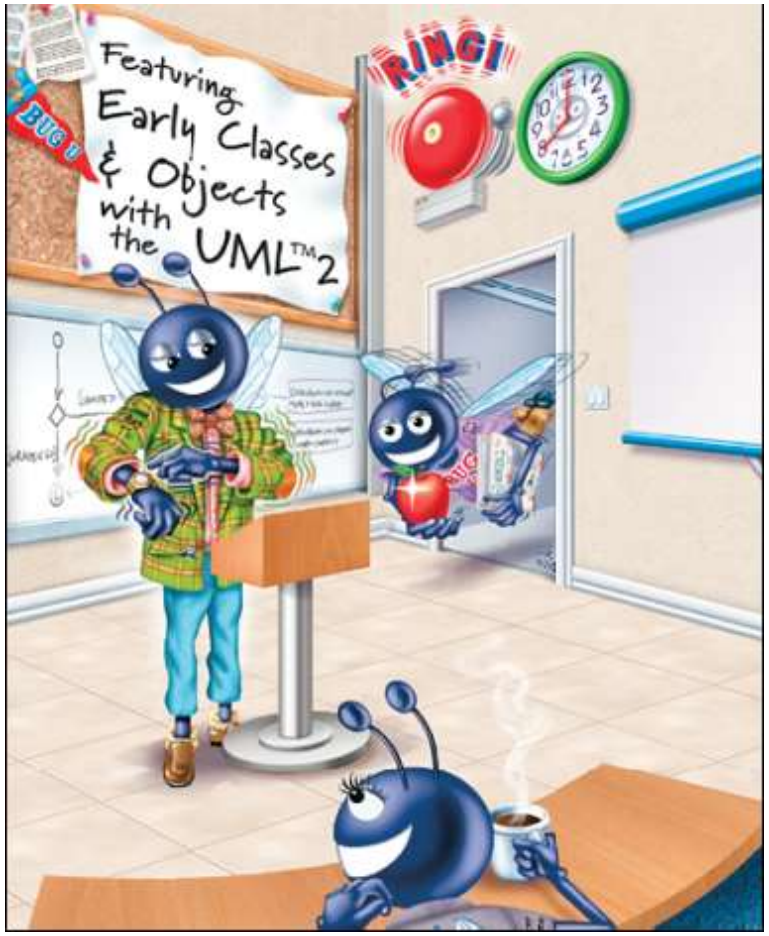
C++程序设计



C++11的一些新特性

- Automatic Type Inference
- Range-Based For Loop
- Initializer Lists
- Lambda Functions
- Scoped/Strongly Typed Enumerations
- Variadic Templates

C++11的一些新特性

- Type-Safe Null Pointer Constant
- Explicit Overrides
- Explicitly Deleted Functions
- Non-Inheritable Classes
- Non-Overridable Member Functions

1. Automatic Type Inference

```
// C++98
for (vector<int>::iterator i = v.begin(), e = v.end(); i != e; ++i) {
    process(*i);
}

// C++11
for (auto i = v.begin(), e = v.end(); i != e; ++i) {
    process(*i);
}
```

2. Range-Based For Loop

```
// C++98
for (vector<int>::iterator i = v.begin(), e = v.end(); i != e; ++i) {
    process(*i);
}

// C++11
for (int& x : v) {
    process(x);
}
```

2. Range-Based For Loop

```
int numbers[] = {1, 2, 3, 4, 5};
 // C++98
for (size_t i = 0, e = sizeof(numbers)/sizeof(numbers[0]); i != e; ++i) {
    numbers[i] *= 2;
}
// C++11
for (int& x : numbers) {
    x *= 2;
}
```

3. Initializer Lists

```
// C++98
std::vector<int> v;
v.push_back(1);
// ...
v.push_back(5);

// C++11
std::vector<int> v = {1, 2, 3, 4, 5};
```

4. Lambda Functions

- 定义和创建匿名函数

```
[ capture list ] (params list) -> return type
{
        function body
}
```

4. Lambda Functions

```
// C++98
bool cmpById(const Person& p1, const Person& p2) {
    return p1.getId() < p2.getId();
}
// ...
std::sort(people.begin(), people.end(), cmpById);
```

4. Lambda Functions

```
// C++11
std::sort(people.begin(), people.end(),
    [ ](const Person& p1, const Person& p2) {
        return p1.getId() < p2.getId();
    }
);
```

4. Lambda Functions

```
// C++11
void printVector(vector<int> v)
{
    for_each(v.begin(), v.end(), [ ](int i)
    {
        std::cout << i << " ";
    });
    cout << endl;
}
```

5. Scoped/Strongly Typed Enumerations

```
// C++98
enum MyEnum {
    VAL1,
    VAL2,
    VAL3
};
MyEnum v = VAL1;        // OK
OtherEnum w = v;        // OK
int i = VAL2;           // OK
```

5. Scoped/Strongly Typed Enumerations

```
// C++11
enum class MyEnum { VAL1, VAL2, VAL3 };


MyEnum v = VAL1;        // ! (compilation error: missing qualification)
MyEnum v = MyEnum::VAL1; // OK
OtherEnum w = v;        // ! (compilation error)
int i = MyEnum::VAL2;    // ! (compilation error: not convertible to int)
```

6. Variadic Templates

```
// C++98
template <typename T1>
void print(const T1& val1) {
    std::cout << val1 << '\n';
}
template <typename T1, typename T2>
void print(const T1& val1, const T2& val2) {
    std::cout << val1;
    print(val2);
}
```

6. Variadic Templates

```
// C++98
template <typename T1, typename T2, typename T3>
void print(const T1& val1, const T2& val2, const T3& val3) {
    std::cout << val1;
    print(val2, val3);
}
 // ...
 print("I am ", 28, " years old."); // I am 28 years old.
```

6. Variadic Templates

```
// C++11
template <typename T>
void print(const T& value) {
    std::cout << value << '\n';
}
template <typename U, typename... T>
void print(const U& head, const T&... tail) {
    std::cout << head;
    print(tail...);
}
 print("I am ", 28, " years old."); // I am 28 years old.
```

7. Type-Safe Null Pointer Constant

```
// C++98
int *i = 0; // or NULL

// C++98
int *func() {
    return 0; // or NULL
}
```

```
// C++11
int *i = nullptr;

// C++11
int *func() {
    return nullptr;
}
```

8. Explicit Overrides

```
class Base {
    virtual void f(int);
};

// C++98
class Derived: public Base {
    virtual void f(double); // Hides Base::f(), does NOT override it.
};
```

8. Explicit Overrides

```
// C++11
class Derived: public Base {
    virtual void f(double) override; // ! (compilation error)
};
```

9. Explicitly Deleted Functions

```
// C++98
class NonCopyable {
private:
    NonCopyable(const NonCopyable&);            // no definition
    NonCopyable& operator=(const NonCopyable&); // no definition
};
```

9. Explicitly Deleted Functions

```
// C++11
class NonCopyable {
public:
    NonCopyable(const NonCopyable&) = delete;
    NonCopyable& operator=(const NonCopyable&) = delete;
}
```

10. Non-Inheritable Classes

```
// C++11
class NonInheritable final {};

class Derived: public NonInheritable {}; // ! (compilation error)
```

11. Non-Overridable Member Functions

```
// C++11
class Base {
    virtual void f();
};
 class Derived {
    virtual void f() final;
};
 class MoreDerived : public Derived {
    virtual void f(); // ! (compilation error)
};
```