与原始缓冲区接口：Map 类

密集矩阵和数组操作

本页解释了如何使用“原始”C/C++ 数组。这在各种情况下都很有用，特别是在将其他库中的向量和矩阵“导入”到 Eigen 中时。

介绍

有时，您可能有一个预定义的数字数组，您想在 Eigen 中将其用作向量或矩阵。虽然一种选择是制作数据的副本，但最常见的是您可能希望将此内存重新用作 Eigen 类型。幸运的是，使用Map类很容易做到这一点。

映射类型和声明映射变量

Map对象具有由其Eigen等效中定义的类型：

```
Map<Matrix<typename Scalar, int RowsAtCompileTime, int ColsAtCompileTime> >
```

请注意，在这种默认情况下，一个Map只需要一个模板参数。

要构造一个Map变量，您需要另外两条信息：一个指向定义系数数组的内存区域的指针，以及矩阵或向量的所需形状。例如，要定义`float`大小在编译时确定的矩阵，您可以执行以下操作：

```
Map<MatrixXf> mf(pf,rows, columns);
```

其中`pf`是`float *`指向内存数组。一个固定大小的只读整数向量可以声明为

```
Map<const Vector4i> mi(pi);
```

`pi`是`int *`. 在这种情况下，不必将大小传递给构造函数，因为它已由 Matrix/Array 类型指定。

注意Map没有默认构造函数；您*必须*传递一个指针来初始化对象。但是，您可以解决此要求（请参阅更改映射数组）。

Map足够灵活以适应各种不同的数据表示。还有另外两个（可选）模板参数：

```
Map<typename MatrixType,
    int MapOptions,
    typename StrideType>
```

- `MapOptions`指定指针是`Aligned`，还是`Unaligned`。默认为`Unaligned`。
- StrideType 允许您使用 Stride类为内存阵列指定自定义布局。一个示例是指定数据数组以行主格式组织：

例子：

```
int array[8];
for(int i = 0; i < 8; ++i)
    array[i] = i;
cout << "Column-major:\n" << Map<Matrix<int, 2, 4>>(array) << endl;
cout << "Row-major:\n" << Map<Matrix<int, 2, 4, RowMajor>>(array) << endl;
cout << "Row-major using stride:\n" << Map<Matrix<int, 2, 4>, Unaligned,
Stride<1, 4>>(array) << endl;
```

输出：

```
Column-major:
0 2 4 6
1 3 5 7
Row-major:
0 1 2 3
4 5 6 7
Row-major using stride:
0 1 2 3
4 5 6 7
```

然而，Stride 比这更灵活；有关详细信息，请参阅 Map 和 Stride 类的文档。

使用 Map 变量

您可以像使用任何其他 Eigen 类型一样使用Map对象：

例子：

```
typedef Matrix<float, 1, Dynamic> MatrixType;
typedef Map<MatrixType> MapType;
typedef Map<const MatrixType> MapTypeConst;   // a read-only map
const int n_dims = 5;

MatrixType m1(n_dims), m2(n_dims);
m1.setRandom();
m2.setRandom();
float *p = &m2(0);  // get the address storing the data for m2
MapType m2map(p, m2.size());   // m2map shares data with m2
MapTypeConst m2mapconst(p, m2.size());  // a read-only accessor for m2

cout << "m1: " << m1 << endl;
cout << "m2: " << m2 << endl;
cout << "Squared euclidean distance: " << (m1 - m2).squaredNorm() << endl;
cout << "Squared euclidean distance, using map: " << (m1 -
m2map).squaredNorm() << endl;
m2map(3) = 7;   // this will change m2, since they share the same array
cout << "Updated m2: " << m2 << endl;
cout << "m2 coefficient 2, constant accessor: " << m2mapconst(2) << endl;
/* m2mapconst(2) = 5; */   // this yields a compile-time error
```

输出：

```
m1:   0.68 -0.211  0.566  0.597  0.823
m2: -0.605  -0.33  0.536 -0.444  0.108
Squared euclidean distance: 3.26
Squared euclidean distance, using map: 3.26
Updated m2: -0.605  -0.33  0.536      7  0.108
m2 coefficient 2, constant accessor: 0.536
```

所有 Eigen 函数都被编写为接受Map对象，就像其他 Eigen 类型一样。然而，编写自己的职能采取征类型时，这并*不*会自动发生：一个Map类型不相同，其Dense等同。有关详细信息，请参阅编写将特征类型作为参数的函数。

更改映射数组

可以使用 C++“placement new”语法在声明后更改Map对象的数组：

例子：

```
int data[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
Map<RowVectorXi> v(data, 4);
cout << "The mapped vector v is: " << v << "\n";
new (&v) Map<RowVectorXi>(data + 4, 5);
cout << "Now v is: " << v << "\n";
```

输出：

```
The mapped vector v is: 1 2 3 4
Now v is: 5 6 7 8 9
```

尽管表面上看，这不会调用内存分配器，因为语法指定了存储结果的位置。

这个语法使得在不知道映射数组在内存中的位置的情况下声明一个Map对象成为可能：

```
Map<Matrix3f> A(NULL);  // 不要尝试使用这个矩阵！

VectorXf b(n_matrices);

for ( int i = 0; i < n_matrices; i++)
{
    new (&A) Map<Matrix3f>(get_matrix_pointer(i));
    b(i) = A.trace();
}
```