
学校的理想装备
电子图书·学校专集
校园网上的最佳资源
Visual j++6.0 附录
BOOK
内部资料 非卖品

返回总目录

目　　录

附录 A　错误和警告

编译器错误 J0001

INTERNAL COMPILER ERROR（内部编译器错误）

编译器在检测出错误之后不能恢复。请参阅 Help（帮助）菜单中可用的技术支持帮助文件，查找 Microsoft Knowledge Base 中关于该问题的最新信息。还有，试着简化编译器报告出现该错误的代码，并重新编译。

编译器错误 J0002

Out of memory（内存溢出）

编译器试图在处理过程中分配一些附加的内存，但是，却做不到这一点。检查系统交换文件的位置、大小和有效性。同时，确保在交换文件所在的驱动器上有足够的空间可以使用。

编译器错误 J0004

Cannot open class file 'filename' for reading（不能打开要读的类文件‘filename’）

编译器不能打开要读入的程序源文件。通常，在其他程序已经独占锁定该源文件时，会发生此错误。关闭其他可能会访问到该源文件的其他进程，并重新编译。

编译器错误J0005

Cannot open class file 'filename' for writing（不能打开要写入的类文件‘filename’）

编译器在生成输出.class文件时失败。编译器没有写入或创建该文件的权限时，通常会发生该错误。要确认该文件没有被设置为只读属性，并且，没有被其他进程使用。在用户正在运行或调试包含指定.class文件的程序时，也会出现此错误。关闭所有的程序实例并重新编译。

编译器错误J0006

Cannot read class file 'filename'（不能读类文件‘filename’）

编译器读取某个指定的类文件时失败。当编译器读取指定的存储设备遇到错误时，或编译器不能得到读取该文件的权限时，通常发生该错误。要确定该文件没有被其他进程使用。另外，使用磁盘扫描程序来确定所要使用的存储设备是有效的。

编译器错误J0007

Cannot write class file 'filename'（类文件‘filename’不能写）

编译器在试图将缓冲区的内容写入到指定的类文件时失败。当目标存储设备上的空间不够时，通常会发生此错误。从该存储设备中释放一些可用空间，并重新编译。

编译器错误J0010

Syntax error（句法错误）

编译器不能确定在源程序中表达式或语句的意义。当错误信息中指定行的句法无效时，通常发生该错误。通常，这类错误还带有更多的说明信息。修改所有相关的错误，并重新编译。
下面举例说明该错误：

```
Public class Simple {

    public void method1() {
        int i=: // error: missing assignment value
    }
}
```

编译器错误J0011

Expected ':'（缺少‘：’）

在case标签后，或在条件表达式中，应当有一个冒号，以便使用三重运算符。在偶然丢掉了冒号时，通常发生该错误。通常情况下，该错误是由编译器报告错误所在行的前一行引起的。确保需要冒号的行是正确的，并重新编译。

下面举例说明该错误：

```
public class Simple {

    private int i;

    private static int x = i;

     public void method1(int arg1) {

        switch (arg1) {

            case 1 // error: ': ' omitted after 'case 1'

           }

           i = (arg1 < x) ? arg1 x; // error: ': ' omitted after 'arg1'

    }

}
```

编译器错误J0012

Expected ';'（缺少‘；’）

编译器期望在错误信息所指出的位置找到一个分号。在语句的结尾偶然遗漏了分号时，通常会发生此错误。当条件表达式的语法不正确时，也会发生此类错误。该错误经常是由编译器报告错误所在行的前一行引起的。确保分号的正确使用，并重新编译。
下面举例说明该错误：

```
public class Simple {

    private static int x = 10 // error : '; ' omitted

     public void method1(int arg1) {

        for (int i = 1; i < x i++) {
            // error: ; omitted
        }
    }
}
```

编译器错误J0013

Expected '('（缺少‘（’）

编译器期望在错误信息所指出的位置找到一个左括号。在下列任何一种情况下，丢失左括号时，通常会发生此错误：

- 类型初始化
- catch 语句
- 带有括号的表达式
- while 循环
- for 循环
- if/else 语句

下面举例说明该错误：

```
public class Simple {

    private int i;

    public void method1(int arg1, int arg2);

      if arg1 < arg2) // error: '(' omitted
        i = arg1;
      else
        i = arg2;

    }
}
```

编译器错误 J0014

Expected ')'（缺少‘）’）

编译器期望在错误信息所指出的位置找到一个右括号。在下列任何一种情况下丢失右括号时，通常会发生此错误：

- 类型初始化
- 类型转换
- catch 语句
- 带有括号的表达式
- while 循环
- for 循环
- if/else 语句

下面举例说明该错误：

```
public class Simple {

    private int i;

    public void method1(int arg1, int arg2) {

        i = (arg1 < arg2 ? arg1 : arg2;
      // error: ) omitted
    }
```

```
}
```

编译器错误J0015

Expect ']'（缺少‘]’）

编译器期望在错误信息所指出的位置找到一个右方括号。当从数组声明中意外地丢失右方括号时，通常会发生此错误。该错误经常是由编译器报告错误所在行的前一行引起的。确保所有的方括号匹配，并重新编译。
下面举例说明该错误：

```
public class Simple {

    private int x[ = new int [500]; //error: ']' omitted
}
```

编译器错误J0016

Expected '{'（缺少‘{’）

编译器期望在错误信息所指出的位置找到一个左大括号。当在类声明或方法代码块的开始位置丢失左大括号时，通常出现该错误。该错误经常是由编译器报告错误所在行的前一行引起的。确保所有的大括号是匹配的，并重新编译。
下面举例说明该错误：

```
public class Simple // error :'{' omitted

    public void method1() {
       //do something meaningful
    }
}
```

编译器错误J0017

Expected '}'（缺少‘}’）

编译器期望在错误信息所指出的位置找到一个右大括号。当找不到类声明或方法的结束大括号时，通常发生此错误。该错误经常是由编译器报告错误所在行的前一行引起的。确保所有的大括号匹配，并重新编译。下面举例说明该错误：

```
public class Simple {
{
    int x , y ;
    // error : no matching brace found for class
```

编译器错误J0018

Expect 'while'（缺少‘while’）

编译器在错误信息所指出的位置需要找到关键字 while。当 do/while 循环的句法不正确时，通常出现该错误。
下面举例说明该错误：

```
public class Simple{
    public void method1 () {
        do{
            // do something useful here
            }; // error: missing while statement
}
```

下面的代码说明了正确的 do/while 格式：

```
public class Simple{
    public static void method1 (){
        int x = 10 ;
        do{
            System.out.printin(x);
            x--;
        }while(x !=0); // this is correct form for do/while loop
    }
    public static void main(String args[]) {Simple.method1();}
}
```

编译器错误J0019

Expect identifier（缺少标识符）

编译器需要在类名、接口、变量或方法声明之前找到标识符。当在声明中偶然丢失类型时，通常出现该错误。
下面举例声明该错误：

```
public class Simple {

    private i;    // error :type omitted

}
```

编译器错误J0020

Expect 'class' or 'interface'（缺少‘class’或‘interface’）

编译器期望在相应的声明中找到class或interface关键字。当在class和interface声明中关键字偶然丢失时，通常出现该错误。另外一种导致该错误的原因就是作用域括号不匹配。
下面举例声明该错误：

```
public Simple {//error: missing the 'class' keyword
    // Do something here
}
```

下面的例子说明了由于范围括号不匹配导致的错误：

```
Public class Simple {

    // Do something meaningful

}}    // error : additional '}' not needed
```

编译器错误J0021

Expected type specifier（缺少类型指定）

编译器期望在错误信息所指出的位置找到类型指定。当在对象实例和或变量声明中存在输入错误时，通常会发生此错误。
下面举例说明该错误：

```
public class Simple {

    public Object o = new; //error : missing 'Object' type specifier
}
```

编译器错误J0022

Expected end of file（缺少文件尾）

编译器找不到所期望的文件结束字符。当源文件在某些地方损坏时，通常出现该错误。实际检查一下来源文件中明显的错误，保存所做的改变，

并重新编译。

编译器错误 J0023

Expected 'catch' or 'finally'（缺少‘catch’或‘finally’）

在相应的 try 块后面缺少 catch 或 finally 块。
下面举例说明该错误：

```
public class Simple {
    public void method1(){
        try {
            // do something meaningful
        }
    } // error: catch or finally not found
}
```

下面的例子说明了使用 try / catch 块的正确句法：

```
public class Simple{
    public void method1(){
        try{
            // Do something here. Possibly throw an exception type
        }
        catch(Exception e){
```

```
            /*Handle any errors here and use the 'Exception' object to
            determine the type of error that occurred */
        }
    }
}
```

编译器错误J0024

Expected method body（缺少方法主体）

在方法声明后面缺少方法主体。当缺少大括号所包围的方法主体时，通常会发生此错误。当方法打算进行abstract或native操作，但abstract或native关键字在方法声明中丢失时，也会产生该错误。
下面举例说明该错误：

```
public abstract class Simple {

    public void method1();
    // error: ′abstract′ omitted
    // This error would also exist if the class is not declared ′abstract′

}
```

编译器错误J0025

Expected statement（缺少语句）

在当前作用域的结尾之前缺少一个语句。当缺少指定当前作用域开始的左大括号，或缺少左右两个大括号时，通常出现该错误。确认当前作用域的大括号是匹配的。
下面举例说明该错误：

```
public class Simple {
    void method1(int arg1) {
        if (arg1==1)
        // error: missing the left brace for the if statement
    }
}
```

编译器错误J0026

Expected Unicode escape sequence（缺少 Unicode 换码序列）

缺少有效的 Unicode 换码序列。当反斜线后面没有跟随用来表示 Unicode 换码序列的字母“u”时，通常会发生此错误。检查 Unicode 换码序列的句法，并重新编译。

下面举例说明该错误：

```
public class Simple {
    int i = \\u0032;
    // error: '\\' not valid
}
```

编译器错误J0027

Identifier too long（标识符太长）

编译器检测出某个标识符的名称长于1024个字符。缩短标识符名称，并重新编译。

编译器错误J0028

Invalid number（无效数字）

编译器检测出某个Java语言所不能支持的数字值。当指定的数字值超出任何Java的原始数据类型的作用域时，通常出现该错误，当数字值远远大于在指定数据类型中存储的值时，也会出现该错误。
下面举例说明该错误：

```
public class Simple {
```

```
    long i = 123456789012345678 90;
    // error: value out of range for 'long'
}
```

编译器错误J0029

Invalid character（无效字符）

编译器检测出不能用来作为标识符的ASCII字符。当类、接口、方法或变量标识符中包含无效字符时，通常会发生此错误。
下面举例说明该错误：

```
public class Simple {
    private int c#:
    // error : '#' not supported
}
```

编译器错误J0030

Invalid character constant（无效字符常量）

编译器检测出，对类型为char的变量，试图指定的字符或字符换码序列无效。

下面举例说明该错误：

```
public class Simple {

    char c = '\';
    // error: invalid escape character
    char x = '\\'; \* correct assignment of the backslash to the char variable */
}
```

编译器错误 J0031

Invalid escape character（无效换码字符）

编译器检测出使用了某个无效换码字符。当在 Unicode 换码序列中找到句法结构错误时，通常会发生此错误。如果在字符串赋值中使用一个“\”字符，也会出现该错误。

下面举例说明该错误：

```
public class Simple {

    int i = \u032;
    // error: Unicode uses 4 hex digits
    int x = \u0032;
    //correct assignment of a Unicode escape sequence
}
```

下例说明了由于使用不适当的字符“\”对字符串赋值所产生的错误：

```
public class Simple{

    public String str = c:\Windows\desktop;
    // error: invalid assignment of the '\' character to a string
    // To assign it correctly use the '\\' character assignment
}
```

编译器错误J0032

Unterminated string constant（字符串常量未终止）

编译器在字符串常量的结尾检测不到使之终止的双引号字符。当字符串结束符丢失或当字符串常量被错误地分成多行时，通常出现该错误。下面举例说明该错误：

```
public class Simple {

    String str = "hello;
    //error : closing quote omitted

}
```

编译器错误J0033

Unterminated comment（注释未终止）

编译器检测到注释块的开始，但检测不到有效的结尾。当注释块的结尾字符意外丢失时，通常会发生此错误。
下面举例说明该错误：

```
public class Simple {

    /* This comment block
     * does not have a valid
     * terminator

}
```

编译器错误J0035

Initializer block cannot have modifiers except 'static'（除了‘static’之外，初始化程序块不能有修饰符）

编译器检测到与初始化程序块关联的修饰符不是 static。只能使用 static 关键字，来指定初始化程序为 static 的，或者不使用修饰符表示为实例初始化程序。从错误信息指定的初始化程序中删除该修饰符，或者将 static 修饰符加入到初始化程序中，然后重新编译。
下面举例说明该错误：

```
public class Simple {

    protected{      // error: ′protected′ is invalid here
        // do something here
    }
}
```

编译器错误J0036

A data member cannot be 'native', 'abstract', or 'synchronized'（数据成员不能为‘native’、‘abstract’或‘synchronized’）

编译器在变量声明中检测到上述修饰符之一。修饰符synchronized和native只能应用于方法声明。abstract修饰符可以应用在方法、类和接口中。

下面举例说明该错误：

```
public class Simple{

    native int myvar1;
    abstract int myvar2;
    synchronized int myvar3;
    // error: the vars above have invalid modifiers

}
```

编译器错误J0037

A method cannot be 'transient ' or 'volatile'（方法不能为transient或volatile）

编译器在方法声明中检测到上述修饰符之一。修饰符transient和volatile只能应用在字段声明中。
下面举例说明该错误：

```
public class Simple{

    transient void Method1() {};
    volatile void Method2() {};
    // error: the methods above have invalid modifiers
}
```

编译器错误J0038

'final' member 'identifier' must be initialized when declared in an interface（当在接口中声明时，final成员identifier必须进行初始化）

编译器在接口定义中检测到未初始化的final变量。在接口定义中的变量声明，如final，必须在声明中设置值。一旦设置该值，就不能在程序中

改变。
下面举例说明该错误：

```
interface ISimple {
    final int COOL_RAD ;
    // error: must have value set
}
```

编译器错误J0040

Cannot define a method body for abstract/native methods（不能为abstract/native方法定义方法主体）

编译器在相应的 abstract 或 native 方法声明的后面检测到方法主体定义。abstract方法必须在子类中定义实现代码。native方法是使用本机语言中的代码实现的，如 C++。
下面举例说明该错误：

```
public abstract class Simple{
    abstract void method1(){
        // Do something here
    }// error: abstract methods are implemented in subclasses
}
```

在接口中声明的代码是隐含的 abstract。同样，当用户试图在接口中定义

方法主体时，也会出现该错误。下面的代码说明了这种情况：

```
public interface Simple {
    public void method1() {
        // error: must define this method body
        // in a class that implements the
        // 'Simple' interface
    }
}
```

编译器错误J0041

Duplicate modifier（修饰符重复）

编译器在声明中检测到一个修饰符使用了两次。在一个声明中，同样的修饰符多次使用时，就会出现该错误。
下面举例说明该错误：

```
Public class Simple {

    public public void method1() { //error: 'public ' used twice
        // do something meaningful
    }
}
```

编译器错误J0042

Only class can implement interface（只有类才能实现接口）

编译器检测到接口声明使用 implements 关键字。接口不能实现其他接口。接口只能由类实现。当接口实现错误，而不是使用 extends 关键字扩展其他接口时，会出现该错误。

```
public interface Simple implements color{
    // error: 'Simple' cannot implement the
    // 'color' interface
}
interface color {
    // do something meaningful
}
interface pattern extends color{
    // extending an interface is OK hee
}
```

编译错误J0043

Redeclaration of member 'identifier'（成员‘identifier’再次声明）

编译器检测到在相同作用域中，相同的标识符名多次声明。当一个变量或方法多次声明时，会出现该错误。确认在同一类中没有多次定义同一个方法或变量名。方法可以使用相同的名称，但为了区别，方法参数必须有所不同。
下面举例说明该错误：

```
public class Simple {
    private int i;
    public void method1 () {
        // do something here
    }
     // Other declarations for the class go here
    private int I; // error: I declared twice
    public void method1(){
        // error: method declared twice
    }
}
```

编译错误J0044

Cannot find definition for class 'identifier'（找不到类‘identifier’定义）

编译器找不到指定类的定义。该错误通常是由于输入错误而引起的。当找不到包含指定类的软件包时，也会出现该错误。
下面举例说明该错误：

```
class NotSimple{
    // do something here
}

public class Simple {

    NotSimple smp = new NotSmple();
    // error: 'NotSmple' not a valid class name

}
```

编译器错误 J0045

'identifier' is not a class name（‘identifier’不是一个类名）

编译器检测到下列几种情况时，出现此错误：

- 找不到作为 import 语句的一部分的类名。
- 导入语句语法无效。
- 类试图展开一个接口。只有类能够与 extends 语句一起使用。
- 确保这个软件包都存在，并且，导入的所有类都要在 import 语句中指定的软件包中存在，然后重新编译。

下面举例说明该错误：

```
package non.existent;
import non.existent; /* error: the package is not existent and cannot be imported */
public class Simple {
    // do something meaningful
}
```

