初始化是一个非常重要的语言特性，最常见的就是对对象进行初始化。在传统 C++ 中，不同的对象有着不同的初始化方法，例如普通数组、POD（plain old data，没有构造、析构和虚函数的类或结构体）类型都可以使用 `{}` 进行初始化，也就是我们所说的初始化列表。而对于类对象的初始化，要么需要通过拷贝构造、要么就需要使用 `()` 进行。这些不同方法都针对各自对象，不能通用。

```
int arr[3] = {1,2,3}; // 列表初始化

class Foo {
private:
 int value;
public:
 Foo(int) {}
};

Foo foo(1); // 普通构造初始化
```

为了解决这个问题，C++11 首先把初始化列表的概念绑定到了类型上，并将其称之为 `std::initializer_list`，允许构造函数或其他函数像参数一样使用初始化列表，这就为类对象的初始化与普通数组和 POD 的初始化方法提供了统一的桥梁，例如：

```
#include <initializer_list>

class Magic {
public:
 Magic(std::initializer_list<int> list) {}
};

Magic magic = {1,2,3,4,5};
std::vector<int> v = {1, 2, 3, 4};
```

这种构造函数被叫做初始化列表构造函数，具有这种构造函数的类型将在初始化时被特殊关照。

初始化列表除了用在对象构造上，还能将其作为普通函数的形参，例如：

```
void func(std::initializer_list<int> list) {
 return;
}

func({1,2,3});
```

其次，C++11 提供了统一的语法来初始化任意的对象，例如：

```
struct A {
 int a;
 float b;
};
struct B {

 B(int _a, float _b): a(_a), b(_b) {}
private:
 int a;
  float b;
 };

 A a {1, 1.1}; // 统一的初始化语法
 B b {2, 2.2};
```