

C & C++ 入门

1. 环境

参考 [Google C++ Style Guide](#) 和 [C++ 入门指南](#)，了解 C++ 的编程规范。

2. 基础

了解 C++ 的编译和运行环境。

2.1. 数据类型

了解 C++ 的 `struct` 和 `class` 的定义。

2.1.1. `struct` 和 `class` 的定义

在 C++ 中，`struct` 和 `class` 的定义如下：

在 C++ 中，`struct` 和 `class` 的定义如下：

```
// 结构体定义
struct Coordinate {
    int x;
    int y;
    int z;
};

// 类定义
class Cat {
public:
    void meow();
private:
    ...
}
```

2.1.2.

--	--	--	--	--

```
[ ] [ ] DISALLOW_COPY_AND_ASSIGN [ ] init( ) [ ] ( ),
```

--

--	--	--	--

[illegible]

--	--	--	--	--

[illegible]

11




```
class Person {
public:
    // 显式转换操作符 explicit
    explicit Person(const std::string& name) : _name(name) {} // 显式转换操作符

private:
    std::string _name;
};

DISALLOW_COPY_AND_ASSIGN(Person)

#define DISALLOW_COPY_AND_ASSIGN(ClassName) \
    ClassName(const ClassName&); \
    ClassName& operator=(const ClassName&)

class Foo {
public:
    explicit Foo(int f);
    ~Foo();

private:
    DISALLOW_COPY_AND_ASSIGN(Foo);
};
```

--	--

“ Effective C++, item 04: Make sure that objects are initialized before they’re used More Effective C++, Item 04: Avoid gratuitous default constructors □□□□non-

2.1.3.

```
·[ ]
[ ] private [ ]
```

```
[ ]
[ ]
default constructor[ MyClass::MyClass() [ ]
[ ]/[ ]
```

```
[ ]
[ ]C++ [ ] private [ ]
[ ]
```

```
[ ]
```

More Effective C++, Item 04: Avoid gratuitous default constructors Effective C++, item 06: Explicitly disallow the use of compiler-generated functions you do not want

```
[ ]
[ ]""[ ]
```

```
class String {
public:
    String() : // [ ]
        _str(_S_EMPTY_C_STR) {} // [ ]_S_EMPTY_C_STR[ ]

    ~String() {
        if (_str != _S_EMPTY_C_STR) {
            free(_str);
        }
    }

    const char* c_str() const {
        return _str;
    }
}
```

```
private:
    char* _str;
    static char _S_EMPTY_C_STR[1]; // = ""
};
```

2.1.4. 显式转换操作符

·[] 显式转换操作符 ·[] 显式转换操作符

·

·

implicit constructor MyClass::MyClass(MyArg arg) MyArg MyClass 显式转换操作符

explicit constructor explicit MyClass::MyClass(MyArg arg) 显式转换操作符

[explicit] 显式转换操作符 bug 显式转换操作符

Person(const std::string&) [explicit string] Person 显式转换操作符

```
class Person {
public:
    explicit Person(const std::string& name) : _name(name) {}
private:
    std::string _name;
};
```

string 显式转换操作符 C 显式转换操作符

```
class String {
public:
    String(const char* cs); // 显式转换操作符

private:
    ...
};
```

·

“ More Effective C++, Item 05: Be wary of user-defined conversion functions

2.1.5. 显式转换操作符

Effective C++, item 06: Explicitly disallow the use of compiler-generated functions you do not want Effective C++, item 14: Think carefully about copying behavior in resource-managing classes

2.1.6. 点

·[]

[]

[]

assignment operator=[MyClass& MyClass::operator=(const MyClass&) []

[]

[]

[]

[]

```
class Point {
public:
    Point(int x, int y) : _x(x), _y(y) {}
    Point(const Point& other) :
        _x(other._x), _y(other._y) {}
    Point& operator=(const Point& other) { //
        _x = other._x;
        _y = other._y;
        return *this;
    }
private:
    int _x;
    int _y;
}
```

[]

```
class Person {
public:
    explicit Person(const std::string& name) : _name(name) {}
private:
    std::string _name;
    DISALLOW_COPY_AND_ASSIGN(Person);
}
```



- “ Effective C++, item 06: Explicitly disallow the use of compiler-generated functions you do not want
- Effective C++, item 14: Think carefully about copying behavior in resource-managing classes

2.1.7.

