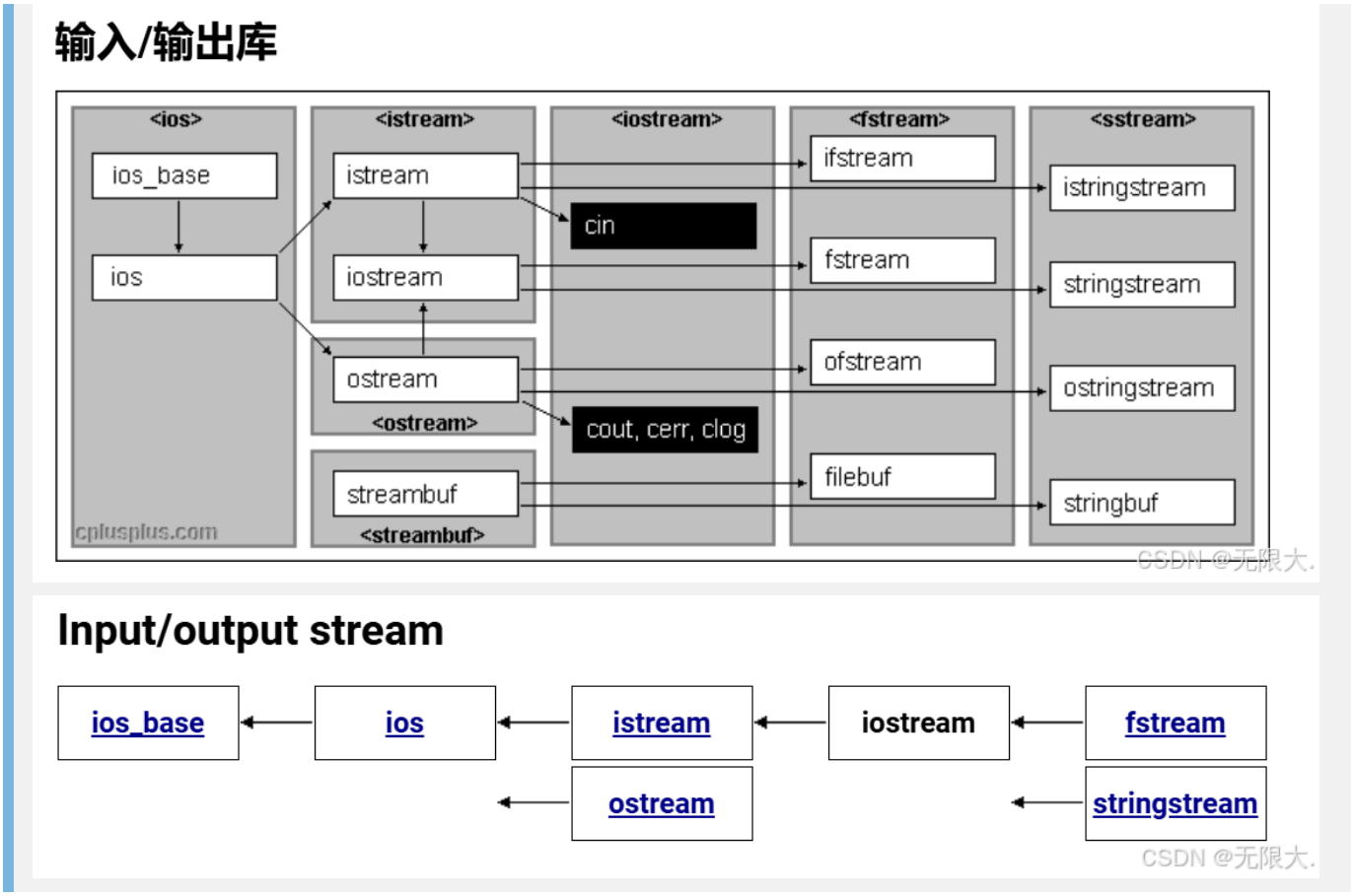
C++ 输入输出流详解

前言

在C++中，输入输出（I/O）流是程序与外界（如用户、文件、网络）进行数据交流的机制。通过输入输出流，我们可以读取用户输入的数据，并向用户展示结果，这一过程可以类比于日常生活中的交流。例如，当我们与他人对话时，信息以语言（相当于流）为载体在两个实体之间传递。在C++中，流的这种特性使得程序能够像人一样进行“交流”。

