# 扩展MatrixBase（和其他类）

在本节中，我们将看到如何向MatrixBase添加自定义方法。由于继承了所有的表达式和矩阵类型MatrixBase，加入方法MatrixBase使其立即提供给所有表达式！例如，一个典型的用例是使Eigen与另一个 API 兼容。

您当然知道在 C++ 中不可能向现有类添加方法。那怎么可能呢？这里的技巧是在MatrixBase的声明中包含一个由预处理器标记定义的文件`EIGEN_MATRIXBASE_PLUGIN:`

```
class MatrixBase {
  // ...
  #ifdef EIGEN_MATRIXBASE_PLUGIN
  #include EIGEN_MATRIXBASE_PLUGIN
  #endif
};
```

因此，要使用您自己的方法扩展MatrixBase，您只需在包含任何Eigen的头文件之前使用您的方法声明创建一个文件并定义 EIGEN_MATRIXBASE_PLUGIN 。

您可以通过定义类似命名的预处理器符号来扩展 Eigen 中使用的许多其他类。例如，定义`EIGEN_ARRAYBASE_PLUGIN`是否要扩展ArrayBase类。可以通过这种方式扩展的完整类列表和相应的预处理器符号可以在我们的页面预处理器指令中找到。

以下是用于向MatrixBase添加方法的扩展文件示例：
**MatrixBaseAddons.h**

```
inline Scalar at(uint i, uint j) const { return this->operator()(i,j); }
inline Scalar& at(uint i, uint j) { return this->operator()(i,j); }
inline Scalar at(uint i) const { return this->operator[](i); }
inline Scalar& at(uint i) { return this->operator[](i); }

inline RealScalar squaredLength() const { return squaredNorm(); }
inline RealScalar length() const { return norm(); }
inline RealScalar invLength(void) const { return
fast_inv_sqrt(squaredNorm()); }

template<typename OtherDerived>
inline Scalar squaredDistanceTo(const MatrixBase<OtherDerived>& other) const
{ return (derived() - other.derived()).squaredNorm(); }

template<typename OtherDerived>
inline RealScalar distanceTo(const MatrixBase<OtherDerived>& other) const
{ return internal::sqrt(derived().squaredDistanceTo(other)); }

inline void scaleTo(RealScalar l) { RealScalar vl = norm(); if (vl>1e-9)
derived() *= (l/vl); }

inline Transpose<Derived> transposed() {return this->transpose();}
inline const Transpose<Derived> transposed() const {return this-
>transpose();}

inline uint minComponentId(void) const  { int i; this->minCoeff(&i); return
i; }
```

```
inline uint maxComponentId(void) const  { int i; this->maxCoeff(&i); return i; }

template<typename OtherDerived>
void makeFloor(const MatrixBase<OtherDerived>& other) { derived() = derived().cwiseMin(other.derived()); }
template<typename OtherDerived>
void makeCeil(const MatrixBase<OtherDerived>& other) { derived() = derived().cwiseMax(other.derived()); }

const CwiseBinaryOp<internal::scalar_sum_op<Scalar>, const Derived, const ConstantReturnType>
operator+(const Scalar& scalar) const
{ return CwiseBinaryOp<internal::scalar_sum_op<Scalar>, const Derived, const ConstantReturnType>(derived(), Constant(rows(),cols(),scalar)); }

friend const CwiseBinaryOp<internal::scalar_sum_op<Scalar>, const ConstantReturnType, Derived>
operator+(const Scalar& scalar, const MatrixBase<Derived>& mat)
{ return CwiseBinaryOp<internal::scalar_sum_op<Scalar>, const ConstantReturnType, Derived>(Constant(rows(),cols(),scalar), mat.derived());
}
```

然后可以在 config.h 或他的项目的任何先决条件头文件中进行以下声明：

```
#define EIGEN_MATRIXBASE_PLUGIN "MatrixBaseAddons.h"
```