--	--	--	--

RULE010

```
❏ destruct MyClass::~~MyClass() ❏❏
```

```
delete pBase;
```

[illegible]

- Effective C++, item 07: Declare destructors virtual in polymorphic base classes
- Effective C++, item 08: Prevent exceptions from leaving destructors
- Effective C++, item 11: Prevent exceptions from leaving destructors

2.1.8.

--	--

.[] [] " [] " [] .[] [] [RULE014] []

[illegible]


```
void attach(void * p_object, DestructorFunc destructor); // C buffer
... // attach

template <typename T>
T& create(); // 
... // create
... // create_raw(),clone_cstring(),reset()
private:
// private
// 
struct ObjectInfo; // 
...
// 
void push_info(); // push_info()pop_info()
void pop_info(); // private
...
// 
ObjectInfo *_p_object_info_list;
...
DISALLOW_COPY_AND_ASSIGN(ResourcePool); // 
};
```

[illegible]

```
operator ==( const Dict&, const Dict& )
```

[illegible]

Effective C++, Item 23, Prefer non-member non-friend functions to member functions

2.1.13. 静态成员函数

在 C++ 中，静态成员函数（static member function）是指属于类（class）但不属于任何对象（object）的函数。它们通常用于实现与类相关的操作，而不需要创建类的对象。静态成员函数的定义通常放在头文件（header file）中，而静态成员函数的实现则放在源文件（source file）中。静态成员函数的调用方式与普通成员函数类似，但需要在函数名前加上类名（class name）作为前缀。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。

在 C++ 中，静态成员函数（static member function）是指属于类（class）但不属于任何对象（object）的函数。它们通常用于实现与类相关的操作，而不需要创建类的对象。静态成员函数的定义通常放在头文件（header file）中，而静态成员函数的实现则放在源文件（source file）中。静态成员函数的调用方式与普通成员函数类似，但需要在函数名前加上类名（class name）作为前缀。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。

在 C++ 中，静态成员函数（static member function）是指属于类（class）但不属于任何对象（object）的函数。它们通常用于实现与类相关的操作，而不需要创建类的对象。静态成员函数的定义通常放在头文件（header file）中，而静态成员函数的实现则放在源文件（source file）中。静态成员函数的调用方式与普通成员函数类似，但需要在函数名前加上类名（class name）作为前缀。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。

在 C++ 中，静态成员函数（static member function）是指属于类（class）但不属于任何对象（object）的函数。它们通常用于实现与类相关的操作，而不需要创建类的对象。静态成员函数的定义通常放在头文件（header file）中，而静态成员函数的实现则放在源文件（source file）中。静态成员函数的调用方式与普通成员函数类似，但需要在函数名前加上类名（class name）作为前缀。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。静态成员函数的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员函数。

2.1.14. 静态成员变量

在 C++ 中，静态成员变量（static member variable）是指属于类（class）但不属于任何对象（object）的变量。它们通常用于存储与类相关的状态信息，而不需要创建类的对象。静态成员变量的定义通常放在源文件（source file）中，而静态成员变量的声明则放在头文件（header file）中。静态成员变量的调用方式与普通成员变量类似，但需要在变量名前加上类名（class name）作为前缀。静态成员变量的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员变量。静态成员变量的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员变量。静态成员变量的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员变量。

在 C++ 中，静态成员变量（static member variable）是指属于类（class）但不属于任何对象（object）的变量。它们通常用于存储与类相关的状态信息，而不需要创建类的对象。静态成员变量的定义通常放在源文件（source file）中，而静态成员变量的声明则放在头文件（header file）中。静态成员变量的调用方式与普通成员变量类似，但需要在变量名前加上类名（class name）作为前缀。静态成员变量的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员变量。静态成员变量的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员变量。静态成员变量的实现与类（class）的定义无关，因此可以在类（class）定义之前使用静态成员变量。