编译器错误 J0046

'identifier' is not an interface name（‘identifier’不是接口名）

编译器检测到，implements 关键字引用的标识符不是接口。在 implements 语句中，使用类来替代接口时，通常出现该错误。
下面举例说明该错误：

```
class Simple2 {
    // do something meaningful
}
public class Simple implements Simple2 {
     // error: cannot implement class 'Simple2'
```

```
}
```

编译器错误J0048

Cannot extend final class 'identifier'（不能扩展final类‘identifier’）

编译器检测到试图细分使用final关键字声明的类。使用final声明的类不能细分子类。

下面举例说明该错误：

```
final class Simple2 {
    // do something meaningful
}
public class Simple extends Simple2 {
     // error: cannot extend 'Simple2'
}
```

编译错误J0049

Undefined name 'identifier'（‘identifier’名未定义）

编译器检测到类、方法或变量的引用不存在。发生该错误的例子包括：

- 引用的变量不存在或类型错误。
- 与方法一起使用的对象不存在或类型错误。
- 导入的类在指定的软件包中不存在或类型错误。

确保在错误信息中显示的类、方法或变量的引用是正确的，并重新编译。

```
import java.io.bogus; // error: unknown class name

public class Simple {

    public int method1 (){
        return novar1;
        // error: 'novar1' does not exist
    }
    public void method2(){
        NotSimple nt = new NotSimple();
        np.methodx();
        // error: the object reference 'np' should be 'nt'
}
class NotSimple{
    public void methodx(){
        // do something meaningful here
    }
}
```

编译器错误J0051

Undefinded package 'identifier'（软件包‘identifier’未定义）

编译器检测到软件包名称声明，但找不到该软件包的定义。当在import语句中存在句法错误时，通常出现该错误。当不能找到该软件包或软件包不存在时，也会出现该错误。
下面举例说明该错误：

```
import java.lang.bogus.*;
// error: 'bogus' not a valid package name
public class Simple {
    // do something meaningful
}
```

编译器错误J0053

Ambiguous name :'identifier' and 'identifier'（不明确的名称：‘identifier’和‘identifier’）

在显示的两个标识符之间有二义性，编译器不能确定。当相同的类名出现在两个软件包中，并且，这两个软件包都使用类导入指定符“*”导入到一个源文件中时，通常会出现该错误。确保正在导入到源文件中的软

件包中，没有两个类是同名的，并重新编译。

编译器错误J0056

Missing return type specification（没有指定返回类型）

编译器检测到方法声明没有指定返回类型。所有方法声明都必须指定一个返回类型。如果方法不需要返回值，则使用 void 关键字。
下面举例说明该错误：

```
Public class Simple {
    public method1() { //error : no return type
        // do something meaningful
    }
}
```

编译器错误J0057

Class file 'identifier ' should not contain class 'identifier'（类文件‘identifier’应该不包含类‘identifier’）

在指定的类文件中，编译器检测不到指定的类。当类被编译到类文件，而这个类文件以后又改名时，通常会出现该错误。因为类文件与类中所包含的名称不同，所以，试图使用类文件名作为类时，将会失败。要避

免这种情况，可以将类定义改变为正确的名称，或将该文件名改回到原来的文件名。

编译器错误J0058

Cannot have a variable of type 'void'（不能有 void 类型的变量）

编译器检测到一个变量声明为 void 类型。关键字 void 不允许在变量声明中使用。更确切地说，void 只能作为方法返回类型使用，这种方法实际上不返回值。
下面举例说明该错误：

```
public interface Simple {

    public final static void i = 1;
    // error : 'void ' not valid
}
```

编译错误J0059

Cannot reference member 'identifier' without an object（不能引用无对象的成员 ‘identifier’）

编译器检测到试图引用没有已知对象与之关联的变量。当一个实例字段或方法(没有被关键字 static 声明的字段或方法)从没有有效实例的 static 方法中引用时，通常会发生此错误。Static 方法不能使用没有有效的类实例的实例字段和方法。
下面举例说明该错误：

```
public class Simple {

    private int x;
    public void method1(){
        // Do something here
    }
    public static void main(String args[]) {
        x = 0; /* error: 'x' must be static or referenced using class
                  instance */
        method1(); /* error: 'method1' must be static or referenced with a class instance */
    }
}
```

下面的例子说明了引用非静态成员字段或方法的正确方法。

```
public class Simple{
```

```
    private int x; // access level of member does not matter
    public void method1(){
        // Do something here
    }
    public static void main(String args[]) {
        // Create an instance of the 'Simple' class
        Simple smp = new Simple();
        smp.x = 0 ; // valid reference to field
        smp.method1(); // valid call to method
    }
}
```

编译器错误J0060

Invalid forward reference to member 'identifier'（前面对成员‘identifier’的引用无效）

编译器检测到程序试图使用其他还未定义的变量来初始化某个变量。要避免这种情况的发生，应该改变字段声明的顺序，首先定义要引用的变量。

下面举例说明该错误：

```
public class Simple {
    private static int i = j;
```

```
    // error: 'j' not yet defined
    private static int j = 0;
}
```

下面的例子显示了正确的方法，来使用在同一个类中的其他字段初始化一个字段。

```
public class Simple {
    private static int j = 0; // field is instantiated and populated
    private static int i = j; // with 'j' properly set , 'i' can be set
}
```

编译器错误J0061

The members 'identifier' and 'identifier' differ in return type only（成员‘identifier’与‘identifier’的返回类型不同）

编译器检测到，一个子类方法试图重载一个基类方法，但是，这些方法的返回类型不同。在 Java 中，重载的方法必须通过唯一的签名来区别。唯一的签名由方法名、数字和它的参数位置和返回类型组成。
下面举例说明该错误：

```
public class Simple extends Simple2{
    public void method1() {
```

```
        // error: only return type differs
    }
}
class Simple2 {
    public int method1() {
        return 1;
    }
}
```

编译器错误J0062

Attempt to reduce access level of member 'identifier'（试图还原成员‘identifier’的访问级别）

编译器在被编译的类中检测到一个重载的方法，它降低了基类方法的访问级别。方法的基类访问修饰符必须由想要重载该方法的所有派生类进行维护。

下面举例说明该错误：

```
public class Simple{
    public void method1(){
        // Do something here
```

```
    }
}
class Simple2 extends Simple{
    private void method1(){
        /* error: cannot change access level of a method
        when overriding a base class's method. */
    }
}
```

编译器错误 J0063

Declare the class abstract , or implement abstract member 'identifier'（声明类 abstract，或实现 abstract 成员 identifier）

编译器检测到一个 abstract 方法在类、超类或实现接口中定义，但是该方法从不实现。当类实现一个接口或展开包含 abstract 方法定义的类，但该类从不实现 abstract 方法时，通常出现该错误。当类定义一个 abstract 方法，但类没有定义的 abstract 修改符时，也会出现该错误。确保类已经实现了任何 abstract 方法，并且再次编译。
下面举例说明该错误：

```
interface ITest{
    public abstract void method1();
```

```
}

public class Simple implements ITest{
    // Do something meaningful here
    // error: the abstract method defined in 'ITest' is never implemented
}
```

编译器错误J0065

Cannot assign to this expression（不能赋值到该表达式）

编辑器在lvalue赋值的通常位置上检测到一个表达式。当给表达式指定一个值，但是该表达式不可能赋这个值时，通常发生该错误。
下面举例说明该错误：

```
public class Simple {

    public void method1() {

        int x = 0 ;


      int y = 0 ;

        x++ = y; // error: '+ +' not valid
    }
}
```

编译器错误J0066

'this' can only be used in non-static methods（this不能用在非静态方法中）

编译器在类（或静态）方法中检测到关键字this的使用。类方法不传递隐含的this引用，同样，它们不能引用实例（非静态）变量或方法。下面举例说明该错误：

```
public class Simple {
    int x;
     public static void method1() {
        this.x = 12; // error: this instance specific
    }
}
```

编译器错误J0067

Connot convert 'type' to 'type'（不能从‘类型’转换到‘类型’）

编译器检测到变量类型的使用超出当前上下文。同样，编译器不能隐式将该结果转换为任何有意义的数值。将试图使用的值转换为方法或字段所需要的数值，然后再编译。

下面举例说明该错误：

```
public class Simple {
    public void method1 () {
        int i = 10;
        if (i--) {
            // error: conditional needs expression
            // or boolean field
        }
    }
}
```

编译器错误J0068

Cannot implicitly convert 'type' to 'type'（不能隐式地从‘类型’转换到‘类型’）

编译器由于没有明确的类型转换（typecast），而不能转换指定的变量。当将数字字段赋予非数字的数据类型时，通常出现该错误。使用类型转换将数据类型转换到适当的数据类型，然后再次编译。
下面举例说明该错误：

```
public class Simple {
```

```
    int i = 10;
     public void method1() {
        char c = i; // error: expected type char
        char c =(char) i; // this is the correct assignment statement
    }
}
```

编译器错误J0069

Connot apply '.' operator to an operand of type 'identifier'

编译器检测到‘.’运算符应用到无效的类型。当无效的方法调用是由固有的数据类型生成时，通常出现该错误。要确保方法调用是使用正确的对象，然后再次编译。

下面举例说明该错误：

```
public class Simple {
     int i = 123;
    int j = i.length; //error: 'i' not an array
}
```

编译器错误J0072

'identifier' is not a member of class 'identifier'

编译器检测到一个方法调用，但是该方法未被定义。当方法名拼写错或是在适当的作用域中找不到时，通常出现该错误。当方法不存在时，也会出现该错误。确保尝试引用的方法在指定的类中存在，并且再次编译。
下面举例说明该错误：

```
public class Simple {

   public static void main(String args[]) {
      System.out.printline("Hello")
      //error : 'printline' shoule be 'println'
   }
}
```

编译器错误J0074

Operator cannot be applied to 'identifier' and 'identifier' values

编译器检测到不能被正确地应用于在错误信息中表示标识符的运算符。当两个字段为非数字时，以及运算符应用于适用于字段中的数字时，通常出现该错误。
下面举例说明该错误：

```
public class Simple {
    public void method1() {
        String s1 = "one";
        String s2= "two";
        String result;
        result = s1 * s2; // error: invalid operands
    }
}
```

编译器错误J0075

Invalid call（无效调用）

编译器检测到方法调用的句法，但标识符表示了一个无效的方法名。当方法名拼写错误或包含有Java语言命名约定中不认识的字符时，通常发生该错误。

下面举例说明该错误：

```
class Simple {
   int x=l();
   // error: 'i()' is not a valid method name
}
```

编译错误J0076

Too many arguments for method 'identifier'

编译器检测一个已知的方法调用，但该调用包含的参数比需要的要多。检查用于试图调用方法的参数数量，并且从调用中删除额外的参数。下面举例说明该错误：

```
public class Simple {

    public void method1(int arg1) {

        // do something meaningful
    }

     public void method2() {

        method1(1,2); // error: Too many arguments
    }

}
```

编译错误J0077

Not enough arguments for method 'identifier'

编译器检测一个已知的方法调用，但该调用包含的参数比需要的要少。

当一个或多个参数从调用中意外丢失时，通常会出现该错误。
下面举例说明该错误：

```
public class Simple {

    public void method1(int arg1) {
        // do something meaningful
    }

     public void method2() {

        method1(); // error: Too few arguments
    }
}
```

编译器错误 J0078

Class 'identifier' doesn't have a method that matches 'identifier'

编译器检测到在一类中到已知重载方法的调用，但是使用正确的参数数量不能检测到匹配的方法。确保该重载方法调用有正确的参数数量和类型，并且再次编译。
下面举例说明该错误：

```
public class Simple {

    public void method1() {
```

```
        // do something meaningful
    }
     public void method1(int arg1) {
        // do something meaningful
    }
}
class Simple2 {
    public void method1() {
          Simple s = new Simple();
        s.method1(1,2,3);, // error: too many arguments
    }
}
```

编译器错误 J0079

Ambiguity between 'identifier' and 'identifier'

编译器不能确定要实现的正确方法。当两个重载方法具有相关的参数列表，并且方法调用不能区分这两个方法时，通常出现该错误。确保方法调用没有使用与其他重载方法有冲突的参数。避免该错误的另外一种方法就是改变这两个重载方法的参数列表，这样它们就有不同的参数数量

和唯一定义的参数类型。
下面举例说明该错误：

```
public class Simple {
    static void method1(Simple2 s2, Simple3 s3) {
        // do something meaningful
    }
    static void method1(Simple3 s3, Simple2 s2) {
        // do something meaningful
    }
     public static void main(String args[]) {
        Simple2 s2 = new Simple2();
        method1(s2, s2);
        // error: Ambiguity between Simple2 and Simple3
    }
}
class Simple2 extends Simple3 {
    // do someghing meaningful
}
class Simple3 {
    // do something meaningful
```

```
}
```

编译器错误J0080

Value for argument 'identifier' cannot be converted from 'identifier' in call to 'identifier'

编译器检测到与方法声明中所指定的参数不匹配的方法参数。当数字字段作为一个参数传递到方法中，但该方法需要不同的数字类型时，通常出现该错误。要传递不同的数字类型到方法的参数，将传递到方法的字段进行类型转换。

下面举例说明该错误：

```
public class Simple {

    public void method1(int arg1) {
        // do something meaningful
    }

     public void method2(){

        float f = 1.0f;
        method1(f); // eror: ,mismatched call types
```

```
    }
}
```

下面举例说明如何避免该错误：

```
public class Simple{
    public void method1(int arg1) {
        // do something meaningful
    }
     public void method2(){
        float f = 1.0f;
        method1(int)f); // typecast the 'float' to be treated as an 'int'
    }
}
```

编译器错误 J0081

Value for argument 'identifier' cannot be converted from 'identifier' in call to 'identifier'

编译器检测到调用到位于不同类文件中的类的方法，但是不能将所提供的某个参数类型转换到在方法声明中显示的类型。当方法使用错误的参

数顺序调用方法或调用错误的方法时，通常发生该错误。检查在错误信息中导致错误的参数数量，并且确保传递了正确的参数类型。

编译器错误J0082

Class 'identifier' doesn't have a constructor that matches 'identifier'

编译器检测不到与调用标识符匹配的构造器。当使用错误的参数数量调用构造器时，通常发生该错误。确保含有构造器的类与所试图调用的类相匹配，然后再次编译。

下面举例说明该错误：

```
public class Simple {

    Simple(int arg1) {
        // do something meaningful
    }

     public static void main (String args[]) {

        Simple s = new Simple(12,13);
        // error: too many arguments
    }
}
```

编译器错误J0083

'super()' or 'this()' may only be called within a constructor

编译器检测到在构造器之外使用 super()或 this()关键字。关键字 super()用于调用一个超类构造器，而关键字 this()用于从其他构造器中调用构造器。在基类中引用方法，用户必须使用关键字 super.。

下面举例说明该错误：

```
public class Simple {

   public void method1 () {

      super(); // error: 'super' cannot be called
   }
}
```

下面举例说明使用关键字 super.在基类中引用方法：

```
class NotSimple{
    public class method1(){
        // do something here
    }
}

public class Simple{
```

```
    public method2(){
        super.method1(); // correct way to call a method of a superclass
    }
}
```

编译器错误J0084

Cannot return a value from a 'void' method

编译器检测到试图从使用 void 的返回类型定义方法中返回值。下面举例说明该错误：

```
public class Simple {

    public void method1() {
        return 1; //error: cannot return a value
    }
}
```

编译器错误J0085

Expected return value of type 'identifier'

编译器检测到关键字 return 在声明返回特定类型的方法主体中，但是 return 没有关联的值。没有关联值的 return 语句将不会返回默认值，并且

因此必须指定一个有效的返回值。
下面举例说明该错误：

```
public class simple {

    public int method1() {
        return ; //error: must return int value
    }
}
```

编译器错误 J0086

'[]' cannot be applied to a value of type 'identifier'

编译器检测使用非数组类型使用数组括号。如果想要使用某字段作为数组，则将字段改变为有效的数组声明，并且重新编译。
下面举例说明该错误：

```
public class Simple {

    public void method1() {
        int i = 0;
        int j, x;

        x = j[i]; // error: 'j' not declared as array
    }
```

```
}
```

编译器错误J0087

'goto' statement is not currently supported by Java

关键字 goto 作为关键字定义时，在 Java 语言中不能实现。

编译器错误J0089

The case 'identifier' has already been defined in switch statement

编译器发现在同一个 switch 语句中，两个或更多的 case 语句使用相同的标识符或值。

下面举例说明这种错误：

```
public class Simple {

    public int method1 (int arg1) {

        switch (arg1) {
         case 1:
             return (int) 1;
         case 2:
             return (int) 2:
```

```
            case 2: // error: duplicate of above
                return (int) 3:;
            default:
                return (int ) 0;
        }
    }
}
```

编译器错误J0090

'default' has already been defined in switch statement

编译器发现在同一个switch语句中出现两个或更多的关键字default。下面举例说明这种错误：

```
public class Simple {

    public int method1(int arg1) {

        switch (arg1) {
         case 1:
            return (int) 1;
         case 2:
            return (int) 2;
         default:
```

```
            return (int) 3;
        default: // error: duplicate of above
            return (int) 0;
    }
  }
}
```

