Python 错误类型完整文档

目录

语法错误

SyntaxError

当 Python 代码存在语法问题时抛出。

```
# 示例
if True
    print("Missing colon")  # 缺少冒号

# 修复
if True:
    print("Missing colon")
```

IndentationError

缩进错误，是 SyntaxError 的子类。

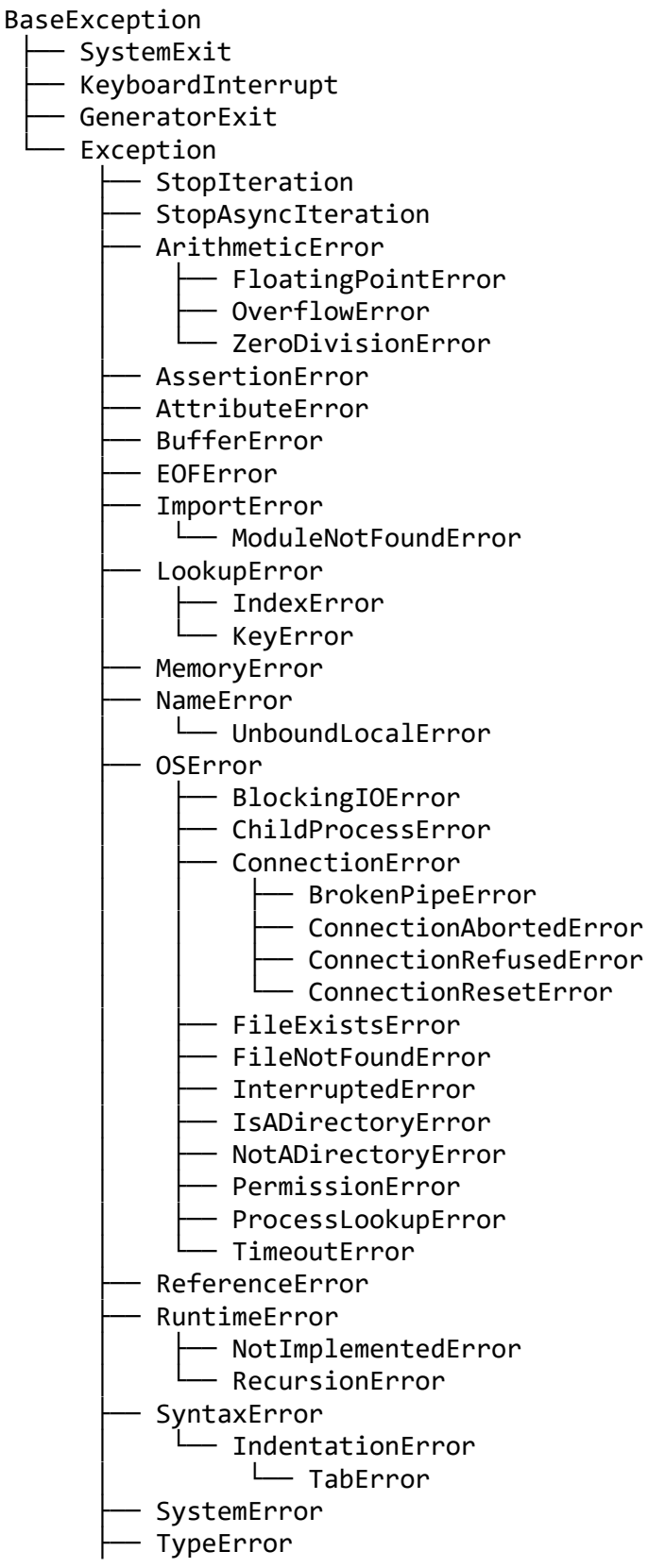
```
# 示例
def func():
print("Bad indentation")  # 缩进错误

# 修复
def func():
    print("Correct indentation")
```

