstIterator& rhs) const;
        bool operator>=(const Mat_ConstIterator& rhs) const;
    };
```

template Mat_

矩阵模板容器,实现了container要求，使用shared_ptr进行内存管理，实现了基本的构造函数，+,-,*,<<运算符，和切分矩阵，获取行列，通道,软/硬拷贝，begin()，end()，迭代器访问,swap,realloc等操作

操作符因为使用了迭代器实现，若亡值引用坍缩为左值引用,调用 `begin(),rbegin()` 等函数后值销毁，迭代器访问会出现问题，因此对右值引用进行了特化，调用移动构造函数临时保存亡值

> 为减少书写使用了宏，所有宏均在文件末尾 `#undef`

```
    // forward declaration for friend operator
    Temp_declare_Mat_ class Mat_;

    Temp_declare_Mat_ std::ostream& operator<<(std::ostream& o, const Mat_<
```

```
Temp_declare_Mat_ std::ostream& operator<<(std::ostream& o, Mat_<_T>&&

//  Mat_ declaration
template <typename _T>
class Mat_ {
    std::shared_ptr<_T[]> dataBegin;
    _T* data;
    size_t rows, cols, channel;
    size_t realRows, realCols, realChannel;
    size_t rowPos, colPos, channelIdx;
    bool subMat, cpy;

    bool isSameSize(const Mat_<_T>& rhs, bool tran = false) const;

    explicit Mat_(size_t _rowBegin, size_t _colBegin, size_t _rows, siz
                  bool hardCopy);

public:
    // constructors
    Mat_() noexcept;
    Mat_(const curMat_& rhs) = default;
    Mat_(curMat_&& rhs) noexcept = default;
    Mat_(size_t _rows, size_t _cols, size_t _channel = 1);
    Mat_(size_t _rows, size_t _cols, const std::initializer_list<_T>& _
    Mat_(size_t _rows, size_t _cols, const std::vector<_T>& _val);

    Mat_& operator=(const Mat_& rhs) = default;
    Mat_& operator=(Mat_&& rhs) = default;

    // operators
    bool operator==(const Mat_& rhs) const;
    Mat_ operator+(const Mat_& rhs) const;
    Mat_ operator-(const Mat_& rhs) const;
    Mat_ operator*(const Mat_& rhs) const;
    bool operator==(const Mat_&& rhs) const;
    Mat_ operator+(const Mat_&& rhs) const;
    Mat_ operator-(const Mat_&& rhs) const;
    Mat_ operator*(const Mat_&& rhs) const;
    friend std::ostream& operator<< <>(std::ostream& o, const Mat_& src
    friend std::ostream& operator<< <>(std::ostream& o, Mat_&& src);

    // access
    size_t size() const;
    size_t max_size() const;
    bool empty() const;
    size_t rowSize() const;
    size_t colSize() const;
    size_t channelSize() const;
```

```
        size_t realRowSize() const;
        size_t realColSize() const;
        size_t realChannelSize() const;
        _T* ptr(size_t _row = 0);
        _T* ptr(size_t _row, size_t _col);
        const _T* ptr(size_t _row = 0) const;
        const _T* ptr(size_t _row, size_t _col) const;
        bool isContinuous() const;
        bool isSubMatrix() const;
        bool isCopy() const;
        Mat_Iterator<_T> begin();
        Mat_ConstIterator<_T> cbegin() const;
        Mat_Iterator<_T> rbegin();
        Mat_ConstIterator<_T> crbegin() const;
        Mat_Iterator<_T> end();
        Mat_ConstIterator<_T> cend() const;
        Mat_Iterator<_T> rend();
        Mat_ConstIterator<_T> crend() const;

        // others
        Mat_ getSubMat(size_t _rowBegin, size_t _colBegin, size_t _rows, si
        Mat_ getSubMat(size_t _rowBegin, size_t _colBegin, size_t _rows, si
        Mat_ getSplitedChannel(size_t _channelIdx, bool hardCopy) const;
        void splitMat(size_t _rowBegin, size_t _colBegin, size_t _rows, siz
        void splitMat(size_t _rowBegin, size_t _colBegin, size_t _rows, siz
        void splitChannel(size_t _channelIdx);
        std::vector<std::vector<_T>> getRow_v(size_t _row) const;
        std::vector<std::vector<_T>> getCol_v(size_t _col) const;
        Mat_ getRow(size_t _row, bool hardCopy = true);
        Mat_ getCol(size_t _row, bool hardCopy = true);
        Mat_ copy(bool hardCopy = true);
        void copy(Mat_& dest, bool hardCopy = true);
        void swap(Mat_& rhs) noexcept;
        template<typename _T>
        friend void swap(Mat_<_T>& lhs, Mat_<_T>& rhs) noexcept;
        void clear();
        void realloc(size_t _rows, size_t _cols, size_t _channel);
    };
```

文件结构