编译器错误J0091

'case' outside of switch statement

编译器发现在一个switch语句作用域外使用关键字case。下面举例说明这种错误：

```
public class Simple {

    public int method1() {
      case 1:  // error:no switch statement
          return 1;
    }
}
```

编译器错误J0092

Constant expression expected

编译器发现需要一个常数值的表达式使用了非常数值。通常，在一个case语句中使用一个变量时，就会出现这种错误。检查这个表达式，确保它使用了一个常数值，然后再次编译。
下面举例说明这种错误：

```
public class Simple{
    int var1 = 10;
    int var2 = 20;
    public void method1 (){
        switch (var1){
        case var2:
           // error: cannot use variable with case
        }
    }
}
```

编译器错误J0093

'break' only allowed in loops and switch statements

编译器检测到关键字 break 出现在 loop 或 switch 语句的作用域之外。下面举例说明这种错误：

```
public class Simple{
    public void method1(){
        if (true)
            break;
            // error: break allowed in loops only
    }
}
```

编译器错误 J0094

label 'identifier' not found

编译器检测到与关键字 continue 或 break 有关的标号名存在，但是没有发现标号。通常，当标号不存在，而它却被 break 和 continue 语句引用时，会出现这种错误。如果标号放在 break 和 continue 语句作用域之外，这种错误也可能出现。break 和 continue 语句必须引用位于一个循环或块之前的标号。确保 break 和 continue 语句能引用一个合法的标号，然后再次编译。

下面举例说明这种错误：

```
public class Simple{
```

```
    public int method1(){
        int y;
        for (int x = 0; x < 10; x++ ){
           y = x *2;
           if (x == 5)
              break test; // error 'test' is not defined as a label
        }
        return y;
    }
}
```

下面举例说明当应用 break 和 continue 语句时，标号的正确使用：

```
public class Simple{
    public int method1(int arg1){
        int x,y = 0;
        test: // label precedes the loop.
        if (arg1 = 0)
           return y;
        for (x=1; x< 10; x++ ){
           y = x* arg1;
           if (y <= arg1){
              y = -1;
```

```
            bradk test;
          }
        }
        return y;
    }
}
```

编译器错误J0095

'continue' only allowed in loop

编译器检测到试图在一个循环之外的作用域内应用关键字 continue。通常，当删除一个循环而把 continue 语句留在代码中时，会出现这种错误。删除循环之外的所有 continue 语句，再次编译。
下面举例说明这种错误：

```
public class Simple{
    public void method1(int arg1){
        if (arg1 ==1)
            continue;
            // error: continue only allowed in loops. Remove 'continue; '.
    }
}
```

编译器错误J0096

Class value expected

编译器检测到一个同步块，但是synchronized语句应用到一个非法类型里。通常，当用synchronized语句使用一个除了类对象实例以外的标识符时，会出现这种错误。
下面举例说明这种错误：

```
public class Simple {
    public void method1() {
        int i;
        synchronized (I){// error I must resolve to an object reference
            // Do something here
        }
    }
}
```

编译器错误J0097

'instanceof' operator expected class or array

编译器检测到instanceof运算符应用的类型不是类或数组。instanceof运算符用于确保标识符是否是一个特定的类或数组的实例。确保和

instanceof 运算符一起使用的 invalue 引用了一个类实例或数组，并且 rvalue 引用了合法的类名或数组，再次进行编译。
下面举例说明这种错误：

```
public class Simple {

    public void method1() {

        Simple2 obj = new Simple2();

        if (obj instanceof int) // error: 'int' is not a class name
            // do something meaningful
    }
}

class Simple2 {
    // do something meaningful
}
```

编译器错误 J0098

Attemp to access non-existent member of 'identifier'

编译器检测到指定的一个数组成员，但是不能标识它。通常，当试图引用一个没标类型的数组方法 length 时，会出现这种错误。当在一个对象数组中试图调用一个方法，但是这个调用没有引用数组元素时，也会出

现这种错误。
下面举例说明这种错误：

```
public class Simple {

    public void method1() {
        String j [] = new String[10];
        // intialize the array elements here
        String str = j.toUpperCase();
        // error: missing array brackets ' []'
    }
}
```

编译器错误J0100

cannot throw 'identifier'——the type does not inherit from 'Throwable'

编译器在 throw 语句里检测到一个对象不是从 Throwable 类里继承的。当一个 throw 语句使用不是从 Throwable 类里继承的类时，往往会出现这种错误。确保正在发出的异常类是一个合法的异常类。
下面举例说明这种错误：

```
class BogusException{
    // Do something useful here
}
```

```
public class Simple{

    public void method1(int arg1) {
        // Do something meaningful
        if (arg1 == 0){
           throw new BogusException();
           // error: BogusException is not a valid exception class
        }
    }
}
```

编译器错误 J0101

the type 'identifer' dos not inherit from 'Throwable'

编译器检测到在一个 catch 语句里使用了一个非法的类参数。当使用一个 catch 语句处理异常时，必须用从 Throwable 里作为参数派生的类来定义它。确保为 catch 语句定义的类是从 Throwable 里派生的，并且再次编译它。

下面举例说明这种错误：

```
public class Simple {

    public void method1() {
```

```
            try {
              // do something meaningful
            } catch (String s) {
              // error: String not a subclass
              // of 'Throwable'
            }
        }
}
```

下面举例说明使用 catch 语句捕获异常的正确方式：

```
public class Simple{
    public void method1(int arg1){
        try{
          // do something here
          if (arg1 == 0)
               throw new myException(); // throw a valid exception object
        }
        catch(myException c) {// this is a valid exception object to catch
            // handle exception here
        }
    }
}
```

```
class myException extends Throwable{
    // class definition goes here
}
```

编译器错误 J0102

handler for 'identifier' hidden by earlier handler for 'identifier'

编译器检测到一个永远不能实现的异常处理程序，因为较早的处理程序已经捕获到这个异常。当用错误的顺序书写 catch 语句时，往往会出现这种错误。

下面举例说明这种错误：

```
class Simple {

    static
    {
        try
        {
        }
        catch (Exception e)
        {
        }
        catch (ArithmeticException e)
```

```
        {
        }
        // error: any exceptions this block
        // could have caught are already caught
        // by the first catch statement
    }
}
```

编译器错误 J0103

cannot override final method 'identifier'

编译器检测到一个类方法试图超越它的一个基类方法，而这个基类方法是用关键字 final 声明的。用 final 修饰符定义的方法不能被派生类取代。

下面举例说明这种错误：

```
public class Simple extends Simple2 {

    public void method1 (){
        // error: 'method1' final in superclass
    }
}
```

```
class Simple2 {
    public final void method1 () {
        // do something meaningful
    }
}
```

编译器错误J0104

编译器检测到一个在任何情况下都不能到达的语句或声明。当从一个方法里调用一个return语句，而代码放在return语句后面时，往往会出现这种错误。如果在循环里使用一个break语句，而循环里没有任何流控制允许它下面的代码运行时，也会出现这种错误。
下面举例说明这种错误：

```
class Simple {
    public int method1(int arg1) {
        for (int y= 10; y <10;y++){
           break;
           int z = y +10; // error:break causes this line to never be run
        }
        // do something here
        return arg1;
        int x = arg1 /2;
```

```
        /*error: this line of code cannot be reached because of return statement */
    }
}
```

编译器错误J0105

Method 'identifier' must return a value

编译器检测到包含除了void以外返回类型的方法声明，但是在方法主体里没有发现关键字return。当返回一个值的方法缺少合法的return语句时，往往会出现这种错误。如果在流控制块里调用一个return语句，但是由于方法逻辑而总是不能访问这个return语句时，也可能会出现这种错误。

下面举例说明这种错误：

```
public class Simple {

    public int method1(int arg1) {
        if (arg1 == 0)
            return arg1 + 2;
    } // error:flow control prohibits a value from always being returned
}
```

编译器错误J0106

Class 'identifier' has a circular dependency

编译器检测到两个或更多的类直接或间接地试图彼此再分类。当两个类彼此扩展时，往往会出现这种错误。一个类应充当其他类的基类。
下面举例说明这种错误：

```
public class Simple extends Simple2 {
    // error: extending 'Simple2'
}
class Simple2 extends Simple {
    // error: also extending 'Simple'
}
```

编译器错误J0107

Missing array dimension

编译器检测到一个数组的初始化，但是没有检测到一个合法的数组维数。当定义一个数组但是没有定义数组的维数时，往往会出现这种错误。为使用一个数组，必须定义所有的维数。
下面举例说明这种错误：

```
public class Simple {
    public void method1() {
        int [][] I = new int [] [12];
    // error: missing first array dimension
    }
}
```

编译器错误J0108

cannot 'new' an instance of type 'identifier'

编译器检测到试图声明一个不需要使用关键字new的数据类型，当试图开始使用带有固有数据类型的关键字new，而声明不是数组时，往往会出现这种错误，确保声明的成员没有使用关键字new，除非它是一个类对象或者是数组声明。

下面举例说明这种错误：

```
public class Simple {
    public void method1() {
        String myString = new String(); /*usage OK since it is a class object*/
        int x[] = new int[10]; // usage of new here is OK
        int i = new int(5);
```

```
        // error: cannot use 'new' on 'int' types
    }
}
```

编译器错误J0109

Cannot 'new' an instance of abstract class 'identifier'

编译器检测到试图实例化一个声明为abstract的类对象。声明为abstract的类不能实例化，它只作为其他获取类的基类而存在。
下面举例说明这种错误：

```
abstract class Simple2 {
    // do something meaningful
}

public class Simple {

    public void method1() {
        Simple2 s2Object = new Simple2();
        // error: class 'Simple2' declared as abstract
    }
}
```

编译器错误J0110

Cannot 'new' an interface 'identifier'

编译器检测到试图实例化声明为 abstract 的接口对象。接口只能通过一个类来实现，所以不能用类的方式来实例化。

> **注意：**默认情况下，接口是默认的，不管在声明里是否使用了关键字 abstract。

下面举例说明这种错误：

```
interface Simple2 {
    // do something meaningful
}

public class Simple {

    public void method1() {
        Simple2 s2Object = new Simple2();
        // error: interface Simple2 is abstract
    }
}
```

编译器错误J0111

Invalid use of array initializer

编译器检测到试图初始化一个数组,但是初始化语句句法不正确。数组可以在声明里用初始设定值来初始化。当试图用不正确的大括号数和逗号数以及不正确的大括号位置和逗号位置初始化一个数组时，往往会出现这种错误。确保初始化数组的语法正确，再次进行编译。

下面举例说明这种错误：

```
public class Simple{
    public void method1() {
        int [] i = {{1,2,3}, {4,5,6,}};
        // error: "i" delcared for only one dimension
     }
}
```

下面举例说明数组初始化的正确句法：

```
public class Simple{
    public void method1(){
        int [] i = (1,2,3,4,5,6);// single dimension initialization
        int [] []x = {{1,2,3},{4,5,6}}; // multi-dimension initialization
    }
}
```

编译器错误J0112

Cannot assign final variable 'identifier'

编译器检测到试图改变声明为final的字段的值。一个声明为final的字段一旦用声明里的值或一个实例初始值或构造器初始化时，这个字段不能再赋一个值。

下面举例说明这种错误：

```
public class Simple {
    private final int i = 3;
     public void method1(int arg1) {
       i = arg1;
       // error: variable "i" declared final
     }
}
```

编译器错误J0113

Call to constructor must be first statement in constructor

编译器检测到从第二个构造器主体里调用的构造器，但是构造器调用没有放在第二个构造器主体的开始处。在一个构造器里，必须在构造器主体代码的第一行调用另一个构造器。确保构造器调用位于构造器主体代

码的第一行，再次进行编译。
下面举例说明这种错误：

```
public class Simple {
    int i, j;
    Simple () {
        i = 0;
    }
     Simple(int arg1) {
        j= arg1;
        this(); // error: call to Simple() must be first
    }
}
```

编译器错误 J0114

Cannot reference 'this' in constructor call

编译器检测到在一个构造器里不恰当地引用了 this。this 语句在构造器里通常用于访问方法和构造器类的字段。在一个构造器里使用 this(this)或 super(this)将导致这种错误出现，因为类实例还没有创建，所以不能传递到另外一个构造器。

下面举例说明这种错误：

```
class SuperSimple {
    SuperSimple() {}
    SuperSimple(Object o) {}
}
public class Simple extends SuperSimple {
    int x;
    public Simple ()
    {
            this(10); // this is OK; calls another constructor
        super(this);
        // error: cannot pass this to a super constructor
        this.x = 1; // this is OK
        this.method1(); // this is OK too
    }
    public Simple(int arg1) {
    this.x = arg1; // this is OK
    }
    public void method1() {}
}
```

编译器错误J0115

Cannot call constructor recursively(directly or indirectly)

编译器检测到递归的构造器调用。当一个构造器调用相同的构造器时，往往会出现这种错误。如果一个构造器调用第二个构造器，而第二个构造器又返回来调用第一个构造器时，也会出现这种错误。
下面举例说明这种错误：

```
public class Simple {
    Simple (int arg1) {
        this (1);
    // error: constructor calling itself
    }
}
```

编译器错误J0116

Varible 'identifier' may be used before initialization

编译器检测到试图在变量正确初始化之前使用这个变量。为了在赋值或表达式中使用一个变量，必须给这个变量指定一个值。在构造器里或字段初始化器里初始化这个变量，再次进行编译。

下面举例说明这种错误：

```
public class Simple {

    static
    {
        int i;
        int j = i;
        // error: "i" not yet initialized
     }
}
```

编译器错误 J0117

Cannot declare an interface or outer class to be 'private'

编译器检测到在外部类或接口声明中使用了修饰符 private。这个修饰符只能通过字段、方法和内部类声明来使用。

下面举例说明这种错误：

```
private class Simple {

   // error: a class cannot be 'private'

}
```

编译器错误J0120

Divide or mod by zero

编译器检测到被零除的除法错误。
下面举例说明这种错误：

```
public class Simple {

    final int x=0;
   int y=1% x;
   // error: x cannot be 0

}
```

编译器错误J0121

Unable to recover from previous error(s)

编译器遇到一系列错误，不能继续可靠地处理文件。确保所有错误都已经消除再次进行编译。

编译器错误J0122

Exception 'identifer' not caught or declared by 'identifier'

编译器检测到发出的异常，但是在这个异常类里从来都没有捕捉到这个异常。当一个方法调用声明为发出异常的另外一个方法时，往往会出现这种错误。为了使一个方法能调用发出异常的另外一个方法，这个方法必须声明为发出异常或者使用一个 try/catch 组合处理错误。
下面举例说明这种错误：

```
class SimpleException extends Exception {
    // do something meaningful
}
class Simple {

    void method1() throws SimpleException { }
    void method2() { method1(); }
    // error: exception not declared for method2
}
```

下面举例说明如何调用声明为发出异常的方法：

```
/*This example illustrates handling by declaring the other method as
throwing an exception duplicate to the method it is calling.*/
class SimpleException extends Exception{
    // do something here
}
public class Simple{
```

```
    void method1() throws SimpleException{
        // do something here
    }
    void method2() throws SimpleException{
        method1(); // caller of method2 now is forced to handle exception
    }
}
/*This example illustrates handling the exception using a try/catch combination.*/
calss SimpleException extends Exception{
    // do something here
}
public class Simple{
    void method1() throws SimpleException{
        // do something here
    }
    void method2(){
        try{
            method1();
        }
        catch(SimpleException e){
            // handle exception here
        }
```

```
    }
}
```

编译器错误J0123

Mutiple inheritance of classes is not supported

编译器检测到一个类试图把关键字 extends 应用到一个以上的基类中。在其他语言中这可以定义成多重继承，而 Java 语言不支持类的多重继承。

下面举例说明这种错误：

```
public lcass Simple extends BaseClass1, BaseClass2 {
    // error: multiple inheritance not supported in Java
}

class BaseClass1 {

    // do something meaningful
}

class BaseClass2 {

    // do something meaningful
}
```

编译器错误J0124

Operator cannot be applied to 'identifier' values

编译器检测到一个运算符应用到它不能使用的类型中。确保正试图使用的运算符的变量类型或对象是合法的，再次进行编译。

下面举例说明这种错误：

```
public class Simple {

    void method1(boolean b) {

        b++;

    /* error: post increment operator cannot
       be applied to boolean variables */

    }
}
```

编译器错误J0125

'finally' block used without 'try' statement

编译器检测到一个finally块，但是没有发现相应的try语句。一个finally块用于实现一个try语句后面的代码，而不管try语句的结果。

下面举例说明这种错误：

```
public class Simple {
    public void method1() {
        finally {
        // error: missing corresponding try statement
        }
    }
}
```

下面举例说明 finally 块的正确使用：

```
public class Simple{
    public int method1(int arg1) {
        try{
            arg1/10;
        }
        catch(Exception e){
            // handle exception here; must come before 'finally'
        }
        finally{
            // do something here; this section is run regardless of 'try'
        }
    }
```

```
}
```

编译器错误 J0126

'catch' block used without 'try' statement

编译器检测到一个 catch 语句，但是没有发现相应的 try 语句。为了使用一个 catch 语句,必须在它之前有一个 try 语句。确保在 catch 语句之前有一个合法的 try 语句,再次进行编译。

