C++11 引入了委托构造的概念，这使得构造函数可以在同一个类中一个构造函数调用另一个构造函数，从而达到简化代码的目的：

```
class Base {
public:
 int value1;
 int value2;
 Base() {
 value1 = 1;
 }
 Base(int value) : Base() { // 委托 Base() 构造函数
 value2 = 2;
  }
};

int main() {
 Base b(2);
 std::cout << b.value1 << std::endl;
 std::cout << b.value2 << std::endl;
}
```

继承构造

在传统 C++ 中，构造函数如果需要继承是需要将参数一一传递的，这将导致效率低下。C++11 利用关键字 using 引入了继承构造函数的概念：

```
class Base {
public:
 int value1;
 int value2;
 Base() {
 value1 = 1;
 }
 Base(int value) : Base() { // 委托 Base() 构造函数
 value2 = 2;
  }
};
class Subclass : public Base {
public:
 using Base::Base; // 继承构造
};
int main() {
 Subclass s(3);
```

```
 std::cout << s.value1 << std::endl;
 std::cout << s.value2 << std::endl;
}
```

显式虚函数重载

在传统 C++中，经常容易发生意外重载虚函数的事情。例如：

```
struct Base {
 virtual void foo();
};
struct SubClass: Base {
 void foo();
};
```

`SubClass::foo` 可能并不是程序员尝试重载虚函数，只是恰好加入了一个具有相同名字的函数。另一个可能的情形是，当基类的虚函数被删除后，子类拥有旧的函数就不再重载该虚拟函数并摇身一变成为了一个普通的类方法，这将造成灾难性的后果。

C++11 引入了 `override` 和 `final` 这两个关键字来防止上述情形的发生。

override

当重载虚函数时，引入 `override` 关键字将显式的告知编译器进行重载，编译器将检查基函数是否存在这样的虚函数，否则将无法通过编译：

```
struct Base {
 virtual void foo(int);
};
struct SubClass: Base {
 virtual void foo(int) override; // 合法
 virtual void foo(float) override; // 非法, 父类没有此虚函数
};
```

final

`final` 则是为了防止类被继续继承以及终止虚函数继续重载引入的。

```
struct Base {
 virtual void foo() final;
};
struct SubClass1 final: Base {
}; // 合法

struct SubClass2 : SubClass1 {
}; // 非法, SubClass 已 final

struct SubClass3: Base {
 void foo(); // 非法, foo 已 final
```

```
};
```

显式禁用默认函数

在传统 C++ 中，如果程序员没有提供，编译器会默认为对象生成默认构造函数、复制构造、赋值算符以及析构函数。另外，C++ 也为所有类定义了诸如 `new delete` 这样的运算符。当程序员有需要时，可以重载这部分函数。

这就引发了一些需求：无法精确控制默认函数的生成行为。例如禁止类的拷贝时，必须将赋值构造函数与赋值算符声明为 `private`。尝试使用这些未定义的函数将导致编译或链接错误，则是一种非常不优雅的方式。

并且，编译器产生的默认构造函数与用户定义的构造函数无法同时存在。若用户定义了任何构造函数，编译器将不再生成默认构造函数，但有时候我们却希望同时拥有这两种构造函数，这就造成了尴尬。

```
C++11 提供了上述需求的解决方案，允许显式的声明采用或拒绝编译器自带的函数。例如：
class Magic {
public:
 Magic() = default; // 显式声明使用编译器生成的构造
 Magic& operator=(const Magic&) = delete; // 显式声明拒绝编译器生成构造
 Magic(int magic_number);
}
```