2.2. C++ 中的命名空间

2.2.1. 命名空间

在 C++ 中，命名空间（namespace）是一种用于组织代码的结构。它允许我们将具有相同名称的变量、函数、类等组织在一起，以避免命名冲突。命名空间的定义通常放在头文件（header file）中，而命名空间的实现则放在源文件（source file）中。命名空间的调用方式与普通变量、函数、类等类似，但需要在名称前加上命名空间名（namespace name）作为前缀。命名空间的实现与类（class）的定义无关，因此可以在类（class）定义之前使用命名空间。命名空间的实现与类（class）的定义无关，因此可以在类（class）定义之前使用命名空间。命名空间的实现与类（class）的定义无关，因此可以在类（class）定义之前使用命名空间。

在 C++ 中，命名空间（namespace）是一种用于组织代码的结构。它允许我们将具有相同名称的变量、函数、类等组织在一起，以避免命名冲突。命名空间的定义通常放在头文件（header file）中，而命名空间的实现则放在源文件（source file）中。命名空间的调用方式与普通变量、函数、类等类似，但需要在名称前加上命名空间名（namespace name）作为前缀。命名空间的实现与类（class）的定义无关，因此可以在类（class）定义之前使用命名空间。命名空间的实现与类（class）的定义无关，因此可以在类（class）定义之前使用命名空间。命名空间的实现与类（class）的定义无关，因此可以在类（class）定义之前使用命名空间。

在 C++ 中，命名空间（namespace）是一种用于组织代码的结构。它允许我们将具有相同名称的变量、函数、类等组织在一起，以避免命名冲突。命名空间的定义通常放在头文件（header file）中，而命名空间的实现则放在源文件（source file）中。命名空间的调用方式与普通变量、函数、类等类似，但需要在名称前加上命名空间名（namespace name）作为前缀。命名空间的实现与类（class）的定义无关，因此可以在类（class）定义之前使用命名空间。命名空间的实现与类（class）的定义无关，因此可以在类（class）定义之前使用命名空间。命名空间的实现与类（class）的定义无关，因此可以在类（class）定义之前使用命名空间。

```
// 命名空间
namespace saick {
...
// 命名空间
namespace my_module {
...
} // namespace my_module
```

```
} // namespace saick
```

~~~~~BadCase~~~~~:

```
// a.h
#include <string>
using namespace std; // ~~~~~using namespace

class A {
public:
    string name() const;
}

// a.cpp
#include "bsl/bsl_string.h"
#include "a.h"

using namespace bsl; // ~.cpp~~~~using namespace

int main() {
    string str; // ~~~~~bsl::string[]std::string

    A a;
    cout << a.name() << endl; // ~~~~std~~~~~
}
```

~~~~~

```
// a.h
#include <string>

class A {
public:
    std::string name() const; // []std::string[]string~~~~~
};

// a.cpp
#include "bsl/bsl_string.h"
#include "a.h"
```

```
using std::cout; // using class
using std::endl;

int main() {
    bsl::string str; //

    A a;
    cout << a.name() << endl;
}
```

2.2.2.

·[] [RULE025] ·[] (10)

10

10

```
class Person {
public:
    std::string get_name() const { //
        return _name;
    }
    int get_age() const {
        return _age;
    }
private:
    std::string _name;
    int _age;
};

inline void greet(const Person& person) { //inline
    std::cout << "Hello, " << person.get_name() << "!" << std::endl;
}
```

10

“ Effective C++, item 30: understand the ins and outs of inlining

2.2.3.

·[] [] ·[] C++ []

C++ [static_cast dynamic_cast const_cast reinterpret_cast]

[] _cast [] C++ []

[static_cast void * []

```
void foo(const void* network_buf) {  
    const MyBuf * buf = static_cast<const MyBuf*>(network_buf); //  
}
```

[const_cast []

```
// [ ]void old_copy(void *dest, const void *src, size_t n);  
// [ ]  
void old_copy(void* dest, void* src, size_t n);  
  
// [ ]  
void new_copy(void* dest, const void* src, size_t n) {  
    old_copy(dest, const_cast<void *>(src), n); // [ ]const [ ]const [ ]  
}
```

[]

“ Effective C++, item 27: Minimize casting More Effective C++, item 02: Prefer C++ style casts

2.2.8. ++/-- []

·[] [], [] ([]).