下面举例说明这种错误：

```
public class Simple {

    public void method1() {

        catch {
         // error: missing corresponding try statement
        }
    }
}
```

下面举例说明 catch 块的正确使用：

```
public class Simple {

    public void method1() {
        try{
```

```
            // do something here
        }
        catch (Exception e){
            // handle exceptions from try statement here
        }
    }
}
```

编译器错误 J0127

'else' keyword used without 'if' statement

编译器检测到关键字 else，但是没有发现相应的 if 语句。当程序中有 else 语句位置的作用域问题时，往往会出现这种错误。如果缺少与 else 语句相匹配的 if 语句时，也会出现这种错误。
下面举例说明这种错误：

```
public class Simple {

    public void method1(int arg1) {
        if (arg1 == 0){
            // Do something here
            else{}
                // error 'else' is inside of the 'if' block instead of outside
```

```
        }
    }
}
```

编译器错误J0128

Cannot declare an interface to be 'final'

编译器检测到使用关键字final声明的一个接口。接口不能定义为final，所以不能使用final修饰符。从接口声明中删除关键字final，再次进行编译。

下面举例说明这种错误：

```
final interface Simple {

    /*error: 'final' only applies
       to classes, methods,or variables*/

}
```

编译器错误J0129

Cannot declare a class to be 'identifier' and 'identifier'

编译器检测到使用修饰符声明的一个类不能组合。确保应用到类里的修饰符彼此没有冲突，再次进行编译。
下面举例说明这种错误：

```
public abstract final class Simple {

    /*error: 'abstract' and 'final' cannot
       be used together in a class declaration */

}
```

编译器错误J0130

Cannot declare an interface method to be 'native', 'static', 'synchronized' or 'final'

编译器检测到在接口方法声明中使用了错误信息里显示的关键字之一。因为一个接口方法没有实现代码，它不能声明为native,static,synchronized或 final。
下面举例说明这种错误：

```
interface Simple {

     public final void method1();


    /*error: 'method1' cannot be declared
```

```
        as final in an interface*/
}
```

编译器错误J0131

Cannot declare a method to be 'identifier' and 'identifier'

编译器检测到在一个方法声明中使用了两个或更多的不一致的修饰符。当使用两个访问修饰符如public和 private定义一个方法时，往往会出现这种错误。确保这个方法的修饰符彼此不冲突，再次进行编译。

下面举例说明这种错误：

```
public class Simple {

    public private void method1() {

        // error: modifiers 'public' and 'private'
        // cannot be combined in a declaration
    }
}
```

编译器错误J0132

Cannot declare a field to be 'identifie' and 'identifier'

编译器检测到在一个变量声明中使用了两个或更多的不一致的修饰符。当使用两个访问修饰符如 public 和 private 定义一个字段时，往往会出现这种错误。确保这个字段的修饰器彼此不冲突，再次进行编译。
下面举例说明这种错误：

```
public class Simple {

    public private int i;
    //error: modifiers 'public' and 'private'
    // cannot be combined in a declaration
}
```

编译器错误 J0133

Constructors cannot be declared 'native', 'static', 'synchronized ' or 'final'

编译器检测到在一个构造器声明中使用了上面显示的修饰符之一。确保这个构造器没有使用错误信息中提到的任何一个修饰符定义，再次进行编译。
下面举例说明这种错误：

```
public class Simple {

    final simple() {}
```

```
    // error: constructors cannot be 'final'
}
```

编译器错误J0134

Interfaces cannot have constructors

编译器检测到一个包含构造器声明的接口。因为一个接口不能实例化，不能为接口定义构造器。如果正在使用与一个接口相同的名字定义一个方法。确保它有一个与构造器声明中不同的相应修饰符。
下面举例说明这种错误：

```
interface Simple {

    Simple() ;
    // error: interfaces cannot
    // declare constructors

}
```

编译器错误J0135

Interface data members cannot be declared 'transient', 'volatile', 'private' or' protected'

编译器检测到在一个接口成员变量的声明中使用了上面显示的修饰符之一。因为接口是公共的并且不能实例化，所以这些修饰符都不能应用，而只应该通过类来使用。
下面举例说明这种错误：

```
interface Simple {

    volatile int i=1;

    // error: 'volatile' cannot be used

}
```

编译器错误J0136

Public class 'identifier' shoud not be defined in 'identifier'

编译器检测到在一个源文件中有一个以上的类使用修饰符 public 声明。从其他类中删除 public 访问修饰符，并且确保这个将通过类文件来公布的类声明为 public。也可以把需要声明为 public 的类移到它们自己的源文件中。如果一个源文件有一个与在它内部定义的 public 类不同的名字，或者 public 类与源文件万一不匹配时，也会出现这种错误。给源文件改名或把源文件内部定义的 public 类改名以便它们一致，再次进行编译。
下面举例说明这种错误：

```
public class Simple {

    //do something meaningful

}

public class Errorclass {

    // error: only one class may be defined as
    // 'public' within the same source file
}
```

编译器错误J0138

Interface cannot have static or instance initializer

编译器检测到在一个接口里有一个 static 初始值或实例初始值。因为一个接口没有获得实例化，不能在一个接口里定义初始值。为设置接口字段的值，应在声明时，就初始化它们。
下面举例说明这种错误：

```
interface Simple {
    int x=10; //This is OK
    {
        // error: initializers cannot
        // be used in interfaces
```

```
    }
}
```

编译器错误J0139

Invalid label

编译器检测到一个非法标号。标号必须用一个非数值字符开始。修改这个标号，再次进行编译。

下面举例说明这种错误：

```
public class Simple {
    public int method1 (int arg1) {
        123:
            // error: label cannot begin with a number.
            Return arg1*2;
    }
}
```

编译器错误J0140

Cannot override static method 'identifier' with non-static method 'identifier'

编译器检测到试图从一个子类里取代一个 static 方法。声明为 static 方法不能被取代。
下面举例说明这种错误：

```
public class Simple {

    static void method1() {}
}

class SimpleSubclass extends Simple {

    void method1() {}
    // error: cannot override
    // static method
}
```

编译器错误 J0141

Argument cannot have type 'void'

编译器检测到一个定义为 void 类型的方法自变量。void 类型只能用于把返回值声明为无返回值方式的方法。修改自变量的数据类型，再次进行编译。
下面举例说明这种错误：

```
public class Simple {
```

```
    public void method1 (void i) {
        //error: type void can only
        // be used as a return value
    }
}
```

编译器错误J0142

Cannot make static call to abstract method 'identifier'

编译器检测到试图直接调用一个抽象方法。定义抽象方法是为细分子类实现的方法提供一个定义;所以abstract方法没有实现代码。由于abstract方法缺少这种实现过程，所以使用 super 关键字调用一个抽象方法是非法的。

下面举例说明这种错误：

```
public abstract class Simple {

    abstract int method1();
}

class SimpleSubclass extends Simple {

    int method1() {

        return super.method1();
```

```
        // error: cannot call abstract mehtod
    }
}
```

编译器错误J0143

Cannot throw exception 'identifier' from static initializer

编译器检测到试图从一个静态初始值中发出异常。当调用一个 throw 语句或在一 个静态初始值作用域中出现静态类实例的初始值时，往往会出现这种错误。为了在一个静态初始值中捕获来自静态类实例的异常，可以联合使用 try/catch 块。
下面举例说明这种错误：

```
public class Simple {

    static {
        ThrowClass TClass = new ThrowClass();
        // error: cannot throw exceptions
        // within static initializers
    }
}

class ThrowClass {
```

```
    ThrowClass() throws Exception{}
}
```

下面举例说明在一个静态初始值中初试化静态类实例时，如何联合使用try/catch块捕获可能发生的错误。

```
public class Simple{
    static ThrowClass thr;
    static{
        try{
            ThrowClass thr = new ThrowClass();
        }
        catch (Exception e){
            // Handle errors here from initialization of 'ThrowClass'
        }
    }
}
class ThrowClass {
    ThrowClass() throws Exception(){
        // Do something here
    }
}
```

编译器错误J0144

Cannot find definition for interface 'identifier'

编译器不能为命名的接口找到定义。当一个实现的接口丢失或拼写错误时，往往会出现这种错误。检查正在实现的接口的位置和名字，再次进行编译。

下面举例说明这种错误：

```
public class Simple implements Bogus {
    // error: the interface 'Bogus' does not exist
}
```

编译器错误J0145

Output directory or file too long: 'identifier'

保存的输出目录或源文件的长度超过228个字符。缩短输出目录路径或源文件的长度，再次进行编译。

编译器错误J0146

Cannot create output directory 'identifier'

不能创建输出目录。当不允许在指定的驱动器上写入时，往往会出现这种错误。

编译器错误 J0147

Cannot access private member 'identifier' in class 'identifier' from class 'identifier'

编译器检测到试图非法访问包含在另一个类里的局部成员。局部类成员只能从这个成员类里访问。一个类的局部成员也只能从它的内部类里访问。

下面举例说明这种错误：

```
class AccessClass {


    private int i = 0;

}

public class Simple {

     public void method1() {

        AccessClass ac = new AccessClass();

          ac.i = 1;
```

```
        // error: cannot access 'i'
    }
}
```

编译器错误J0148

Cannot reference instance method 'identifier' before superclass constructor has been called

编译器检测到试图在调用超类构造器之前引用一个方法实例。当使用super（）语句在一个子类构造器中调用一个基类方法时，往往会出现这种错误。如果子类使用 super（）语句从构造器中调用它自己的方法时，也会出现这种错误。这种错误出现的原因是调用构造器的同时，子类和基类的实例没有进行实例化。为避免这种情况，使用 super.语句调用一个基类方法，并且使用 this.语句调用一个子类方法。
下面举例说明这种错误：

```
abstract class Simple {
    Simple (int i) {}
     int method1() {
        return 0;
    }
```

```
}
class SimpleSubclass extends Simple {
    SimpleSubclass() {
        super (method1());
      // error: constructor must be called first
    }
}
```

在上面的例子中使用抽象类，下面举例说明如何从构造器里调用基类方法：

```
class SimpleSubclass extends Simple {
    SimpleSubclass () {
        Super.method1 ();
    }
}
```

编译器错误 J0150

Cannot have repeated interface 'identifier'

编译器检测到在一个类声明中重复使用一个接口名字。当一个类实现许多接口，而其中的一个接口在 implement 表里具有双重入口时，往往会

出现这种错误。确保没有双重接口入口，再次进行编译。
下面举例说明这种错误：

```
interface SimpleI {
    // do something meaningful
}

class Simple implements SimpleI.Icolor.IFont.SimpleI {
//error: 'SimpleI' repeated
}
```

编译器错误J0151

Variable ‘identifier’ is already defined in this method

编译器检测到在一个类的相同作用域内有使用相同名字定义两次的两个变量。确保在相同的作用域内变量没有定义两次，或没有把变量定义成与传递到方法的一个自变量有相同的名字，再次进行编译。
下面举例说明这种错误：

```
public class Simple {

    public void method1() {

        int i = 1;
      int j = i;
```

```
        // more code here
        int i = 0;
        // error: 'i' defined twice within
        // the same scope
    }
}
```

编译器错误 J0152

Ambiguous reference to 'identifier' in interface 'identifier' and 'identifier'
编译器检测到标识符的引用具有二义性。可能在两个或更多的接口生命了标识符，而编译器不能确保要引用使用哪一个。确保没有使用两个接口定义相同的字段。
下面举例说明这种错误：

```
interface Interface1 {
    final int i = 0;
}

interface Interface2 {
    final int i = 1;
}

public class Simple implements Interface1, Interface2 {
```

```
    int method1() {
        return i; // error: cannot determine which instance of 'i' to use
    }
}
```

编译器错误J0158

Class 'identifier' already defined

编译器检测到使用相同的名字定义了两个或更多的类。确保在同一个源文件或软件包里的类定义次数没有超过一次（定义为外部类）。如果一个类被一个导入类复制，也会出现这种错误。重新命名其中的一个类或删除双重实例，再次进行编译。
下面举例说明这种错误：

```
public class Simple {
    // do something meaningful
}

class Simple {
    // error: class 'Simple' already defined
}
```

编译器错误J0159

'@' must be followed by the response filename

编译器检测到在JVC命令行里有@字符，但是没有检测到紧随其后的合法响应文件。提供响应文件，再次进行编译。

编译器错误J0160

Response file 'identifier' could not be opened

编译器不能打开响应文件。当响应文件名拼写错误或文件不存在时，往往会出现这种错误。检查响应文件的位置，再次进行编译。

编译器错误J0161

Cannot open source file: 'identifier'

不能打开错误信息中指出的源文件。当错误信息中指出的源文件名拼写错误或文件不存在时，往往会出现这种错误。检查所指出的文件的位置，再次进行编译。

编译器错误J0162

Failed to initialize compiler

初始化失败。这种错误经常发生的原因是编译器和（或）Java 上的 Microsoft 虚拟机（VM）没有正确安装或者是非正确版本。确保这个正确版本和 VM 及编译器的安装。

编译器错误J0163

Array 'identifier' missing array index

编译器检测到访问一个数组类型，但是缺少下标值。为访问一个数组元素，必须为数组提供一个合法的整数型下标。确保数组的下标使用了合法的整数，再次进行编译。

下面举例说明这种错误：

```
public class Simple {

    int j[]={1, 2, 3};
    void method1() {
        j[]=0;
        // error: 'j' missing index value
    }
}
```

编译器错误J0164

Ambiguous import of class 'identifier' from more than one package

编译器检测到两个或更多的import语句试图从不同的软件包里导入相同的类名。当两个软件包包含相同的类，而两个软件包都导入到同一个源文件里时，通常会出现这种错误。检查与相同类有关的导入到源文件中软件包。从一个软件包里删除相同类，或者从原文件中删除一个 import 语句。

下面举例说明这种错误：

```
import Box.Test; // This package contains a class named 'Text'
import Carton.Test; // This package contains a class named 'Text' also
// error: which 'Text' class should be used by the compiler?

public class Simple {
    // Do something here
}
```

编译器错误J0165

Cannot throw exception 'identifier' from method 'identifier'——it is not a subclass of any exceptions thrown from overridden method 'identifier'

编译器检测到一个被超越的方法试图比超越它的方法发出更多的异常。

在 Java 中，一个超越方法不能声明为发出比被超越方法更多的异常。把基类 throws 的异常改为 1，或把基类声明改为子类需要发出的发出异常类型。
下面举例说明这种错误：

```
class ExceptionA extends Exception {
    // do something meaningful
}
calss ExceptionB extends Exception {
    // do something meaningful
}
class AnotherClass {
    public void method1() throws ExceptionA {
        // do something meaningful
    }
}
public class Simple extends AnotherClass {
    public void method1() throws ExceptionA, ExceptionB {
        // error: cannot throw greater than
        // one exception here
    }
```

```
}
```

编译器错误J0166

Cannot access member 'identifier' in class 'identifier' from 'identifier'——it is in a different package

编译器检测到试图非法引用在不同软件包内定义的变量成员或方法。当试图访问 定义在另一个软件包里的 protected 或默认访问成员时，通常会出现这种错误。位于不同软件包里一个类的 protected 或默认访问成员是不能访问的。确保在另一个软件包里正试图访问的成员不是 protected 或默认访问成员。

编译器错误J0167

Cannot override non-static method 'identifier' with static method 'identifier'

编译器检测到试图使用修饰符 static 声明的子类方法超越一个超类方法。当一个方法在一个子类里被超越时，这个方法不能提高或降低方法的访问级别，也不能使用 static 修饰符。从超越方法声明中删除 static 修饰符，再次进行编译。
下面举例说明这种错误：

```
public class Simple {
    public void method1() {
        // do something meaningful
    }
}
class Simple2 extends Simple {
    static public void method1() {
        // error: voerriding superclass 'method1'
        // with a static method is not valid
    }
}
```

编译器错误J0168

The declaration of an abstract method must appear within an abstract class

编译器检测到在一个没有定义为abstract的类内使用修饰符abstract声明的方法。当打算把一个类声明为abstract，但在类声明中丢失abstract修饰符时，往往会出现这种错误。改变类使它声明为abstract或从类内定义的方法中删除修饰符。

下面举例说明这种错误：

```
public class Simple {
    abstract void method1();
      // error: class must also be abstract
}
```

编译器错误J0169

Cannot access 'identifier'——only public classes and interfaces in other packages can be accessed

编译器检测到试图访问包含在另一个软件包里的一个非公共类或接口。只有用修饰符 public 定义的类或接口才能在其他软件包里访问。检查正在其他软件包里访问的类或接口的访问级别，确保它是 public，再次进行编译。

下面举例说明这种错误：

```
// Source located in 'Boxes.Java' in the 'Box' package
package Box;
public class Box{
    TapeRoll tr = new TapeRoll();
    // Do meaningful stuff here
}
class TapeRoll{
```

```
    // Define the class here
}

// Source located in 'Simple.java'
import Box.TapeRoll;
public class Simple{
    public static void main(String args[]){
        Box.TapeRoll tr = new Box.TapeRoll();
        // error: cannot access a 'non-public' class in a different package
    }
}
```

编译器错误 J0170

Cannot load predefined class 'identifier'