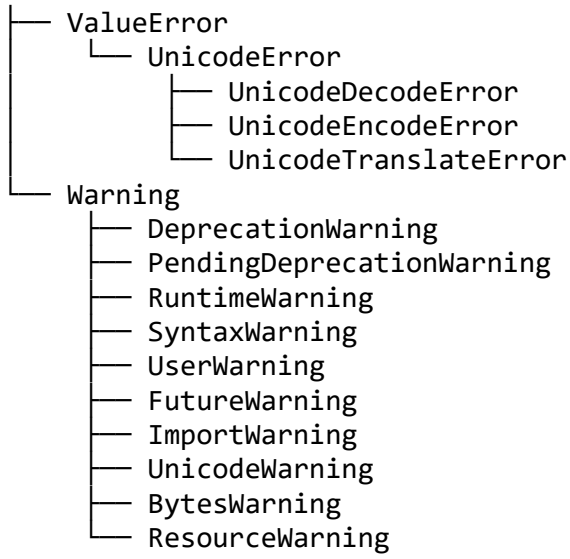
TabError

当代码中混用制表符和空格时抛出。

```
# 示例（文件中混用 tab 和空格）
def func():
    print("Spaces")  # 使用空格
    print("Tab")     # 使用制表符
```

内置异常层次结构


BaseException
SystemExit
KeyboardInterrupt
GeneratorExit
Exception
StopIteration
StopAsyncIteration
ArithmeticError
FloatingPointError
OverflowError
ZeroDivisionError
AssertionError
AttributeError
BufferError
EOFError
ImportError
ModuleNotFoundError
LookupError
IndexError
KeyError
MemoryError
NameError
UnboundLocalError
OSError
BlockingIOError
ChildProcessError
ConnectionError
BrokenPipeError
ConnectionAbortedError
ConnectionRefusedError
ConnectionResetError
FileExistsError
FileNotFoundError
InterruptedError
IsADirectoryError
NotADirectoryError
PermissionError
ProcessLookupError
TimeoutError
ReferenceError
RuntimeError
NotImplementedError
RecursionError
SyntaxError
IndentationError
TabError
SystemError
TypeError



常见内置异常详解

BaseException

所有内置异常的基类。

Exception

所有内置的非系统退出异常的基类。

ArithmeticError

算术错误的基类。

ZeroDivisionError

除以零错误。

```
# 示例
result = 10 / 0
result = 10 // 0
```

OverflowError

数值运算结果太大无法表示。

```
# 示例
import math
math.exp(1000)  # 可能引发 OverflowError
```

FloatingPointError

浮点运算失败（通常不常见，因为 Python 遵循 IEEE754）。

AssertionError

assert 语句失败时抛出。

```
# 示例
x = 5
assert x == 10, "x should be 10"
```

AttributeError

属性引用或赋值失败。

```
# 示例
class MyClass:
    pass

obj = MyClass()
print(obj.non_existent_attr)  # 访问不存在的属性
obj.non_existent_attr = 10    # 这不会报错，但读取会
```

BufferError

缓冲区相关操作无法执行时抛出。

EOFError

input()函数遇到文件结束条件（EOF）。

```
# 示例（在无输入的情况下）
try:
    data = input()
except EOFError:
    print("No input provided")
```

ImportError

导入模块或对象失败。

```
# 示例
import non_existent_module
from math import non_existent_function
```

ModuleNotFoundError

ImportError 的子类，专门用于模块未找到。

```
# 示例
try:
    import non_existent_module
```

```
except ModuleNotFoundError as e:
    print(f"Module not found: {e}")
```

LookupError

查找错误的基类。

IndexError

序列下标超出范围。

```
# 示例
my_list = [1, 2, 3]
print(my_list[10])  # 索引超出范围
```

KeyError

在字典中查找不存在的键。

```
# 示例
my_dict = {'a': 1, 'b': 2}
print(my_dict['c'])  # 键不存在
```