[++/-- []

[]

“ More Effective C++, item 06: Distinguish between prefix and postfix forms of increment and decrement operators

2.3. C/C++ 比較

2.3.1. 比較演算子

・[C] 比較演算子は、[C] 比較演算子・[C] 比較演算子==[C]・[C] [RULE033] 比較演算子

if (my_value == 0) { ... }

if

```
// 比較演算子
if (is_valid) {
    ...
}
if (!is_finished) {
    ...
}
// 比較演算子
if (my_value == 0) {
    ...
}
// 比較演算子
if (p_value == NULL) {
    ...
}
// 比較演算子!=0
const double EPSILON = 1e-9; // 比較演算子1e-9
if (fabs(my_value) < EPSILON) {
    ...
}
```

2.3.2. NULL, nullptr, 0

・[C] 0 0.0 NULL '\0' 比較演算子 nullptr 比較演算子++11 比較演算子

if (my_value == 0) { ... }

if

```
int i = 0;
double i = 0.0;
```



```
void* p = NULL;
char ch = '\0';
```

2.3.3. sizeof

·[] sizeof() [] ·[] sizeof() sizeof() [] 32/64

```
int g_my_id;

void func() {
    size_t size_of_my_id = sizeof(g_my_id); // OK
}

int g_my_id;

void func() {
    size_t size_of_my_id = sizeof(int);      // NO! g_my_id long long size_t
}
```

2.3.4. typedef

·[] type tag typedef [] ·[] struct typedef

typedef typedef

YES:

```
class MyContainer {
public:
    typedef char value_type; // type tag

    void my_method() {
        typedef std::vector<int>::iterator iterator_type; //
        ...
    }
}
```

NO:

```
typedef tagMyStruct {
    ...
}
```

```
} MyStruct; // C++에서 C 스타일
```

2.3.5. goto

·[] [RULE037]] goto

goto

2.3.6.

·[DISALLOW_COPY_AND_ASSIGN CFATAL_LOG etc.) . [c ## W # break continue return

inline

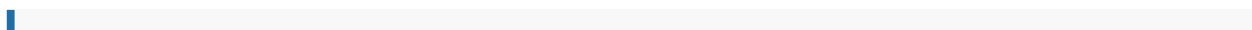
inline

```
inline int abs(int i) {  
    return (i >= 0 ? i : -i);  
}  
  
//  
  
template <typename T>  
inline T abs( T value ) {  
    return (value >= 0 ? value : -value);  
}
```

```
enum Color {  
    RED,  
    BLUE,  
    GREEN,  
    COLOR_COUNT  
};
```

const

```
const size_t BUFFER_SIZE = 1048576;
```



2.3.7. 友元函数

·[友元函数] 友元函数

友元函数

友元函数

```
class Iterator; //友元函数
class Container {
    Iterator* begin();
};
```

友元函数

Effective C++, item 31: Minimize compilation dependencies between files

2.3.8. 友元函数/友元类

·[友元函数] 友元函数 single main() 友元函数 class

友元函数 友元函数, 友元函数 C++ 友元函数, 友元函数, [

2.3.9. 友元函数

·[友元函数] [RULE050] 友元函数/友元函数 ·[友元函数] [RULE051] 友元函数 enum 友元函数 const 友元函数 define 友元函数

友元函数 define 友元函数 strlen("filename") 友元函数 file_name 友元函数 define 友元函数 友元函数

友元函数

```
enum Color {
    RED,
    BLUE,
    GREEN,
    COLOR_COUNT
};
```

友元函数 const 友元函数

```
const size_t BUFFER_SIZE = 1048576;
```

--	--

“ Effective C++, item 02: Prefer consts, enums, and inlines to #define

2.3.10. const

·[] []const [] []const []const []/[]const []

```

const [bug, const_cast] = C.C

```

```
class MyClass {
public:
    void my_const_method(const OtherClass& arg) const;
};

const MyClass my_obj; // const
const MyClass* p1;    //
const MyClass* const p2 = &my_obj; //
```

```
const MyClass* const p2 = &my_obj; // [ ][ ][ ][ ][ ][ ]
```

--	--

Effective C++, item 03: Use const whenever possible

2.3.11.