编译器试图装入一个预先定义的类，但是不能找到相应的文件。当在系统上找不到 Java API 类文件或 Java 上的 Microsoft 虚拟机（VM）没有安装适当的版本时，往往会出现这种错误。确保系统上有 Java API 类文件以及 VM 的正确安装，再次进行编译。

编译器错误J0173

Found class 'identifier' in package 'identifier' rather than package 'identifier'

编译器发现指定的类，但是这个类没有定义成正确的软件包成员。当从一个错误的目录导入一个类文件时，通常会出现这种错误。确保正导入的类位于正确的软件包目录，再次进行编译。

编译器错误J0175

Cannot invoke method on 'null' literal

编译器检测到试图从 null 关键字中调用一个方法。null 不是一个类对象而且也不提供方法。删除试图从 null 关键字中调用方法的语句，再次进行编译。

下面举例说明这种错误：

```
public class Simple{
    public String method1(){
        // Do something meaningful here
        return null.toString();
        // error: cannot invoke a method from null
    }
```

```
}
```

编译器错误J0176

Duplicate label 'identifier' nested inside another label with same name

编译器检测到一个嵌套标记与另一个标记相同。给这个标记改成不同的名字。修改引用这个标记的所有 break 和 continue 语句，再次进行编译。下面举例说明这种错误：

```
public class Simple{
    void method1(){
        outsideLoop:
        for (int i=0; i<10; i++)
        {
            outsideLoop: // error: duplicate label
            for (int x=0;x<10;x++)
            {
                break outsideLoop;
            }
            break outsideLoop;
        }
    }
}
```

编译器错误J0189

'return' not allowed in a static initializer or instance initializer

在一个静态或实例初始化里发现一个return语句。初始化像一个构造器一样，不能返回值。删除return语句，再次进行编译。

下面举例说明这种错误：

```
public class Simple {

    static int var1;

     static {
        var1 = 0;
        return;
        // error: return statement not allowed in static initializer
    }
}
```

编译器错误J0191

Expected '.class'

编译器检测到在一个表达式或赋值语句中使用了一个内在类型的名字，但是在名字后面没有发现。当漏掉关键字.class时，通常会出现此错误。

把.class 作用域加到内在类型的后边，再次进行编译。
下面举例说明这种错误：

```
public class Simple {
    public static void main (String args[]){
        Class x=int; //error: missing '.class'
    }
}
```

编译器错误J0192

'.class' on intrinsic type requires Java 1.1 compatible class libraries

编译器检测到通过一个内在数据类型使用.class，但是Java上的Microsoft虚拟机（VM）或Java类库是基于Java 1.0的。确保类库和Java VM是Java 1.1版本，再次进行编译。

编译器错误J0193

Cannot have an array of type 'void'

编译器发现试图定义一个类型为void的数组。void数据类型用在方法上用来声明这个方法没有返回值，不能用于数组。

```
Public class Simple {
```

```
    Void MyArray[]; // error: void arrays not supported
}
```

编译器错误J0194

Class or interface cannot be declared 'volatile'，'native'，'transient'，or 'synchronized'

编译器检测到使用上边提到的修饰字声明的一个内部类或接口。当把一个方法或字段修饰字用到内部类或接口定义时，往往会出现这种错误。内部类和接口可以使用private，public和protected访问修饰字。内部类还可以使用修饰字，例如，abstract，static和final。

下面举例说明这种错误：

```
public class Simple{
    // Do something here
    volatile class InnerClass{
        /*error: like outer class, inner classes cannot be defined as volatile */
    }
}
```

编译器错误J0195

Cannot declare 'identifier' as 'static' in inner class 'identifier'

编译器检测到试图在一个内部类中把一个变量或方法声明为 static。和常规的类声明不同，内部类不支持静态成员。在另一个类内定义的类声明为 static 时，可以有静态成员，但是这个类被认为是外部类。如果在一个内部类中定义一个接口，也会出现这种错误。
下面举例说明这种错误：

```
public class Simple{

    // Do something meaningful here

     class InnerClass{

    static int var1; /* error: cannot declare static members in inner class*/

    }
}
```

下面举例说明如何在一个类内定义包含 static 成员的类：

```
public class Simple{

     // Do something meaningful here

    /*Because 'InnerClass' class is declared as static it is now treated
        as an outer class enclosed within the 'Simple' class*/
    static class InnerClass{
        static int var1 = 100; // This is OK as long as the class is static
    }
```

```
}
```

编译器错误J0196

Nested class 'identifier' cannot have the same name as any of its enclosing classes

指定内部类与嵌套下的一个类有相同的名字。确保在嵌套的内部类中没有重复类名，再次进行编译。

下面举例说明这种错误：

```
public class Simple{

    // Do something meaningful here
    class InnerClass{
        // Do something meaningful here
           class Simple{
              // error: inner class has same name as a parent class
           }
    }
}
```

编译器错误J0197

Cannot declare interface in inner class 'identifier'

编译器检测到试图在一个内部类内部声明一个接口。内部类定义不支持在其内部声明一个接口。从内部类声明中删除接口声明，再次进行编译。下面举例说明这种错误：

```
public class Simple{
    // Do something meaningful here
    class InnerClass{
    // Do something meaningful here
       interface MyInterface{
         /*error: interfaces cannot be declared
                  inside inner classes */
        }
    }
}
```

编译器错误J0198

An enclosing instance of type 'identifier' is required

编译器检测到一个内部类定义试图引用它作用域之外的东西。出现这种

错误的情况有：
在另外一个类里声明为 static 的类引用它父类中的一个非静态变量或方法。因为这个类声明为 static，如果没有父类定义的实例，它不能引用父类中的任何成员。
一个内部类试图引用一个不是它外部类的类，而这个被引用类使用在它前面带有类名的关键字 this。因为这个引用类不是内部类的父类，所以不能应用一个实例。
下面举例说明这种错误：

```
// This example illustrates the first error situation
public class Simple{
    int x = 10;
    static class InnerClass{
        public void method1(){
            int y = x; /*error: instance variable needed
                                to reference parent variables. */
        }
    }
}
// This example illustrates the second error situation
class A{
    int x;
```

```
}
class B{
    // Do something meaningful here
    class InnerClass{
        void method1(){
            int y = A.this.x;
            /*error: no instance of A defined. Cannot use
                    the <classname.this.variable> syntax here.*/
        }
    }
}
```

编译器错误J0199

Call of 'this()' cannot be qualified

编译器检测到一个内部类构造器试图通过使用带有 this（）方法的类名调用它的外部类构造器。内部类不能调用它们的外部类构造器。从内部类构造器删除调用的外部类构造器，再次进行编译。
下面举例说明这种错误：

```
public class Simple{
    int x;
    Simple(int x){
```

```
        this.x = x;
    }
    class InnerClass{
        InnerClass(){
            Simple.this(10);
            // error: cannot call outer class constructor
        }
    }
}
```

编译器错误J0200

'this' must be qualified with a class name

编译器检测到试图在一个内部类里使用关键字 this 引用外部类成员，而关键字 this 带有的名字不是外部类名。为引用外部类成员，只能从一个内部类使用外部类名。

下面举例说明这种错误：

```
public class Simple{
    int x;
    class InnerClass{
        void method1(){
            int j = x.this;
```

```
            /*error: only a class name can be used with
            this to reference outer class */
        }
    }
}
```

下面举例说明引用外部类成员正确的方式：

```
public class Simple{

    int x;
    int method2(int arg1){
        return arg1 *;
    }
    class InnerClass{
        void method1(){
            int j = Simple.this.x; // This is OK!
            int z = Simple.this.method2(10);
        }
    }
}
```

编译器错误J0201

'super' cannot be qualified except as a superclass constructor call

编译器检测到使用前面带有超类实例的关键字 super 来访问超类的字段或方法。应当与关键字 super 一起使用超类实例，限于在超类内部有一个定义的内部类时，引用一个超类构造器。

下面举例说明这种错误：

```
class Simple{
    int x;
}

class NotSimple extends Simple{
    NotSimple(Simple smp){
        smp.super.x = 100;
        /*error Cannot reference field with superclass name*/
        super.x = 100; // This is OK!
    }
}
```

编译器错误J0202

'super()' cannot be qualified;superclass 'identifier' is not an inner class

编译器检测到使用超类实例调用超类构造器，但是这个超类不是一个内部类。只有一个超类是内部类时，超类实例与关键字 super 才能一起使用。

下面举例说明这种错误：

```
class Simple{
    // Do something meaningful here
}
class NotSimple extends Simple
{
    NotSimple(Simple smp){
        smp.super();
        /*error: cannot call super with instance when
           superclass does not contain inner classes*/
    }
}
```

下例给出带有 super 关键字的超类实例的正确用法：

```
class Simple{
    // Do something here
    class InnerClass{
        int var1, var2;
    }
```

```
}
public class NotSimple extends Simple.InnerClass{
    NotSimple(Simple smp){
        smp.super();
        // This is OK!
    }
}
```

编译器错误J0203

Cannot access protected member 'identifier' in class 'identifier' from class 'identifier'

编译器检测到试图从一个类的不同软件包中访问保护成员。一个类的保护成员可以从软件包的外部访问，在这个软件包中，只通过负责类实现的代码声明。删除到其他软件包的保护成员的调用，或者使这个类成为其他软件包类的一个子类，重新编译。

下面举例说明这种错误：

```
/*(source located in a file called PublicClass.java in
    the the Boxes Package directory) */
package Boxes;
public class PublicClass{
```

```
    protected void method1(){
        // Do something here
    }
}

// (source located in a file called Simple.java)
import Boxes.PublicClass;

public class Simple extends PublicClass{
    public void method1(){
        PublicClass pub = new PublicClass();
        pub.method1();
        /*error: Cannot access protected method 'method1' because it
                 is located in a different package.*/
    }
}
```

编译器错误J0204

Cannot access protected member 'identifier' in class 'identifier' via a qualifier of type 'identifier'

编译器检测到一个软件包里的一个类是另一个软件包的扩展类，而这个类试图使用一个基类实例访问基类的保护成员。当一个类试图通过一个

实例而不是 this, super 或派生类的实例访问它基类的成员时，通常会出现这种错误。确保被访问它基类的成员使用 this 或 super 关键字或一个派生类的实例，再次进行编译。
下面举例说明这种错误：

```
/*(source located in a file called Point.java located in the Boxes package directory) */
    package Boxes;
public class Point{
    protected int x, y;
    // Do other meaningful code here
}
// (source located in a file called simple.java)
import Boxes.Point;
public class Simple extends Point{
    public void method1(Point p){
        super.x = 0; // this is OK!
        p.x = 0; /*error: cannot use a protected member
                    for usage other than extending it. */
    }
}
```

编译器错误J0205

Cannot use non-final local variable 'identifier' from a different method

编译器检测到在一个方法里引用了一个局部变量，而这个局部变量没有声明为最终变量。如果在一个局部块或方法声明中定义一个内部类，而这个内部类试图引用一个在它作用域之外定义的参数或局部变量时，可能会出现这种错误。

下面举例说明这种错误：

```
public class Simple{
    void method1(int var1){
        class InnerClass{
            boolean getVar1(){
                return(var1 == 1);
                /*error cannot reference local variable of
                  method from within inner class. */
            }
        }
    }
}
```

编译器错误J0206

Cannot assign a second value to blank final variable 'identifier'

编译器检测到试图多次把一个值赋值到final变量。在一个构造器或初始化中多次初始化一个空的final变量时，通常会出现这种错误。检查错误信息中提到的final变量的双重初始化，重新编译。

下面举例说明这种错误：

```
public class Simple{
    final int var1;
     {
        var1 = 10;
        // Do other initializations here
        var1 = 20; //error: duplicate assignment
    }
}
```

编译器错误J0207

Cannot assign blank final variable 'identifier' in a loop

编译器检测到在程序控制循环作用域内给一个final变量赋值。一个final变量只能一次赋值，所以在一个循环内不能被初始化。删除错误信息中

指出的循环外 final 变量的初始化，确保它只初始化了一次。
下面举例说明这种错误：

```
public class Simple{
    final int x;
     public Simple(){
        for (int z=0; z<10; z++){
            x = z;/*error: cannot assign final variable in loop */
        }
    }
}
```

编译器错误 J0208

Constructor or instance initializer must assign a value to blank final variable 'identifier'

编译器检测到说明了 final 变量，但在初始化或构造器中从来都没有给它赋值。为使一个变量正确声明为 final，必须给它赋值。
下面举例说明这种错误：

```
public class Simple {
    final int x; // error: final variables must be assigned a value
    final int z=10;  // This is OK
```

```
}
```

编译器错误J0209

Expected '='

编译器检测到在注释标记（如：@com,@security,@dll）属性中缺少等号。当在注释标记声明的属性中缺少等号时，通常会出现这种错误。如果在编译器计算时，另一个符号或字符隐藏了等号，也会出现这种错误。确保所有注释标记属性都使用了正确的等号，重新编译。
下面举例说明这种错误：

```
//**@com.interface(iid 31415926-5358-9793-2384-612345678901)*/
//error: missing equal sign in 'iid' parameter
interface Itest {
    // do something meaningful here
}
```

编译器错误J0210

Expected '.'

编译器检测到指定的注释标记声明（如@com,@dll）在注释标记(如@com.method)后面缺少句点。如果在编译器计算时，另一个符号或字符

隐藏了句点，也会出现这种错误。
确保注释标记声明中在注释标记后面包含了句点，重新编译。
下面举例说明这种错误：

```
/**@com class */
//error: missing '.' from @com statement
public class Simple {
    // Do something meaningful here
}
```

编译器错误J0214

Invalid GUID specified

编译器检测到需要GUID的@com属性具有非法的GUID条目。导致这种错误的原因是在所输入的GUID中有句法错误。检查GUID的句法，重新编译。

编译器错误J0215

Syntax error in @com declaration

在指定的@com声明中发现句法错误。这往往是不正确地键入声明引起的。检查@com语句的句法，重新编译。

下面举例说明这种错误：

```
/**@com interface(iid = 31415926-5358-9793-2384-612345678910)
// error: The word 'inteface' is misspelled.
interface ITest {
    // Do something here
}
```

编译器错误J0216

@com attribute 'identifier' on 'identifier' is illegal in this context

编译器检测到在一个@com 声明中给属性赋的值是违法的。当指定一个属性，但是利用这个属性所需要的其他指定属性被指定时，往往会出现这种错误。如果在一个@com 声明的错误位置指定属性时，也会出现这种错误。

编译器错误J0217

@com attribute 'identifier' was not specified for 'identifier' but is required in this context

编译器检测到缺少一个属性（错误中显示的@com 声明），而这个属性

正是声明这个类型所需要的。每个@com 声明类型都有所需要的属性。错误信息中列出的有关@com 声明所需要属性的更详细资料，参见有关资料的@com 部分。
下面举例说明这种错误：

```
/**@com .class() */
//error: must specify classid for @com.class
public class Simple {
    // Do something here
}
```

编译器错误J0218

@com attribute 'identifier' on 'identifier' has an invalid value

编译器检测到错误信息中指定的值不是错误中指定属性的正确类型，或者这个属性的合法作用域外。检查赋值到指定属性的值，重新编译。
下面举例说明这种错误：

```
/**@com.class(classid=911CAED0-2957-11d1-A55E-00A0C90F26EE) */
      class Simple {
      /**@com .parameters([type=CUSTOM,customMarsha1="foo.bar",
          customMarsha1Flags=5] i) */
          //error: customMarshalFlags cannot be set higher than 3
```

```
    native void method1(Object i);
}
```

编译器错误J0219

An @com attribute cannot be placed on member 'identifier' unless the containing class or interface also has an @com attribute

编译器检测到一个放在类或接口成员上的@com 声明，但是没有使用@com 声明来声明这个类或接口。确保包含错误中指定方法的类或接口定义了一个合法的@com 声明，重新编译。

下面举例说明这种错误：

```
interface Simple {
    /**@com.method(dispid=777);   */
    public void method1();
    /*error: interface does not have an @com comment tag assigned*/
}
```

编译器错误J0220

An @com attribute cannot be placed on static member 'identifier'

编译器检测到在类的静态成员上放置了一个@com 声明。不能通过 COM

放置静态成员。从方法或字段中删除 static 关键字，重新编译。
下面举例说明这种错误：

```
/*@com.class(classid = 31415926-5358-9793-2384-612345678910)
public class Simple {
    /**@com.method(dispid=777);   */
    static void method1() {}
    // error: cannot expose static method via COM
}
```

编译器错误 J0221

The @com attribute on member 'identifier' cannot be used in this type of class or interface

编译器检测到放在类或接口成员上的@com 声明，但是由于用类或接口使用的@com 属性类型是非法的，所以这个@com 声明不允许使用。这种错误可能在下列情况下出现：
在一个应用@com.class 或@com.interface 声明的类或接口成员中使@com.structmap 声明。
在一个应用@com.struct 声明的类成员中使用@com.method 声明。
下面举例说明这种错误：

```
/**@com.struct() */
```

```
class Simple {
    /**@com.method(dispid=777) */
    public native void method1();
    /*error: cannot use @com.method inside of @com.struct*/
}
```

编译器错误J0222

@com attribute cannot be placed on method 'identifier'——it must be declared 'native' or be in an interface