```
.
├── CMakeLists.txt
```

```
├── modules
│   ├── CMakeLists.txt
│   └── core
│       ├── CMakeLists.txt
│       ├── core.hpp                          包装，类似opencv/core,同时对基础类
│       ├── include
│       │   ├── Mat_helper.hpp                迭代器和异常类声明
│       │   ├── Mat_helper.inl                迭代器和异常类的实现
│       │   ├── Mat.hpp                       Mat_的声明
│       │   └── Mat.inl                       Mat_的函数实现
│       └── src
│           └── PreSpecialized.cpp            强制实例化常见类型的模板，之后直接链
├── project5.pdf
├── refresh.sh
├── report.md
└── test                                      单元测试
    ├── CMakeLists.txt
    ├── include
    │   └── test.hpp
    └── src
        ├── Functions.cpp
        └── test.cpp
```

实现

迭代器

迭代器的所有移动操作本质上是同一操作，因此使用一个私有方法来复用代码,由于有些方法需要返回一个新迭代器,所以需要两份略有不同的函数

```
Temp_declare void Mat_Iterator_temp::movePtr(long long _n, Mat_Iterator_t
      size_t absN = std::abs(_n);
      if(_n < 0 ^ src.forward) {
          long long sum = absN + src.colPos;
          src.data += (src.step + src.jump) * (sum / src.cols * src.ld +
          src.colPos = sum % src.cols;
      } else {
          long long sum = absN - src.colPos;
          size_t crossTimes = (sum > 0 && sum < src.cols) ? 1 : sum / src
          if(sum < 0)
              crossTimes = 0;
          src.data -= (src.step + src.jump) * (crossTimes * src.ld + absN
          src.colPos = (sum > 0 ? src.cols - sum % src.cols : -sum);
```

```
        }
    }

    Temp_declare Mat_Iterator_temp Mat_Iterator_temp::createMoved(long long
        size_t absN = std::abs(_n);
        if(_n < 0 ^ src.forward) {
            long long sum = absN + src.colPos;
            return Mat_Iterator_temp(src.data + (src.step + src.jump) * (su
                                     src.cols, sum % src.jump, src.ld);
        } else {
            long long sum = absN - src.colPos;
            size_t crossTimes = (sum > 0 && sum < src.cols) ? 1 : sum / src
            if(sum < 0)
                crossTimes = 0;
            return Mat_Iterator_temp(src.data - (src.step + src.jump) * (cr
                                     src.cols, (sum > 0 ? src.cols - sum %
        }
    }
```

移动迭代器的操作全部依赖这两个方法，这里就不再赘述

Mat_

构造函数，赋值运算符

由于存储完全依靠智能指针，因此构造函数初始化列表且无需考虑异常导致的内存泄漏，RAII类会在栈解退时自动销毁；使用一个私有构造函数来处理需要roi的情况，同时提供了从常用的vector容器创建的构造函数

```
private:
        explicit Mat_(size_t _rowBegin, size_t _colBegin, size_t _rows, siz
                      bool hardCopy);
public:
        Mat_() noexcept;
        Mat_(const curMat_& rhs) = default;
        Mat_(curMat_&& rhs) noexcept = default;
        Mat_(size_t _rows, size_t _cols, size_t _channel = 1);
        Mat_(size_t _rows, size_t _cols, const std::initializer_list<_T>& _
        Mat_(size_t _rows, size_t _cols, const std::vector<_T>& _val);

        Mat_& operator=(const Mat_& rhs) = default;
        Mat_& operator=(Mat_&& rhs) = default;
```

运算符

对内存不连续的矩阵使用 begin(),end() 等获取迭代器进行访问，因此需要特化亡值的情况然后在内部使用万能引用延长变量声明周期