--	--	--	--	--	--

.[] [] [] [] [] [] [] [] .[] [] [] [] 4KB [] [] [] []

□□ □□□□(variable-length array):□□□□□□□□, □□ GCC □□□□

crash crash

3. ☐ ☐ ☒ ☐

3.1. | | | | | | |--|--|--|--|--| | | | | | | |--|--|--|--|--|

3.1.1. `#include`

·[1] `foo.cpp`(`foo.h`), C(`foo.h`)C++(`foo.h`)

`foo.cpp` `foo.h`; `foo.h` `foo.h`;

`foo.h`, `some-project/foo/internal/fooserver.cc` `foo.h`:

```
#include "foo/public/fooserver.h" // foo.h
#include <sys/types.h>
#include <unistd.h>
#include <hash_map>
#include <vector>
#include "base/basic_types.h"
#include "base/commandlineflags.h"
#include "foo/public/bar.h"
```

3.1.2. `#include <>` `#include ""`

·[1] `#include <>` `#include ""`

3.1.3. `#include`

·[1] `#include`

`#include` `foo.h`

`foo.h`

```
#include "bsl/ResourcePool.h"
#include "bsl/var/IVar.h"
#include "mylib/MyClass.h"
```

3.1.4. `#include guards`

·[1] [RULE060] `#include guards` ·[1] [RULE061] `include <SVNPATH>_<FILE>_H` `SVNPATH` `trunk`, `branch-xxx`;

`include guards` `#ifndef` `#define` `#endif` `foo.h`

`include guards` `svn` `include guards` `foo.h`; `foo.h`

// `foo.h`, `svn` `https://svn.saick.net/ps/se/trunk/ac/strategy/ StrategyQueue/default/time_open_cluster_data.h` `foo.h`:

```
#ifndef PS_SE_AC_STRATEGY_STRATEGYQUEUE_DEFAULT_TIME_OPEN_CLUSTER_DATA_H
#define PS_SE_AC_STRATEGY_STRATEGYQUEUE_DEFAULT_TIME_OPEN_CLUSTER_DATA_H
...
#endif // PS_SE_AC_STRATEGY_STRATEGYQUEUE_DEFAULT_TIME_OPEN_CLUSTER_DATA_H
```

```
// Copyright 2017 Saick net. All Rights Reserved.  
// Author: LastName FirstName (eric@saick.net)  
//  
// <[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]>  
[ ][ ][ ][ ]Author[ ][ ][ ][ ][ ][ ][ ][ ]  
  
// Copyright 2017 Saick net. All Rights Reserved.  
// Author: LastName FirstName (eric@saick.net)  
//      LastName2 FirstName2 (x@saick.net)  
//  
// <[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]>
```

--	--

```
// mylib/AwesomeClass.h
namespace mylib {
class AwesomeClass {
    void foo() { //[]
        ...
    }
    void bar();
}; // MyClass

} //namespace mylib

// mylib/AwesomeClass.cpp
namespace mylib {
void AwesomeClass::bar() {
    while (...) {
        if (...) {
            ...
        } else {
            ...
        }
    } // while loop
}

} // namespace mylib
```

3.2.3. []

·[[]] [RULE068] [] []·[[]] []5[] []·[[]] [RULE069] []4[]

[] terminal [] [] 4[]

3.2.3.1. []

·[[]] [RULE070] []

[] namespace [](100[],[]); [] namespace [];

[] []

```
namespace bsl {
namespace var {
class IVar {
```

```
};

} // namespace var
} // namespace bsl
```

3.2.3.2. 命名空间

·[命名空间] [RULE071] 命名空间

3.2.3.3. 访问控制

·[命名空间] [RULE072] public/protected/private命名空间

命名空间

```
class MyClass {
public:
    void foo();
protected:
    void bar();
private:
    void meow();
    int _haha;
};
```

3.2.3.4. 初始化

·[命名空间] 初始化, 初始化, 初始化: 初始化. 命名空间 4初始化8命名空间

```
// 初始化
MyClass::MyClass(int var) : _some_var(var), _some_other_var(var + 1) {}

// 初始化 8命名空间
// the first initializer line:
MyClass::MyClass(int var) :
    _some_var(var),          // 8 space indent
    _some_other_var(var + 1) { // lined up
do_something();
...
}
```

3.2.3.5. 命名空间


```
    return "you come from solar system";
}
```

--	--

“1TBS”

3.2.3.8. switch

```
[ ] switch case [ ] break [ return [ ] [ ] [ ] [ ] [ ] default { //pass } //do nothing switch switch case [ ] [ ]
```



```
switch [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
```

[case [break [return [case]]]] default

--	--	--	--	--	--	--	--

```
switch (cur_char) {
case '\0':
case '\': {
    return "it's a string!";
}
default:
    printf("not found yet");
    break;
}
```

--	--	--	--	--	--	--	--

```
switch (cur_char) {
case '\0':
    printf("wow!"); // break return
case '\':
    return "it's a string!";
} // default
```

3.2.3.9.