编译器检测到用@com.method 或@com.parameters属性指定的方法，但是没有使用 native 修饰符声明这个方法，或者没有在一个接口定义中声明这个方法。对于通过 COM 接口公布的方法，必须在一个接口声明中声明它们，或者在一个类声明中声明为 native。

下面举例说明这种错误：

```
public class Simple {

    /* @com.method(dispid=306);*/
    public void method1() {}
    /*error: method must be declared 'native' or declared in an interface*/
}
```

编译器错误J0223

The @com.parameters declaration on member 'identifier' has the wrong number of parameters

编译器检测到@com.parameters属性（错误信息中指定的）与通过COM公布的方法有不同的参数个数。确保这个方法的声明和@com.parameters属性都有正确的参数个数，重新编译。

下面举例说明这种错误：

```
//**@com.interface(iid= 31415926-5358-9793-2384-612345678910) */
interface Itest {
    /** @com.method(dispid=306)
        @com.parameters([type=BOOLEAN] var1,var2,var3)
    */
    public void method1 (boolean var1,int var2);
    /*error: extra parameter added to com.parameters declaration*/
}
```

编译器错误J0224

'return' must be the last item in an @com.parameters declaration

为一个方法声明@com.parameters时，return参数必须是表里的最后一个

参数。编译器检测到 return 参数在参数表的其他位置。检查错误信息中指出的@com.parameters 属性中返回参数的位置，做适当的修改，重新编译。
下面举例说明这种错误：

```
/** @com.class(classid=911CAED0-2957-11d1-A55E-00A0C90F26EE) */
class Simple {
    /**@com .parameters([type=I4] return, [type=CUSTOM,
        customMarsha1="foo.bar", customMarsha1Flags=3] i) */
        //error: 'return' cannot be defined as the first parameter
    native int method1 (Object i);
}
```

编译器错误 J0225

An @COM.'identifier' declaration is illegal for this type of item

编译器检测到为错误项类型定义的@COM 声明。这种错误通常是由于修改代码产生的，但是它不属于@COM 声明的句法或位置问题。确保代码中在指定的项上使用了正确的@COM 声明。
下面举例说明这种错误：

```
/**@com.interface(iid= 31415926-5358-9793-2384-612345678901, dual) */
interface Itest {
```

```
    /**@com.struct()*/
    public int method1();
    // error: wrong type of @com declaration applied
}
```

编译器错误J0226

The @com declared type of 'identifier' is illsgal for a dispatch or dual interface

编译器检测到一个@com接口声明为dual或 dispatch接口，具有一个包含 @com.parameters声明的成员，而这个@com.parameters声明使用错误的type。

当接口声明为dual或 dispatch接口时，某些@com.parameters类型不允许在在接口里定义。下面列表显示对于type来说哪些值是非法的：

- FIXEDARRAY
- SYSFIXEDSTRING
- I8
- U8
- STRUCT
- CUSTOM ,CUSTOMBYREF/CUSTOMBYVAL
- PTR(除了 VARIANT以外的 PTR)
- 以上列出值的任何数组

下面举例说明这种错误：

```
/**@com.interface(iid= 31415926-5358-9793-2384-612345678901, dual) */
interface Itest {
    /**@com.parameters([in,out] n, [type = I8] j]); */
    /*error: cannot use 'I8' as a type in a dual interface*/
    public void method1 (int n, int j);
}
```

编译器错误J0227

It is impossible for an expression of type 'identifier' to be an instance of 'identifier'

编译器检测到使用 instanceof 运算符进行两个类的比较，可能永远也不会彼此成为实例。为正确使用 instanceof 运算符，必须比较有共同类系统的两个类。改变包含 instanceof 运算符的表达式，以便为它的比较使用一个相关类和实例，或者删除这个表达式，重新编译。

下面举例说明这种错误：

```
class Simple1{

    // do something meaningful here
}
```

```
class Simple2{
    // do something meaningful here
}
class CompClasses{
    Simple1 x = new Simple1();
    public static void main(String args[]){
     if(smp.x instanceof Simple2){
      /*error: non related classes cannot be instances of each other*/
    }
}
```

下面举例说明正确使用 instancesof 运算符确保一个类是否是一个指定类的实例。

```
class Simple1 extends Simple2{
    // do something meaningful here
}
class Simple2{
    // do something meaningful here
}
class CompClasses{
    Simple1 x = new Simple1();
```

```
    public static void main(String args[]){
      if(smp.x instanceof Simple2){
        // This is OK since 'Simple1' is a subclass of 'Simple2'
      }
}
```

编译器错误J0228

Syntax error in @dll declaration

编译器检测到在一个@dll注释标记中有一个句法错误。当错误地键入一个@dll注释标记时，通常会出现这种错误。检查语句的句法错误，重新编译。

下面举例说明这种错误：

```
public class Simple {
    /**@dll.import(kernel32, ansi)*/
    /*error: missing quotes around 'kernel32' */
    public static native boolean GetComputerName（StringBuffer s， int[]cb）;
}
```

编译器错误J0229

Expected string constant

编译器检测到注释标记声明（如@com,@dll,或@security）的字符串常数参数缺少字符串常数，或者输入的字符串常数不正确。当传递到属性的值周围缺少相匹配的引用时，很可能会出现这种错误。检查注释标记声明的属性，重新编译。

下面举例说明这种错误：

```
/**@com.interface(iid== 31415926-5358-9793-2384-612345678901, dual) */
interface ITest {
    /**@com.method(name=method();*/
    //error: missing quotes around 'method1'
    public void method1();
}
```

编译器错误J0230

Class or interface name 'identifier' conflicts with import 'identifier'

指定的类或接口与导入的一个类冲突。导致这种错误的原因可能是声明一个已经在Java API中声明的类或接口，并且试图把那个API类或接口导入到源文件中。

下面举例说明这种错误：

```
import java.lang.Cloneable;

class Cloneable { /*error: Cloneable has already been imported */
    // class implementation
}
```

编译器错误J0231

Expression statement must be assignment, method call, increment, decrement, or "new"

编译器检测到一个非法的表达式语句。一个表达式语句是可以放在源代码所属行上的语句。下面是一些合法的表达式语句的例子。

```
m_cars.changeColor(); // method calls
int x=y+z;// Assigment statements
j++;   // Increment statement
m_tempVar1+=3;
new Simple ();
```

下面是一些非法的表达式语句的例子。

```
1+2; // error: No assignment statement
j+k-method1(); // a method call but not a valid assigment statement
```

```
var1 = = var2 ; /*Comparison statements should be contained in flow control statements */
```

编译器错误J0232

Expected '{' or ';'

编译器检测到在一个类或接口中使用方法声明的错误。如果一个接口的方法声明在声明的末尾缺少分号或类声明缺少开始的标志‘{’时，通常会出现这种错误。检查缺少分号或缺少开始标志的指定类或接口的声明，重新编译。

下面举例说明这种错误：

```
interface ISimple {
    public void method1()
    // error: missing semicolon to end declaration
}

public class Simple {
    public void method1();
    // error: missing opening brace
}
```

编译器错误J0233

Catch clause is unreachable; exception 'identifier' is never thrown in the corresponding try block

编译器检测到指定的catch字句永远不会实现，因为相应的try语句块永远不会发出这个catch语句的异常类型。需要用catch语句捕捉异常或引出try语句块能发出的异常。可以修改这个catch语句使它不捕捉这个指定的异常类型，而是捕捉基本异常类型，这样就可以避免这种错误。修改catch语句要捕捉的异常类，重新编译。

下面举例说明这种错误：

```
class Simple {
    void method1() {
        int I=0;
        try {
            I=I+1; //This try block has nothing to do with clones
        }
        catch (CloneNotSupportedException c){
            // error: The exception type can never be caused by try block
        }
    }
}
```

下面举例说明如何通过使用基本 Exception 类避免这种错误：

```
class Simple {
    void method1() {
        int I=0;
        try {
            I=I+1; // This try block has nothing to do with clones
        }
        catch (Exception e){
            // This is OK since all exceptions must derive from this class
        }
    }
}
```

编译器错误 J0234

'identifier' is not a field in class 'identifier'

编译器检测到一个引用的字段，但是这个字段在指定的类里不存在。当错误地键入字段或所引用的字段意味着在不同类里引用字段。确保在指定的类里存在这个字段，引用的字段句法正确，并且重新编译。

下面举例说明这种错误：

```
public class Simple {
```

```
    int var1;
    public Simple (){
        this.var=10;
        // error: 'var' is not a field in this class
    }
}
```

编译器错误J0235

'identifier' is not a method in class 'identifier'

编译器检测到一个方法调用，但是在指定的类里不存在这个方法。当不正确的调用方法调用，或方法调用意味着在不同类里引用一个方法时，通常出现此错误。确保在指定的类里存在这个方法，方法调用的句法正确，并且重新编译。

下面举例说明这种错误：

```
class Simple{
    public void method1(){
        // do something here
    }
}
class Simple2{
```

```
    public void method1(){
        Simple smp = new Simple();
        smp.method2();
        // error: 'method2' does not exist in the class 'Simple'
    }
}
```

编译器错误J0236

'identifier' is not a nested class or interface in class 'identifier'

编译器检测到引用一个嵌套类或接口，但是在指定的类中不存在这个嵌套类或接口。出现这种错误的原因是在引用一个嵌套类或接口时，有语法错误。检查引用内部类的语法错误，并确保在指定的类里存在这个类或接口，重新编译。

下面举例说明这种错误：

```
class NotSimple{
}

public class Simple{
    void method1(){
        NotSimple nt = new NotSimple();
        Object o = nt.new InnewClass();
```

```
        // error: 'InnerClass' is not an inner class of 'NotSimple'
    }
}
```

编译器错误J0237

'identifier' is not a field or nested class in class 'identifier'

编译器检测到引用一个嵌套类或字段，但是在指定的类中不存在这个嵌套类或字段。出现这种错误的原因是在引用一个嵌套类或字段时，有语法错误。如果在内部类中引用一个字段，但是在内部类定义中不存在这个字段时，也会出现这种错误。确保存在这个内部类或字段，重新编译。下面举例说明这种错误：

```
class Simple1{
    int var1;
}

class Simple2{
    static class InnerClass{
        int var1;
    }
    static Simple1 smp;
}
```

```
public class Simple{
    void method1(){
        int x, y;
            x = Simple2.innerclass.var1; /*error: 'innerclass' is not name of
                                        an inner class. 'InnerClass' is */
        y = Simple2.smt.var1; /*error: 'smt' is not the name of a field in
                                'Simple2', Should be 'smp' */
    }
}
```

编译器错误 J0238

Cannot throw exception 'identifier' from field initializer

编译器检测到在一个字段初始化内发出的异常。当一个类的构造器内有可发出的异常而这个类的实例声明并且实例化为另一个类的成员时，通常会出现这种错误。为了解决这个问题，让发出异常的类对象在一个构造器内实例化，以便让它能组合使用 try/catch 捕捉从其他类构造器调用的任何异常。

下面举例说明这种错误：

```
public class Simple{
    public int i;
```

```
    public Simple(boolean var1) throws Exception{
        if (var1 = true)
            i = 0;
        else
            throw new Exception();
    }
}
// This is the incorrect way to instantiate this class
class Simple2{
    Simple smp = new Simple(true);
    /* error: cannot call constructor that throws exception in field initializer*/
}
```

用下面代码替换上例的‘Simple2’类代码声明实例化‘Simple’类实例的正确方式。

```
// This is the correct way to instantiate Simple
class Simple2{
    Simple smp;
     public Simple2(){
        try{
            smp = new Simple(true);}
        catch(Exception e) {}
```

```
    }
}
```

编译器错误 J0239

Static initializer must assign a value to blank final variable 'identifier'

编译器检测到没有使用静态或字段初始化值给一个 static final 变量初始化一个值。为把变量声明为 static 和 final，必须使用静态或字段初始化值设置一个初始值。在构造器内如果一个字段声明为 static 和 final 并且赋值一个值，也会出现这种错误。

下面举例说明这种错误：

```
public class Simple {
    static final int MAX_CONTROL;
}// error: static final variables must have value assigned
```

编译器错误 J0240

Syntax error in @security declaration

编译器检测到指定的 @security 注释标记中有句法错误。当注释标记缺少一个结束

括号或标记的属性不正确时，通常会出现这种错误。确保所指定的 @security 注释标记声明的句法是正确的，重新编译。

下面举例说明这种错误：

```
/**@security()*/    //error: invalid @security syntax
public class Simple {
    // class members defined here
}
```

编译器错误J0241

'@security'can only be specified on a class or interface

编译器检测到@security 注释标记使用到不是类或接口声明的地方。@security注释标记用来为类或接口定义安全设置，不能应用到方法或类的字段或者接口上。

下面举例说明这种错误：

```
public class Simple {
    /**@security(checkDllCalls=on)*/
    public void method1() {
    // error: cannot use @security tag on method
    }
}
```

编译器错误J0242

Cannot make static call to non-static method 'identifier'

编译器检测到通过使用静态方法调用句法‘<classname>.<method>’调用的一个方法，但这个方法不是静态方法。用包含方法的类实例代替类名本身来修改这个方法调用，重新编译。

下面举例说明这种错误：

```
public class Simple{
    public void emthod1(){
        // do something here
    }
}

class NotSimple{
    public void methodx() {
        Simple.method1();
        // error: 'method1' is not a static method
    }
}
```

编译器错误J0243

'identifier' is obsolete; use 'identifier' instead

编译器检测到使用一个格式的注释标记（如@com,@dll,@security)，而所使用的格式已经过时。为避免这种错误，把出现错误的代码行改为错误信息中指定的新格式，并且重新编译。

下面举例说明这种错误：

```
public class Simple {
    /**@dllimport("kernel32",ansi)*/
    /*error: 'dllimport' is no longer a supported format for
        @dll.import declarations */
    public static native boolean GetComputerName(StringBuffer s, int[]cb);
}
```

编译器错误J0244

@conditional allowed only on void-returning methods

编译器检测到一个方法的@conditional 注释标记，而这个方法有一个非空的返回值。把方法声明改为返回 void，或删除@conditional注释标记，重新编译。

下面举例说明这种错误：

```
public class Simple {
    /**@conditional(DEBUG)*/
    public int method1(int x) {
        // error: conditional methods cannot return a value
    return x*2
    }
}
```

编译器错误J0245

Warning treated as error

编译器使用/wx 开关运行，编辑过程中发现一个警告。编译器开关/wx 把所有的警告都视为错误。为使编译器开关/wx 显示这种错误，必须设置足够高的警告级别开关（/w{0-4})以显示这个警告。确定警告的起因，再次进行编译。也可以从编译器选项中删除这个开关，并且再次进行编译。

编译器错误J0246

Invalid token on a #directive

编译器发现一个非法标识用于一个条件编译表达式。当一个标识缺少类型或使用的标识非法时，往往会出现这种错误。确保错误中指出条件编

译伪指令正确，并且再次进行编译。
下面举例说明这种错误：

```
public class Simple{
    public void method1(){
        #iff DEBUG //error: 'iff' is not a valid token
            System.out.println("Do something meaningful here");
        #endif
    }
}
```

编译器错误 J0247

#elif without mathing #if

编译器检测到一个#elif 条件编译伪指令，但是没有检测到相匹配的#if 伪指令。确保有一个合法的#if 条件编译伪指令匹配#elif 伪指令，再次进行编译。
下面举例说明这种错误：

```
public class Simple {
    public void method1() {
        //error: need to have '#if' before '#elif'
        #elif DEBUG
```

```
            System.out.printl("Do something meaningful here");
        #endif
    }
}
```

编译器错误J0248

#endif without matching #if

编译器检测到一个#endif 条件编译伪指令，但是没有检测到相匹配的#if 伪指令。这种错误的出现往往是因为提供了额外的#endif 伪指令。当删除或注释掉一个#if 条件编译伪指令，但没有删除相匹配的#endif 伪指令时，也会出现这种错误。确保具有完全匹配的伪指令，再次进行编译。下面举例说明这种错误：

```
public class Simple{
    public void method1(){
        // #if SIMPLE
        // do something here
        #if DEBUG
           // do something here
           #if WIN95
              // do something here
           #endif
```

```
        #endif
        #endif // error: extra '#endif' directive not permitted
    }
}
```

编译器错误J0249

#else without matching #if

编译器检测到一个#else 条件编译伪指令，但是没有检测到相匹配的#if伪指令。这种错误的出现往往是因为提供了额外的#else 伪指令。当删除或注释掉一个#if 条件编译伪指令，但没有删除相匹配的#else 伪指令时，也会出现这种错误。确保具有完全匹配的伪指令，再次进行编译。
下面举例说明这种错误：

```
public class Simple {
    public void method1() {
        // Do something here
        #else /*error: this '#else' directive has no matching '#if' directive */
    }
}
```

编译器错误J0250

Already had an #else

编译器检测到在一个#if 伪指令块里有双重的#else 条件编译伪指令，一个#if 伪指令只能有一个相匹配的#else 伪指令。确保在错误出现的地方只有一个#else 条件编译伪指令匹配#if 伪指令，再次进行编译。

下面举例说明这种错误：

```
public class Simple{
    #if A
        #if B
        void method1{
        // do something here
        }
    #else
        void method2{
        // do something here
    #else // error: extra '#else' directive supplied
    #endif
}
```