MemoryError

内存不足错误。

```
# 示例
try:
    huge_list = [0] * (10**10)
except MemoryError:
    print("Out of memory")
```

NameError

尝试访问未声明的变量。

```
# 示例
print(undefined_variable)
```

UnboundLocalError

访问未初始化的局部变量。

```
# 示例
def func():
    print(x)  # 在赋值前引用局部变量
    x = 10

func()
```

OSError

操作系统相关错误。

```
# 示例
import os
try:
    os.remove("non_existent_file.txt")
except OSError as e:
    print(f"OS error: {e}")
```

FileNotFoundError

文件或目录不存在。

```
# 示例
with open("non_existent_file.txt", "r") as f:
    content = f.read()
```

PermissionError

没有足够的权限执行操作。

```
# 示例（在 Unix 系统上尝试写入只读文件）
try:
    with open("/etc/passwd", "w") as f:
        f.write("test")
except PermissionError:
    print("Permission denied")
```

FileExistsError

尝试创建已存在的文件或目录。

```
# 示例
import os
os.mkdir("existing_directory")
```

IsADirectoryError / NotADirectoryError

期望文件但得到目录，或反之。

```
# 示例
import os
try:
    with open("/etc", "r") as f:  # /etc 是目录
        content = f.read()
except IsADirectoryError:
    print("Expected file but got directory")
```

ConnectionError

连接相关错误的基类。

```
# 各种连接错误子类
import socket

try:
    # 连接被拒绝
    s = socket.socket()
    s.connect(('localhost', 9999))  # 假设该端口未开放
except ConnectionRefusedError:
    print("Connection refused")
```

ReferenceError

弱引用对象被垃圾回收后访问。

```
# 示例
import weakref

class MyClass:
    pass

obj = MyClass()
weak_obj = weakref.ref(obj)
del obj
print(weak_obj())  # 返回 None，不会抛出 ReferenceError
# 但在某些弱引用代理场景中可能抛出
```

RuntimeError

当检测到不属于其他类别的错误时抛出。

```
# 示例
def bad_function():
    raise RuntimeError("Something went wrong")
```

NotImplementedError

抽象方法需要子类实现。

```
# 示例
class BaseClass:
    def must_implement(self):
        raise NotImplementedError("Subclasses must implement this metho
d")

class ChildClass(BaseClass):
```

```
    def must_implement(self):
        print("Implemented!")
```

RecursionError

递归深度超过最大限制。

```
# 示例
def recursive_function(n):
    return recursive_function(n + 1)

recursive_function(0)
```

SystemError

解释器内部错误。

TypeError

操作或函数应用于不适当类型的对象。

```
# 示例
"2" + 2                     # 字符串和数字相加
len(5)                      # 对整数使用 len()
[1, 2, 3]["key"]            # 用字符串索引列表
```

ValueError

操作或函数接收到具有正确类型但不合适的值。

```
# 示例
int("hello")                # 无法转换为整数
float("abc")                # 无法转换为浮点数
[1, 2, 3].index(4)          # 列表中不存在的值
```

UnicodeError

Unicode 相关错误的基类。

```
# 示例
"hello".encode('ascii').decode('utf-8')  # 编码解码不匹配
```

UnicodeDecodeError

解码时发生的 Unicode 错误。

```
# 示例
try:
    b'\xff'.decode('utf-8')
```

```
except UnicodeDecodeError as e:
    print(f"Decode error: {e}")
```

UnicodeEncodeError

编码时发生的 Unicode 错误。

```
# 示例
try:
    " café ".encode('ascii')
except UnicodeEncodeError as e:
    print(f"Encode error: {e}")
```

StopIteration

迭代器没有更多值。

```
# 示例
my_list = [1, 2, 3]
iterator = iter(my_list)
print(next(iterator))  # 1
print(next(iterator))  # 2
print(next(iterator))  # 3
print(next(iterator))  # 抛出 StopIteration
```