--	--	--	--

[] [RULE083] do-while while E084

--	--	--	--

```
while (iter.next()) {  
    // pass  
}
```

--	--	--	--	--	--	--

```
for (size_t i = 0; i < N; ++i) {  
    sum += arr[i];  
}  
  
// do-while  
  
do {  
    ...  
} while (++it != end); // □□  
  
do {  
    ...  
}  
  
while (++it != end); // □□□□□□□□□□□□□□□□
```

--	--

“1TBS”

3.2.4.

--	--	--	--	--

[REDACTED]

[REDACTED] [RULE085 if/switch/while/for/catch [REDACTED] for [REDACTED]] [REDACTED] [RULE086 [REDACTED] [RULE087] [REDACTED]

[REDACTED] [RULE090] [REDACTED] a.b, a->b, a.*b, a->*b, a::b [REDACTED] [RULE091[RULE092]] [REDACTED]

--	--

```

if (is_good && is_powerful) // 1
if(is_good && is_powerful) // 1 if(111111)
if ( is_good && is_powerful ) // 1, 11111111
switch (type) // 1
while (condition) // 1
catch (std::exception& ex) // 1
for (int i = 0; i < 10; ++i) // 1
for (int i = 0;i<10;+i) // 1, 111111

```



```
UB_LOG_WARNING(
    "An exception[%s] was thrown from[%s:%d:%s] " // 异常信息, 文件
    "with a message[%s]! stack_trace:%s%s",
    e.name(), e.file(), int(e.line()), e.function(), // 异常信息
    e.what(), e.get_line_delimiter(), e.stack()
);
```

3.2.6. 异常

·[文件] 异常信息100字节 ·[文件] .h 异常信息, 异常信息, 文件 .hpp 异常信息

异常信息

3.2.7. 异常

·[文件] 异常return异常信息

异常 return 异常信息 异常()

异常

```
return (ret_val);
```

3.2.8. 异常

·[文件] 异常信息 ·[文件] 异常 const 异常信息 const 异常

异常 异常信息 异常信息 异常 异常

异常

```
/*
C/C++ 异常信息, 异常, 异常/异常.
异常信息 ``const`` 异常. 异常+异常const异常
异常(异常), 异常 ``const`` 异常
异常, 异常信息: 异常, 异常. 异常/异常 (异常/异常) 异常, 异常信息, 异常.
*/
// 异常const
void foo(const std::string& input1, const MyClass& input2);

// 异常
void foo(int input1, float input2);
```

```

// [ ]const[ ]
void foo(const MyClass* input1);

// [ ]
void foo(int* output1, MyClass* output2);

// [ ]std::string*[ ]char*[ ]char*[ ]buffer[ ]
void foo(std::string* out);

// [ ]
void foo(const InClass& input1, OutClass* out1);

// [ ]
void foo(const InClass& input1, // [ ]const[ ]
        int input2, // [ ]POD[ ]
        OutClass* outpu1); // [ ]

// [ ]
void foo(const InClass& input1,
        int* inout, // [ ]
        OutClass* output1)

[ ]:
// Always have named parameters in interfaces.
class Shape {
public:
    virtual void rotate(double radians) = 0;
}

// Always have named parameters in the declaration.
class Circle : public Shape {
public:
    virtual void rotate(double radians);
}

// Comment out unused named parameters in definitions.
void Circle::rotate(double /*radians*/) {}

```

3.3. []/[]

3.3.1.