流之间的关系

在C++中，众多流之间存在包含及层级关系。标准流（如cin、cout、cerr）是基础，而文件流则是其扩展。标准输入输出流用于控制台交互，而文件流则可用于访问外部文件。



第一部分 -- 标准输入输出流

1. 标准输入输出流及其符号

标准输入输出流cin和cout分别用于输入和输出。它们使用提取符>>和插入符<<进行数据的输入输出。我们可以使用标准输入输出流给不同变量赋值。

```
#include <iostream>
using namespace std;

int main() {
    string name;
    int age;
    cout << "请输入姓名和年龄（用空格分隔）: ";
    cin >> name >> age; // 提取姓名和年龄
    cout << "姓名: " << name << ", 年龄: " << age << endl;

    // 如果用户输入了不匹配的格式，例如"John twenty"，将导致age未赋值
    return 0;
}
```

然而，如果我们使用提取符从cin提取字符串，例如"Hello World"，会发现仅能提取到第一个单词"Hello"，而"World"将留在标准输入流中，需再次提取。

```
#include <iostream>
using namespace std;

int main() {
    string s;
    cin >> s; // 只提取到Hello
    cout << s << endl; // 输出 Hello
    // 若要完整提取，需使用 getline
    return 0;
}
```

这个时候就利用到了get系列的相关函数，如下。

2. get系列函数

get系列函数用于更细粒度的控制输入：

- int get()：返回输入流一个字符的ASCII值。
- istream& get(unsigned char* pszBuf, int nBufLen, char delim='\n')：从输入流读取字符到缓冲区，遇到delim时停止，但不提取delim。
  - pszBuf：指向字符缓冲区的指针，用于保存结果
  - nBufLen：缓冲区长度
  - delim：结束字符，根据结束字符判断何时停止读入操作
- istream& getline(unsigned char* pszBuf, int nBufLen, char delim='\n')：读取字符并在遇到delim时停止，同时提取并丢弃delim。
  - pszBuf：指向字符缓冲区的指针，用于保存结果
  - nBufLen：缓冲区长度
  - delim：结束字符，根据结束字符判断何时停止读入操作

```
#include <iostream>
#include <limits>
#include <cstring>
using namespace std;

int main() {
    // 使用 get() 读取单个字符
    char ch;
    cout << "请输入一个字符：";
    ch = cin.get(); // 读取一个字符
```

```
    cout << "你输入的字符是: " << ch << ", ASCII值是: " << static_cast<int>(ch) <<
endl;

    // 清除输入流中的换行符
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    // 使用 cin.get(unsigned char* pszBuf, int nBufLen, char delim) 读取一行
    char buffer[50];
    cout << "请输入一行文本（最多49个字符）: ";
    cin.get(buffer, 50, '\n'); // 读取一行直到换行符
    cout << "你输入的文本是: " << buffer << endl;

    // 清除输入流的状态，以便后续操作
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    // 使用 cin.getline() 读取整行文本
    cout << "请输入另一行文本: ";
    cin.getline(buffer, 50); // 读取整行
    cout << "你输入的整行文本是: " << buffer << endl;

    return 0;
}
```

---

cin.get() 与 cin.getline() 的比较

让我们使用代码示例、图示化解释和详细的说明来比较`cin.get()`和`cin.getline()`的行为和差异。

1. 换行符处理

代码示例：

```
#include <iostream>
using namespace std;

int main() {
    char buffer[10];

    cout << "Using cin.getline():\n";
    cin.getline(buffer, 10); // 假设输入是 "Hello\n"
    cout << "Buffer content: " << buffer << endl;

    cout << "Using cin.get():\n";
    cin.get(buffer, 10); // 假设输入是 "Hello\n"
    cout << "Buffer content: " << buffer << endl;

    return 0;
}
```

解释及图示：

- `cin.getline()`：
  - 读取输入直到遇到换行符（`'\n'`），丢弃换行符，并在最后添加空字符（`\0`）。
  - 图示：

```
输入："Hello\n"
缓冲区：[H][e][l][l][o][\0]
输入流：为空（换行符丢弃）
```

- `cin.get()`：
  - 读取字符直到数组满或遇到特定字符，换行符不被读取，仍保留在输入流中。
  - 图示：

```
输入："Hello\n"
缓冲区：[H][e][l][l][o]
输入流：[\n]
```