StopAsyncIteration

异步迭代器没有更多值。

```
# 示例
async def async_gen():
    yield 1
    yield 2

async def main():
    async for value in async_gen():
        print(value)
    # 自动处理 StopAsyncIteration
```

GeneratorExit

生成器或协程被关闭时抛出。

```
# 示例
def my_generator():
    try:
        yield 1
        yield 2
    except GeneratorExit:
        print("Generator is closing")
```

```
        raise  # 必须重新抛出

gen = my_generator()
next(gen)
gen.close()  # 触发GeneratorExit
```

KeyboardInterrupt

用户按下中断键（通常是Ctrl+C）。

```
# 示例
try:
    while True:
        pass
except KeyboardInterrupt:
    print("Program interrupted by user")
```

SystemExit

sys.exit()函数被调用。

```
# 示例
import sys
try:
    sys.exit(0)
except SystemExit as e:
    print(f"Program exiting with code: {e.code}")
```

警告

警告不是错误，但表示潜在问题。

DeprecationWarning

已弃用功能的警告。

```
# 示例
import warnings
warnings.warn("This function is deprecated", DeprecationWarning)
```

FutureWarning

将来会改变语义的构造的警告。

RuntimeWarning

可疑的运行时行为的警告。

SyntaxWarning

可疑语法的警告。

UserWarning

用户代码生成的警告。

ImportWarning

导入模块过程中触发的警告。

自定义异常

创建自定义异常类：

```
class MyCustomError(Exception):
    """我的自定义异常"""
    def __init__(self, message, error_code):
        super().__init__(message)
        self.error_code = error_code

    def __str__(self):
        return f"{self.args[0]} (Error Code: {self.error_code})"

# 使用自定义异常
def validate_age(age):
    if age < 0:
        raise MyCustomError("Age cannot be negative", 1001)
    if age > 150:
        raise MyCustomError("Age seems unrealistic", 1002)

try:
    validate_age(-5)
except MyCustomError as e:
    print(f"Custom error: {e}, Code: {e.error_code}")
```

异常处理最佳实践

基本异常处理

```
try:
    # 可能抛出异常的代码
    result = 10 / 0
except ZeroDivisionError:
    # 处理特定异常
    print("Cannot divide by zero")
except (TypeError, ValueError) as e:
    # 处理多个异常
```

```
    print(f"Type or value error: {e}")
except Exception as e:
    # 处理所有其他异常
    print(f"Unexpected error: {e}")
else:
    # 如果没有异常发生
    print("Operation successful")
finally:
    # 无论是否发生异常都会执行
    print("Cleanup code")
```

上下文管理器处理资源

```
# 使用with 语句自动处理资源
try:
    with open("file.txt", "r") as f:
        content = f.read()
except FileNotFoundError:
    print("File not found")
except PermissionError:
    print("No permission to read file")
```

异常链

```
# Python 3.0+ 支持异常链
def process_data():
    try:
        int("not a number")
    except ValueError as e:
        raise RuntimeError("Data processing failed") from e

try:
    process_data()
except RuntimeError as e:
    print(f"Error: {e}")
    print(f"Original cause: {e.__cause__}")
```

记录异常

```
import logging
import traceback

logging.basicConfig(level=logging.ERROR)

try:
    risky_operation()
except Exception as e:
    logging.error(f"Operation failed: {e}")
    logging.debug(traceback.format_exc())  # 记录完整堆栈跟踪
```

重新抛出异常

```python
def handle_exception():
    try:
        # 某些操作
        pass
    except ValueError as e:
        print(f"Handling ValueError: {e}")
        # 清理操作后重新抛出
        raise

try:
    handle_exception()
except ValueError:
    print("ValueError was re-raised")
```

这份文档涵盖了 Python 中绝大多数错误类型和异常处理的最佳实践。在实际开发中，根据具体需求选择合适的异常处理策略非常重要。