--	--	--	--	--	--

`.[] creat [usr] []`

VSriptEvaluator vse _evaluator ~~WHEN BACK~~ extern "C"

--	--

```
// ██████████ ████████████ ████████████████████████;
// ████████

int num_errors; // Good.

int num_completed_connections; // Good.

// ██████

int n; // Bad - ██

int nerr; // Bad - ██████

int n_comp_conns; // Bad - ambiguous abbreviation.

// █: ██████████;
// █████:

int num_dns_connections; // █████ "DNS" █████
int price_count_reader; // OK, price count. Makes sense.

// █████

int wgc_connections; // wgc █?

int pc_reader; // pc█████!

// ████████████████

int error_count; // Good!

int error cnt; // Bad! ████████, ████████;
```

3.3.2.

```
[.] [.] .h [.] .hpp [C++ ] .cpp [C] [RULE100] [<_test.h>] :  
[> <> lib mylib.h,mylib utils.h,mylib api.h
```

3.3.3.

[illegible]

using namespace

3.3.4.

[RULE103] getter: my_member_variable(), setter: set_my_member_variable(), mutable: mutable_my_member_variable()

accessor Google protobuf

```
class Person {
public:
    ...
    const IdCard& id_card() const { return _id_card; }          // getter
    IdCard* mutable_id_card() { return &_id_card; }
    void set_id_card(const IdCard& id_card) { _id_card = id_card; } // setter

private:
    IdCard _id_card;
    int _age;
    int get_age() const { return _age; }
};
```

3.3.5.

[RULE105] [RULE106]

enum struct class typedef template

```
enum Color {
    RED,
    BLUE,
    GREEN,
    COLOR_COUNT
};
struct Point {
    int x;
    int y;
};
```


3.3.6.

·[] [RULE107] g ·[] [RULE108]

3.3.7.

·[] [RULE109] ·[]

cur_user_index i,j,k i,j,k

3.3.8.

·[] [RULE110] s_ ·[] [RULE_s_]

[s_] s static(_ _s_);

3.3.9. ✓

·[] [RULE112] MyClass::_my_attr [RULE113] MyStruct::my_attr ,

3.3.10.

·[] const

3.3.11. ✓

·[] MY_LIB_MY_MACRO_THAT_SCARES_SMALL_CHILDREN

3.3.12.

·[] <stdint.h> int64_t, int32_t

unsigned long long

3.4.

3.4.1.

·[] : , UTF-8 ·[] ·[]

3.4.2.

编译选项: `g++ 1_10_0.cpp -std=c++11 -g -DTESTING -DFOO -DFOO_ENVIRONMENT`

编译选项:

```
class FooEnvironment : public testing::Environment {
public:
    virtual void SetUp() {
        std::cout << "Foo FooEnvironment SetUP" << std::endl;
    }
    virtual void TearDown() {
        std::cout << "Foo FooEnvironment TearDown" << std::endl;
    }
};

// Example:
// template
class FooTest : public testing::Test {
public:
    ...
    typedef std::list List;
    static T shared_;
    T value_;
};
```

4.2. 编译选项

编译选项: `g++ 1_10_0.cpp -std=c++11 -g -DTESTING -DFOO -DFOO_ENVIRONMENT`

4.3. Windows 编译

Windows 编译 C++ 编译选项: `g++ 1_10_0.cpp -std=c++11 -g -DTESTING -DFOO -DFOO_ENVIRONMENT`

4.3.1. 编译选项

编译选项: `g++ 1_10_0.cpp -std=c++11 -g -DTESTING -DFOO -DFOO_ENVIRONMENT`

编译选项: `g++ 1_10_0.cpp -std=c++11 -g -DTESTING -DFOO -DFOO_ENVIRONMENT`

4.3.2. 编译选项

4.3.2.1. 花括号

·[C] [WCPP006] { } 花括号必须成对出现 ·[C] [WCPP005] 花括号必须成对出现

花括号

```
if (i == j)
{
    //花括号必须成对出现
    //i for do while switch case 花括号必须成对出现
    i++;
}
```

·[C] 在 int x = 3 & (4 < 32) 花括号, 花括号

4.3.3. 花括号WINAPI VS IDE 花括号

·[C] 在 WINAPI 花括号 DWORD HANDLE 在 C++ 花括号 include guard 花括号 #pragma once

花括号

```
#ifndef ABC_H
#define ABC_H
#endif
```

4.4. 花括号"花括号"花括号

·[C] 花括号"花括号"花括号 花括号

Revision #5

Created 17 August 2024 14:48:26 by Danny

Updated 15 September 2024 12:13:05 by Danny