编译器错误J0251

Unexpected EOF while looking for #endif

编译器检测到一个#if条件编译伪指令，但是没有发现相匹配的#endif伪指令，而被#if伪指令使用的符号也没有定义。当使用了一个#if条件编译伪指令，但是没有定义相匹配的#endif伪指令来关闭这个条件编译块时，往往会出现这种错误。确保有一个#endif伪指令与错误中指定的#if伪指令相匹配，再次进行编译。

下面举例说明这种错误：

```
public class Simple {
    void method1() {
        #if DEBUG // debug has not been defined
            System.out.println("Do something here");
        // error: '#endif' not specified for the '#if' directive
    }
}
```

编译器错误J0252

#if nested too deeply

编译器检测到嵌套#if条件编译块嵌套太深了。#if条件编译块的嵌套极

限是 64 层。确保#if 块的嵌套在极限作用域内，再次进行编译。

编译器错误 J0253

Cannot have #define/#undef after source

编译器检测到在其他 Java 源代码之前有一个#define 或#undef 条件编译伪指令。#define 或#undef 伪指令必须放在其他 Java 源代码之前（其他条件编译伪指令和源注释除外）。

把错误中指出的#define 或#undef 伪指令移到文件中所有 Java 源代码之前，再次进行编译。

下面举例说明这种错误：

```
package boxes;

#define DEBUG
// error: '#define' must occur before the 'package' statement

public class Simple {
}
```

编译器错误 J0254

Cannot change predefined symbol

编译器检测到试图在 Java 中已定义的符号上使用#define 或#undef 伪指令。当已为条件编译定义的符号如 true 或 false，又使用#define 或#undef 条件编译伪指令重新定义时，往往会重新这种错误：删除或改变错误中指出的#define 或#undef 伪指令，再次进行编译。
下面举例说明这种错误：

```
#undef false //error: cannot undefine the 'false' symbol
#define true // error: cannot undefine the predefined 'true' symbol
public class Simple{
    // Do something here
}
```

编译器错误 J0255

Expected #endif

编译器检测到一个#if 条件编译伪指令，但是没有发现相匹配的#endif 伪指令；可是定义了#if 伪指令使用的符号。当一个使用了一个#if 条件编译伪指令，但是没有定义一个相匹配的#endif 伪指令来关闭这个条件编译块而想要编译代码时，往往会出现这种错误。确保有一个‘#endif’伪指令与错误中指出的‘#if’伪指令相匹配，再次进行编译。
下面举例说明这种错误：

```
#define DEBUG
```

```
public class Simple {
    void method1() {
      #if DEBUG
          System.out.println("Do somthing here");
    //error: '#endif' has not been specified for the '#if' directive
    }
}
```

编译器错误 J0256

Expected 'class','interface',or 'delegate'

编译器希望在相应的声明中使用关键字 class, interface 或 delegate。当在一个 class, interface 或 delegate 声明中漏掉关键字时，往往会出现这种错误。可能引起这种错误的另外一个原因是大括号不匹配。

> **注意：** 在项目中如果启用 Microsoft Extensions，则只会出现这种错误。否则，就会显示编译器错误 J0020。

下面举例说明这种错误：

```
public Simple {//error: missing the 'class' keyword
    // Do something here
}
```

这个例子说明由大括号不匹配引起这种错误：

```
public class Simple {
// do something meaningful
}}    // error: additional '}' is not permitted
```

编译器错误 J0257

Delegate cannot be initialized with static method 'identifier'

编译器检测到一个 delegate 实例，但作为参数传递的方法引用是一个 static 方法。当实例化一个 delegate 时，必须把非静态方法作为参数来传递。确保定义 delegate 要使用的方法是一个非静态方法，再次进行编译。下面举例说明这种错误：

```
delegate int MyDelegate (it var1, String var2)throws Exception;
public class Simple{
    public static int method1(int var1, String var2) throws Exception{
      // do something here
      return var1;
   }
    public static void main(String args[]){
```

```
        Simple smp = new Simple();
        MyDelegate md = new MyDelegate(smp.method1);
        /*error: the method passed as a parameter to the delegate is a static method */
    }
}
```

编译器错误J0258

Cannot declare delegate in inner class 'identifier'

编译器检测到试图在一个内部类里声明一个 delegate。内部类定义不支持在它们内部声明的 delegate。从内部类声明中删除 delegate 声明，再次进行编译。

下面举例说明这种错误：

```
public class Simple{
    // Do something meaningful here
    class InnerClass{
    // Do something meaningful here
       delegate int MyDelegate (int var1, String var2);
          /* error: delegates cannot be declared inside inner classes */
    }
}
```

编译器错误J0259

Exception 'identifier' from method 'identifier' is not a subclass of any exceptions declared thrown by delegate 'identifier'

编译器检测到通过一个发出异常方法初始化的一个 delegate，这个发出异常与 delegate 声明的异常设置是不相容的。每个通过方法使用的发出异常必须与 delegate 发出的异常有相同的类，或是 delegate 发出的异常的子类。改变通过 delegate 使用的发出异常设置，或者改变通过方法使用的异常发出设置，再次进行编译。

下面举例说明这种错误：

```
delegate void SimpleDelegate(String var1) throws Exception;

public class Simple{
    public void method1(String var1) throws Throwable{
        // do something here
    }
    public static void main (String args[]){
            Simple smp = new Simple();
        SimpleDelegate del1 = new SimpleDelegate(smp.method1);
        /* error: the method reference argument throws an exception that is not the same or a
                subclass of the exception declared by the delegate */
    }
```

```
}
```

编译器错误J0260

Cannot declare an interface method to be 'protected' or 'private'

编译器检测到带有声明为 protected 或 private 的方法的一个接口。接口方法必须声明为 public 或不带访问修饰符（默认情况下）。改变错误中指出的接口方法声明，使它不声明为 public 或 private，再次进行编译。下面举例说明错误：

```
interface ISimple{
    public void method1(); // this is OK!
    void method2(); // this is OK! Declared as 'default' access
    protected void method3(); /*error: cannot declare interface method as protected */
    private void method4(); /*error: cannot declare interface method as private */
}
```

编译器错误J0261

An explicit enclosing instance of class 'identifier' is needed to instantiate inner class 'identifier'

编译器检测到试图在没有指定实例的类内创建一个内部类实例。当试图

在一个静态方法内像实例化外部类一样实例化一个内部类时，往往会出现这种错误。为创建一个内部类实例，必须使用它的现有外部类实例。创建一个类作用域内的实例，并且把用它实例化内部类，再次进行编译。下面举例说明这种错误：

```
public class Simple{
    class InnerClass{
        // do something here
    }
     public static void main(String args[]){
        InnerClass inc = new InnerClass();
        /*error: an instance of the outer class is required to instantiate its inner class */
    }
}
```

下面举例说明如何使用外部类实例创建内部类实例：

```
public class Simple{
    class InnerClass{
        // Do something here
    }
    public static void main(String args[]){
        Simple smp = new Simple();
        /*use the instance of 'Simple' to create an instance of its inner class*/
```

```
        InnerClass inc = smp.new InnerClass();
    }
}
```

编译器错误J0262

An explicit enclosing instance of class 'identifier' is needed to call constructor of superclass 'identifier'

编译器检测到试图通过没有超类实例的派生类调用它超类的构造器。这种错误的出现往往是因为超类也是内部类。当一个派生类被实例化时，它的超类实例也创建了（并且超类构造器被引用）。为创建一个内部类实例，必须使用外部类来实例化内部类。
下面举例说明这种错误：

```
class OuterClass{
    class InnerClass{
    // do something here
    }
}
public class Simple extends OuterClass. InnerClass{
    // do something here
}
```

```
/*error: superclass 'InnerClass' cannot be instantiated without its
   outer class 'OuterClass' being instantiated */
```

下面举例说明如何实例化一个从内部类派生的类：

```
class OuterClass{
    class InnerClass{
    // do something here
    }
}
public class Simpole extends OuterClass.InnerClass{
    Simple(){
        /*this line creates an instance of the outer class and calls
           the superclass (the inner class) constructor */
        new OuterClass(). super();
    }
}
```

编译器错误J0264

Array cannot have a dimension

编译器检测到初始化数组不正确。在创建实例之前指定一个数组维数的大小时，往往会出现这种错误。在适当的地方给定数组维数，但是又提

供一个初始化表来设置数组值时，也会出现这种错误（当使用一个初始化表时，不能指定数组维数的大小）。改正错误中指定的数组初始化，并且再次进行编译。
下面举例说明这种错误：

```
public class Simple{
    public void method1(){
        String[3] var1; /*error: cannot specify array size here since
                              array is not instantiated yet */
        String [] var2 = new String [3]; // this is OK!
          boolean var3 = new boolean[3]{true,false, true};
        /*error: cannot specify an array size for the dimension when using
                  an initialization list */
        boolean var4 = new boolean[]{true, false, true}; // this is OK!
    }
}
```

编译器错误J0265

@dll attribute be placed on method 'identifier'——it must be declared 'native'

编译器检测到@dll注释标记放在了一个没有使用native修饰符声明的方法上。如果定义了@dll注释标记但删除或注释掉了它的方法，而类中的其他方法可能有相适应的注释标记时，往往会出现这种错误。确保使用@dll注释标记声明的方法声明为native或删除@dll注释标记，再次进行编译。
下面举例说明这种错误：

```
public class Simple{
    /** @dll.import("USER32") */
     // private native static int MessageBix(int hwndOwner, String text, String title, int fuStyle);
     /*error: the @dll comment tag cannot applied to 'method1' which is a
            non-native method instead of the intended native method*/
    public void method1(){
        // do something here
    }
}
```

编译器错误J0266

Compilation canceled by user
用户取消编译。这种错误简单地通知用户请求取消编译成功。

编译器错误J0267

A delegate cannot be 'identifier'

编译器检测到试图使用非法的修饰符声明一个 delegate。只有修饰符 public 和 final 可以用于类外部的 delegate 声明。类内部的 delegate 声明可以应用修饰符 public, final, protected 和 private。确保 delegate 声明没有使用非法修饰符，再次进行编译。

> **注意**：虽然可以在 delegate 声明中使用修饰符 final，但这是没有必要的，因为 delegate 隐含的就是 final。

下面举例说明这种错误：

```
private delegate void Simple(int var1,int var2);
// error: cannot declare a delegate as 'private'
```

编译器错误J0268

A delegate cannot be 'identifier' ang 'identifier'

编译器检测到在一个类内试图使用不允许的访问修饰符组合来声明 delegate。当使用两个访问修饰符声明一个 delegate 时，往往会出现这种错误。删除应用到指定 delegate 声明中的多余访问修饰符，再次进行编

译。
下面举例说明这种错误：

```
public class Simple {
    public private delegate void SimpleDelegate();
    //error: cannot declare a delegate to be 'public' and 'private'
}
```

编译器错误J0269

Ambiguous name:inherited 'identifier' and outer scope identifier'—an explicit 'this' qualifier is required

编译器检测到从内部类里引用在外部类和超类里都定义的变量或方法。编译器不能确保使用哪个变量或方法。可以通过使用class.this.name引用外部类变量或方法，这里的class是希望引用的变量和方法的外部类名字，name是变量或方法名字。为引用这个超类变量或方法,必须在引用之前使用关键字this super。
下面举例说明这种错误：

```
class NotSimple{
    int var1 = 20;
}
public class Simple{
```

```
    int var1 = 10;
    class InnerClass extends NotSimple{
        int var2 = var1;
        // error: cannot determine which 'var1' to use
    }
}
```

下面举例说明如何在外部类和超类变量之间解析名字的二义性：

```
class NotSimple{
    int var1 = 20;
}
public class Simple{
    int var1 = 10;
    class InnerClass extends NotSimple{
        int var2 = Simple.this.var1;
        // this is OK and references the outer class variable
        int var3 = super.var1;
        // this is OK and references the superclass variable
    }
}
```

编译器错误J0270

Ambiguous reference to 'identifier' in interface 'identifier' and class 'identifier'

编译器检测到对字段的二义性引用。当一个类的超类与这个类实现的接口有相同的字段声明时，往往会出现这种错误。编译器不能确定使用这个字段的哪一个实例。确保超类和实现的接口没有相同定义的字段，再次进行编译。

下面举例说明这种错误：

```
public class Simple extends SuperClass implements SuperInterface {
    public static void main (String args[]) {
        float x=var1; /*error: cannot determine which instance of 'var1' to use */
    }
}

class SuperClass {
    float var1;
}

interface SuperInterface {
    float var1=6.0f;
}
```

编译器错误 J0271

Expected 'delegate'

编译器检测到 multicast 修饰符但没有检测到关键字 delegate。multicast 修饰符只能连同关键字 delegate 一起使用来创建多点委托。确保在声明中有错误消息中指出的 delegate 关键字，再次进行编译。

下面举例说明这种错误：

```
public multicast void SimpleDelegate(String x, int y);
// error: the 'delegate' keyword is missing after the 'multicast' modifier
public class Simple {
    // Do something here
}
```

编译器错误 J0272

A multicast delegate cannot return a value

编译器检测到与为 multicast delegate 声明的 void 不同的返回类型。虽然 delegate 可以返回一个与 void 不同的值，但 multicast delegate 不能返回与 void 不同的值。确保错误中指定的 multicast delegate 声明返回 void，再次进行编译。

下面举例说明这种错误：

```
multicast delegate int SimpleDelegate(int x, int y);
// error: cannot declare multicast delegate with non-void return type
public class Simple{
    public int mehtod1(int x, int y){
        // Do something here
    }
     public static void main (String args[]) {
        Simple smp = new Simple;
        SimpleDelegate sd = new SimpleDelegate(smp.method1);
    }
}
```

编译器错误 J0273

Cannot cast to 'void'

编译器检测到试图把方法调用的返回值设置为 void。在 Java 语言中不允许把方法调用设为 void。删除为错误中指出的方法调用设置的 void，再次进行编译。

下面举例说明这种错误：

```
public class Simple{
    public void method1(){
```

```
        // Do something meaningful here
    }
    public static void main (String args[]){
        Simple smp = new Simple();
        (void) method1();
        // error: cannot cast method's return value to 'void'.
    }
}
```

编译器错误J0274

Expected 'identifier' within single comment preceding class 'identifier'

编译器检测到错误信息中指出的类，使用@com.typeinfo或@com.transaction注释标记声明，而在同一个注释里没有相应的@com.register注释标记。当声明一个@com.register注释标记，但是把它放在独立的注释里时，往往会出现这种错误。确保为错误信息中指出的类定义的@com.register注释标记与定义的@com.typeinfo或@com.transaction注释标记在同一个注释里，再次进行编译。

下面举例说明这种错误：

```
/**@com.register(clsid=AE032C46-E6A0-11d0-8C83-00C04FC2AAE7) */
/**@com.typeinfo(attrid=31415926-5358-9793-2384-612345678901,
value="help") */
```

```
public class Simple {
    //error: the '@com.register' tag is ignored since it is in a separate comment*/
}
/**@com.register(clsid=AE032C46-E6A0-11d0-8C83-00C04FC2AAE7)
    @com.typeinfo(attrid=31415926-5358-9793-2384-612345678901,
value="help") */
public class Simple2 {
    // This comment is OK. It combines both tags in a single comment.
}
```

编译器错误 J0275

Class or interface name 'identifier' conflicts with package 'identifier'

指定类或接口名与软件包名发生冲突。在一个软件包内使用与一个子软件包名相同的名声明类或接口时，可能引起这种错误。删除引起冲突的子软件包或类或接口，再次进行编译。

下面举例说明这种错误：

```
// located in a file called Plastic.java
package Boxes.Dishes;
public class Plastic{
    // do something here
```

```
}

// located in a file called Dishes.java
package Boxes;
public class Dishes{
    // error: the name 'Dishes' is also the name of the subpackage
}
```

编译器错误J0500

#error 'text'

当程序中有#error条件编译伪指令时，产生这种错误信息。下面举例说明这种错误：

```
#define DEBUG
public class Simple {
    public void method1() {
        #if DEBUG
            #error you shoud not be running in debug mode here !
            // error displayed when 'DEBUG' is defined
        #endif
    }
}
```

编译器警告 J5001

Local variable 'identifier' is initialized but never used

编译器检测到在任何类代码中都没有引用的初始化变量。这种信息出现在 3 级警告或较高级的警告里。在类代码中使用变量或删除它们。

下面举例说明这种警告：

```
public class Simple {

    public int method1() {

          int i=1;

        return 1;

        //warning: 'i' is never used

     }

}
```

编译器警告 J5002

Compiler option 'identifier' is not supported

编译器检测到指定一个不支持的命令行操作。检查编译器开关设置，再次进行编译。

编译器警告 J5003

Ignoring unknown compiler option 'identifier'

编译器检测到在 JVC 命令行上指定了一个不认识的操作。当存在输入错误时，通常会出现这种警告。例如，在命令行上指定/W4 将导致这种警告，因为警告级别操作必须使用小写‘w’。

编译器警告 J5004

Missing argument for compiler option 'identifier'