```
        bool operator==(const Mat_& rhs) const;
        Mat_ operator+(const Mat_& rhs) const;
        Mat_ operator-(const Mat_& rhs) const;
        Mat_ operator*(const Mat_& rhs) const;
        bool operator==(const Mat_&& rhs) const;
        Mat_ operator+(const Mat_&& rhs) const;
        Mat_ operator-(const Mat_&& rhs) const;
        Mat_ operator*(const Mat_&& rhs) const;
        friend std::ostream& operator<< <>(std::ostream& o, const Mat_& src
        friend std::ostream& operator<< <>(std::ostream& o, Mat_&& src);
```

访问

访问矩阵基本属性和数据,没什么好说的

```
    size_t size() const;
        size_t max_size() const;
        bool empty() const;
        size_t rowSize() const;
        size_t colSize() const;
        size_t channelSize() const;
        size_t realRowSize() const;
        size_t realColSize() const;
        size_t realChannelSize() const;
        _T* ptr(size_t _row = 0);
        _T* ptr(size_t _row, size_t _col);
        const _T* ptr(size_t _row = 0) const;
        const _T* ptr(size_t _row, size_t _col) const;
        bool isContinuous() const;
        bool isSubMatrix() const;
        bool isCopy() const;
        Mat_Iterator<_T> begin();
        Mat_ConstIterator<_T> cbegin() const;
        Mat_Iterator<_T> rbegin();
        Mat_ConstIterator<_T> crbegin() const;
        Mat_Iterator<_T> end();
        Mat_ConstIterator<_T> cend() const;
```

```
Mat_Iterator<_T> rend();
Mat_ConstIterator<_T> crend() const;
```

其他

矩阵的roi,切分子矩阵，获取行列，清空容器和重新分配空间的操作,其中切分通道和子矩阵统一使用私有构造函数完成,能够同时处理切分通道，切分矩阵和拷贝的操作

```
Mat_ getSubMat(size_t _rowBegin, size_t _colBegin, size_t _rows, si
Mat_ getSubMat(size_t _rowBegin, size_t _colBegin, size_t _rows, si
Mat_ getSplitedChannel(size_t _channelIdx, bool hardCopy) const;
void splitMat(size_t _rowBegin, size_t _colBegin, size_t _rows, siz
void splitMat(size_t _rowBegin, size_t _colBegin, size_t _rows, siz
void splitChannel(size_t _channelIdx);
std::vector<std::vector<_T>> getRow_v(size_t _row) const;
std::vector<std::vector<_T>> getCol_v(size_t _col) const;
Mat_ getRow(size_t _row, bool hardCopy = true);
Mat_ getCol(size_t _row, bool hardCopy = true);
Mat_ copy(bool hardCopy = true);
void copy(Mat_& dest, bool hardCopy = true);
void swap(Mat_& rhs) noexcept;
friend void swap(Mat_& lhs, Mat_& rhs) noexcept;
void clear();
void realloc(size_t _rows, size_t _cols, size_t _channel);
```

Demo

以下为一段示例

```
#include <core.hpp>
#include <iostream>

int main(){

    //创建一个5*4的四通道矩阵,初始化每个通道为1,2,3,4
    rmat::Mat_<int> test_a(5,4,{1,2,3,4});

    //创建一个5*4的四通道矩阵,初始化各个通道为4,3,2,1
    rmat::Mat_<int> test_b(5,4,std::vector<int>{4,3,2,1});

    //test_c 为单通道5*4矩阵, 每个通道值为6
    auto test_c=a.splitChannel(3)+b.splitChannel(2);
```

```
    //软拷贝一份以test_a(2,2)为起点的3*2大小4通道子矩阵
    auto test_d=test_a.getSubMat(2,2,3,2,false);

    //硬拷贝一份以test_d(0,0)为起点的仅包含2号通达的子矩阵
    auto test_f=test_d.getSubMat(0,0,1,1,2,true);

    //所有硬拷贝矩阵均不为子矩阵，不保留原矩阵信息
    std::cout<<(test_f.isSubMat()?"True":"False");
    //False

    //所有硬拷贝内存均为连续，可以直接指针遍历
    std::cout<<(test_f.isContinuous()?"True":"False");
    //True

    //遍历test_a
    auto channel=test_a.channelSize();
    std::for_each(test_a.begin(),test_a.end(),[channel](auto&& first){
        auto ptr=&first;
        for(int i=0;i<channel;++i)
            *ptr++=i;//or do something other
    });

    //清空 test_c
    test_c.clear();

    //重新分配空间
    test_c.realloc(100,200,2);
}
```

> 以上所有均进行过单元测试