编译器检测到一个合法的命令行操作，没有指定所需要的变量。检查编译器开关设置，再次进行编译。

编译器警告 J5005

Package 'identifier' was already implicitly imported

编译器检测到已经隐含导入了用于软件包的 import 语句，例如 Java.lang。这种信息出现在 1 级警告或较高级别的警告中。
下面举例说明这种警告：

```
import java.lang.*;
//warning: process is already imported in the first import statement
```

```
import java.lang.Process;

public class Simple {
    // Do something meaningful here
}
```

编译器警告J5006

'private protected' not supported,using 'protected'

编译器检测到使用修饰符组合 private protected。现在这个组合已经作废而用 protected 代替了它们。这个信息出现在 1 级警告或较高级别的警告中。

编译器警告J5014

'identifier' has been deprecated by the author of 'identifier'

引用的方法或类标记为“反对”。反对的方法或类被源代码的创建者标记为过时的和可移动的内容。为保持代码稳定，应避免调用被标记为反对的函数。

编译器警告 J5015

The parameter 'identifier' in an @com.parameters declaration does not match the corresponding argument 'identifier'

一个@com.parameters 声明与实现接口的方法不匹配。这种错误表示数据类型不一致或参数位置匹配错误。确保@com.parameters 声明与声明的方法有相同数量的参数、正确的类型和正确的顺序，再次进行编译。

下面举例说明这种警告：

```
/**@com.interface(iid=31415926-5358-9793-2384-612345678901,dual) */
interface ExampleInterface
{
/**
@com.method(dispid=306)
@com.parameters([type=BOOLEAN] when,on)
*/
// error: @com.parameters declaration has parameters in wrong location
public String Method1(boolean on,int when);
}
```

编译器警告 J5016

The 'instanceof' operator will always be true

编译器判定指定的 instanceof 表达式将总是计算为正确。当使用 instanceof 确保一个子类实例是否是一个基类成员或实现接口时，可能会出现这种警告。在这种情况下，虽然使用 instanceof 运算符并没有错误，但它没有特殊用途，因为表达式将总是计算正确。
下面举例说明这种警告：

```
interface I {}
class X {}
class Y extends X implements I {}

public class Simple{
    void f()
    {
        Y y = new Y ();
        X x = new X ();
        Object o;

          // These statements give J5016 warnings...
        if (y instanceof X) {
          // y is of type Y, which extends X, so this is always true
        }
```

```
        if (y instanceof I) {
            // type Y implements I, so this is alway true.
        }
        if (x instanceof Object ) {
            // everything is of type Object
        }
    }
}
```

编译器警告J5018

Class,interface,or package name contains characters not in the ASCII character set

编译器检测到class,interface或package名字使用的字符没有在ASCII字符集里。一些计算机系统可能不支持这个字符。为确保class，interface或package名字可以被其他计算机系统上的虚拟机（VM）解释，从ASCII字符集中把非法字符变为合法字符，再次进行编译。

编译器警告J5019

Public member of COM-exposed class or interface contains characters not in the ASCII character set

编译器检测到用于方法或字段的一个字符没有在 ASCII 字符集里，这种方法或字段可以通过 COM 到达其他语言。其他语言可能不支持这个字符。确保可以从其他语言中访问成员名字，从 ASCII 字符集中把这个字符改为合法字符，再次进行编译。

编译器警告 J5020

Directive 'identifier' ignored— extensions are turned off

编译器检测到在源代码中应用了注释标记，例如 @com 或 @dll，但是在 Microsoft 语言扩展中不允许使用。为使用注释标记或其他 Microsoft 语言扩展，必须使扩展操作可用。

> **警告：** 使用禁止 Microsoft 语言扩展操作，任何利用 Microsoft 语言扩展的代码都不可能正确工作。

下面举例说明这种警告：

```
//Microsoft Language Extensions are 'disabled'
public class Simple {
    /**@dll.import("user32")*/
    public static native int MessageBox(int hWnd,String text,String caption, int type);
    // warning: the 'MessageBox' method may not work correctly
```

编译器警告 J5021

Package 'identifier' should not be defined in directory 'identifier'

编译器检测到在源文件中有 package 语句，但是源文件所在的目录与 package 不匹配。尽管源代码可以编译，但是其他源文件将不能使用 package 名引用在这个源文件中定义的类、接口和委托。

下面举例说明这种错误：

```
// source file resides in c:\files\simple
package boxes; // warning: package 'boxes'does not match directory name 'simple'
public class NotSimple {
    // warning: this class is not accessible using the current package name
}
```

编译器警告 J5022

Reference class file 'filename' may be older than 'filename'

编译器检测到正编译的源文件引用的一个类文件，这个类文件缺少日期或没有找到日期。当所引用类的自动重新编译被禁止时，（使用 ref-编译器选项），通常会出现这种错误。如果正在 Visual J++内编译，在默认情况下 ref-选项已经传送到了编译器。如果任何一个源文件引用的类文件不在项目中，而且这个文件缺少使用源的日期，则需要修改所引用

的类文件。可以编译外部类文件，使它们补上日期，把外部类添加到项目里，也可以增加 ref 选项。增加 ref 编译选项的方法是在 Project Settings 对话框 Compile 选项卡的 Additional Compiler Option 文本框中输入 ref 编译器选项。

编译器警告 J5023

File 'filename' has more than 65535 lines—debug information may be incorrect

编译器试图把调试信息放入包含多于 65535 行代码的源文件中。代码可以编译，但是在这个文件中并不包含应用程序所需要的全部调试代码。缩短源文件的大小，使调试信息可以增加进去，重新编译。

编译器警告 J5024

Package 'identifier' was already imported

编译器检测到在指定的源文件中导入的 package 多于一次。虽然这不会导致编译失败，也应该删除与警告信息中指定的 package 有关的多余 import 语句。

编译器警告 J5500

#warning 'text'

在程序中有#warning 条件编译伪指令时，产生这个警告信息。下面举例说明这种错误：

```
#define DEBUG
public class Simple {
    public void method1(){
        #if DEBUG
            #warning: You are compiling for debug mode
            // warning: displayed when 'DEBUG' is defined
        #endif
    }
}
```

COM注册错误(Visual J++)

在 COM 类注册中，可以使用错误或警告信息提示。下面是许多带有普遍性错误和警告的列表。每个错误或警告包含的信息包括出现错误的可能原因、应用的地点以及如何处理错误或原因的建议。

No registration attribute in class file 'filename'（类文件 **filename** 中无注

册属性）

试图用错误消息中引用的类文件注册，但是这个类文件不包含@com.register 注释标记。确保在类文件中正试图注册的 COM 类有一个合法的@com.register 注释标记。

Cannot create instance of class 'identifier'（不能创建 identifier 类的实例）

为创建一个类型库和注册 COM 类，类必须能被初始化和能够调用标准构造器（这个构造器不包含任何参数）。如果 COM 类的任何一个非独立类不能被定位或者与类相关的标准构造器有错误，可能会出现这种错误。确保与 COM 类相关的非独立类可以被访问，并且类的标准构造器不包含任何错误。

Cannot open filename 'filename'（不能打开 filename 文件名）

在 COM 注册进程中，产生一个临时的类型库。当临时的文件不能拷贝和重命名到最终类型库名时，将会出现这种错误。确保这个文件不是只读的，没有在一个只读目录里，或这个文件没有被其他应用程序所使用。

Unable to create register keys for class 'identifier'（不能创建 identifier 类的注册关键字）

当试图注册 COM 类时，COM 注册进程不能在系统中创建一个关键字。如果 COM 注册进程不能打开指定关键字或不能向注册中写入一个值时，也会出现这种错误。确认系统注册没有被损坏。

Class file 'filename' does not have the extension '.class'（类文件 filename 中没有.class 扩展名）

为注册一个 COM 类，编译类文件必须有.class 扩展名。如果这个被注册为 COM 类的文件不是合法的 Java 类文件，也会出现这种错误。确保这个正试图注册为 COM 类的类文件包含.class 扩展名，而且是一个合法的 Java 类文件。

The internal class name 'identifier' does not match the file or directory name of class file 'filename'——it cannot be loaded（内部类名 identifier 不能匹配类文件 filename 的文件或目录名——不能加载它）

注册的 COM 类与定位的类文件有不同的类名，或者位于目录中的类文件与这个类定义的软件包冲突。确保这个 COM 类名和它定义的软件包与这个类文件的位置和名字相匹配。

Class contain more than one type library id. Only one type library can be created at a time（类包括不止一个类型库 id。一次仅能生成一种类型库）

在 Visual J++项目中，一次只能指定一个类型库。在项目中，当一个 COM 类的@com.register 的注释标记与其他类有不同的类型库 GUID 时，通常会出现这种错误。在一个项目中，所有的 COM 类必须指定相同的类型库。在 Project Properties 对话框的 COM Class 选项卡中，通过清除这个类旁边的复选框来解决这种冲突，应用这种改变，然后重新复选这个类的复选框。这将重新指定这个类为 COM 类。并且指定相同的类型库

GUID。

Base name of class 'identifier' conflicts with another class in the same type library（类 identifier 的基本名与在同一类型库中的另一个类冲突）

发现定义的类已经在类型库里。当软件包里定义的一个 COM 类与不同软件包里的带有相同名字的 COM 类都添加到同一个类型库时，通常会出现这种错误。改变添加到类型库的一个类名，以便使冲突解决。

Windows EXE/COM DLL打包错误(Visual J++)

在 Windows EXE 或 COM DLL 的创建中，可能接收到一个错误消息。下面是可能出现的错误消息列表。每个错误消息还列出了导致这种错误的可能原因、可应用的地点以及如何处理这种错误的建议。

Path too long:'path'（路径太长：'path'）

Windows EXE 或 COM DLL 的路径和文件名的长度超过 256 个字符。缩短路径和文件名的长度，重新编译项目。

No main class file specified（没有指定主类文件）

试图创建一个 Windows EXE, 但是没有指定主类（一个用主方法定义的类）。为创建一个 Windows EXE，必须在 Project Properties 对话框的 Launch 选项卡中指定一个主类。

No class files found（没有找到类文件）

试图创建一个 Windows EXE 或 COM DLL，但是没有发现或没有指定用来创建 EXE 或 DLL 的类文件。确保项目至少包含一个类文件，并且这个类文件在 Project Properties 对话框的 Output Format 选项卡中指定。

Unable to open output file :'filename'（不能打开输出文件：'filename'）

不能打开错误消息中指出的文件。确保这个指定文件不是只读的或没有被其他程序应用。如果这个指定文件位于不允许写的目录中，也可能会出现这种错误。

Unable to update resources in file:'filename'（不能更新文件中的资源：'filename'）

试图把资源信息写到错误消息中指出的资源文件中，但是不能修改这个资源文件。确保在计算机上有足够的可用磁盘空间。

Badly formatted class file:'filename'（类文件格式非法：'filename'）

错误消息中指出的类文件使用了一个非法格式。当类文件遭到损坏或者是使用类扩展的文件，但它不是合法的 Java 文件。删除错误消息中指出的类文件，重新建立项目。

File not found:'filename'（找不到文件：'filename'）

没有发现错误消息中指出的文件。在 Project Properties 对话框的 Output Format 项中，确保这个文件的正确位置。

Unable to load resources from 'filename'（不能从 filename 中加载资源）

不能打开错误消息中指出的资源文件。确保这个资源文件是合法的 Windows 资源文件，并且这个资源文件没有被另外的程序所使用。

Unable to read typelib file 'filename'（不能读取类型库文件 filename）

不能打开错误消息中指出的类型库文件。确保这个类型库文件是合法的类型库文件，并且这个文件没有被另外的程序所使用。

Corrupt registration attribute in class 'identifietr'(类 identifier 中注册属性损坏)

错误消息中所指出类的一个 COM 注册属性有误。当这个指定类的类文件被损坏或是一个非法文件格式时，通常会出现这种错误。删除指定类的类文件，重新建立项目。

Class ‘identifier’ does not have a method of the form ‘static public void main (String[])’（类 identifier 没有 static public void main stirng[]形式的方法）

一个类指定为 Windows EXE 的主类，但这个类不包含一个主方法。主方法为 Windows EXE 提供起点，并且必须被指定。向错误消息中指出的类里添加一个主方法，或者在 Project Properties 对话框的 Launch 选项卡中选择不同的主类，重新建立项目。

Main class name 'identifier' is too long（主类名 identifier 太长）

错误消息中指出的主类名长度超过 256 个字符。缩短这个类名的长度，重新编译项目。

Specified main class 'identifier' not found（指定的主类 identifier 没找到）

创建一个 Windows EXE 文件时，没有发现为 Windows EXE 指定为主类的类。在 Project Properties 对话框的 Launch 选项卡中指定一个主类，但从类文件中把这个主类删除或改名时，可能会出现这种错误。在 Project Properties 对话框的 Launch 选项卡中指定一个新的主类，重新建立项目。

Unable to read from file 'filename'（不能读取文件 filename）

不能打开错误消息中指定的准备读取的文件，确保这个指定文件没有其他程序所使用。

Cannot create a DLL with no COM classes（不能创建无 COM 类的 DLL）

试图创建一个 COM DLL，但是这个项目不包含任何 COM 类。可以通过在 Project Properties 对话框的 COM Classes 选项卡的类表中选择一个类来创建一个类。如果不希望把任何类置为 COM 类，在 Project Properties 对话框的 Output Format 选项卡中修改要创建的软件包类型，并且重新编译。

Failure during auto-registration of 'filename'（在 filename 的自动注册时失败）

一个 COM DLL 在创建后自动注册失败。当系统注册器被损坏时，通常会出现这种错误。

Out of memory（内存溢出）

创建一个 Windows EXE 或 COM DLL 时，没有足够的内存可以使用。关闭当前正运行的其他程序，以提供更多的可用内存，重新建立项目。

附录 B　条件编译

Visual J++提供两个用来在 Java 中条件编译代码的新机制：条件伪指令和条件方法。下表描述了所有这些可用的条件伪指令。要了解一般的情况，见“条件伪指令”一节。要了解句法的描述和条件方法的使用，见“条件方法”一节。这两节都在本附录的后面。

> **注意：**由 Visual J++提供的条件伪指令和条件方法机制是扩充到由 Microsoft 提供的 Java 语言中的。因此，包含这些机制的源代码在使用其他 Java 开发工具时不能适当编译。

条件伪指令	说明
#if	条件包含或排除源代码，取决于它的表达式或标识符的结果值
#elif	用于#if 伪指令的可选项。如果前面的#if 测试失败，#elif 包含或排除源代码，这取决于它自己的表达式或标识符的结果值
#else	用于#if 伪指令的可选项。如果前面的#if 测试失败，跟随#else 伪指令的源代码将被包含
#endif	使用#if 伪指令所需要的。#endif 伪指令关闭一个代码的条件区

#define	定义预处理中的标识符
#undef	取消预处理中标识符的定义
#error	生成开发者定义的编译时的错误信息
#warning	生成开发者定义的编译时的警告信息

#if，#elif，#elif，#else和#endif条件伪指令

#if，#elif，#elif，#else 和#endif 伪指令共同使用，用来进行条件包含或排除源文件编译的源代码行。开始所有的条件区的#if 伪指令，必须有一个关闭条件区的#endif 伪指令相匹配。

语法：

#if <表达式>
包含的代码
#elif <表达式>
包含的代码
#else
包含的代码
#endif

跟随在#if 和#elif 伪代码后面的表达式必须是一个有效的布尔表达式，由布尔运算符组成，并且使用#define 和#undef 编译器伪指令声明的标识符。如果跟在#if 伪指令后面的表达式值为真，紧接在该伪指令后面的行将保留在源文件中。如果该表达式的值为假，则在该伪指令后到#elif，

#else 或#endif 之前的行将从源文件中排除。
在一个条件区中只能包含一个#else 伪指令。只有当与之相对应的#if 伪指令值为假时，跟随在#else 伪指令后面的行才在源文件中保留。
在一个代码的条件区中可能不包含或包含多个#else 伪指令。#elif 伪指令是“else if”的缩写，用于当#if 伪指令已经被赋值，并且该值为假时其随后表达式的赋值。
用于条件编译中的表达式与用于Java语言中的表达式有着同样的限制和行为。例如，所有Java定义的布尔和位运算符都被接受，并且运算符的优先级是同等的。此外，Java语言还定义括号用来强制改变赋值的顺序。
下面是用来说明每一个条件伪指令的例子：

```
#define DEBUG
#undef RETAIL

public class test {

    #if DEBUG
      if (cmdStatus.equals(invokeError))
      {
          // The following line displays a diagnostic message:
          System.out.println("Error: command timed out.");
          // appropriate actions
          // taken here
      }
```

```
    #elif RETAIL
       if (cmdStatus.equals(invokeError))
       {
           // appropriate actions
           // taken here
         }
    #else
       #error Must define DEBUG or RETAIL;
    #endif
}
```

#define条件伪指令

#define 伪指令用来将一个条件标识符包含到源文件中。当定义标识符时，该标识符能够用在该源文件所包含的所有类中。这个伪指令必须放在源文件的开始，在它的前面只能有注释和其他的条件编译伪指令。
根据规定，被#define 伪指令定义的标识符的值将总是为真。

语法

#DEFIND <*标识符*>
在上面显示的标识符长度最长可为 1024 字符。根据规定，标识符由大写字符组成，用来说明它们的用途。
下面的例子用来说明#define 伪指令在 Java 应用程序中的用途：