2. 调用次序影响

代码示例：

```cpp
#include <iostream>
using namespace std;

int main() {
    char ch1, ch2;

    cout << "First cin.get(): ";
    ch1 = cin.get(); // 读取第一个字符假设输入 "H\n"
    cout << "First character: " << ch1 << endl;

    cout << "Second cin.get(): ";
    ch2 = cin.get(); // 读取换行符
    cout << "Second character: [" << ch2 << "]" << endl;

    return 0;
}
```

解释及图示：

- **调用顺序问题**:
  - 如果连续调用 `cin.get()`，第一次读取后，换行符仍在输入流中。
  - 第二次调用会读取残留的换行符，导致误解为输入空白。
  - 图示:

```
第一次 cin.get(): "H"
输入流: [\n]

第二次 cin.get(): "\n" -> 导致空读取（通常不期望的行为）
```

3. 缓冲区超长输入影响

代码示例:

```cpp
#include <iostream>
using namespace std;

int main() {
    char buffer[5];

    cout << "Using cin.getline():\n";
    cin.getline(buffer, 5); // 输入 "HelloWorld\n"
    cout << "Buffer content: " << buffer << endl; // 缓冲区溢出，设置状态位

    cin.clear(); // 清除状态位
    cin.ignore(numeric_limits<streamsize>::max(), '\n'); // 清理剩余内容

    cout << "Using cin.get():\n";
    cin.get(buffer, 5); // 输入 "HelloWorld\n"
    cout << "Buffer content: " << buffer << endl; // 剩下的字符保留在输入流中

    return 0;
}
```

解释及图示:

- `cin.getline()`:
  - 当输入字符数量超出缓冲区大小，`cin.getline()`会设置流的失效位并丢弃超出的字符。
  - 图示:

```
输入: "HelloWorld\n"
缓冲区大小: 5
```

```
缓冲区：[H][e][l][l][o][\0]
流失效，需重置状态
```

- `cin.get()`:
  - 超长输入情况下，只读取定长字符，剩余字符保留在输入流供后续处理。
  - 图示:

```
输入："HelloWorld\n"
缓冲区大小：5
缓冲区：[H][e][l][l][o]
输入流：[W][o][r][l][d][\n]
```

小结:

- `cin.getline()`:
  - 适用于读取整行文本，自动管理换行符并丢弃它。
  - 防止缓冲区溢出，设置和管理流失效位。
- `cin.get()`:
  - 精确读取字符，换行符保留在流中，需要注意换行符处理。
  - 剩余输入保留在流中供后续读取。

---

3. 处理流错误

流的状态处理是C++中非常重要的一个主题，因为在进行输入输出操作时，预料之外的错误常常会影响程序的运行。以下是对流状态函数的详细介绍，以及如何使用这些函数进行流状态管理的代码示例。

状态检测函数

- `good()`:
  - 返回 `true` 表示流没有错误，正常运行。
- `eof()`:
  - 返回 `true` 如果已到达流的末尾。这在处理文件读写时尤其重要。
- `fail()`:

- 返回 `true` 表示上一个输入输出操作失败，通常是由于非法数据导致的。

- `bad()`:
  - 返回 `true` 表示发生了严重的错误，流无法继续使用，比如流对象损坏或底层设备访问错误。

- **rdstate()**
  - rdstate() 是一个无参数的方法，它返回一个整数，表明流的当前状态。该整数是 `failbit`、`eofbit` 和 `badbit` 的组合，可以通过后续的状态检测函数进行分析。

3. 代码示例

下面的代码示例展示了如何使用这些方法进行流状态检查：

```
#include <iostream>
#include <limits> // 用于 numeric_limits<streamsize>::max()
using namespace std;

int main() {
    int number;

    // 输入读取
    cout << "请输入一个整数: ";
    cin >> number;

    // 检查输入状态
    if (cin.good()) {
        cout << "输入的整数是: " << number << endl;
    }
    else if (cin.eof()) {
        cout << "已到达输入流的末尾。" << endl;
    }
    else if (cin.fail()) {
        cout << "输入错误! 非法数据。" << endl;
        // 清除错误状态
        cin.clear(); // 将状态重置为正常
        // 清理输入缓冲区，直到下一个换行符
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "请重新输入一个整数: ";
        cin >> number; // 再次尝试读取
    }
    else if (cin.bad()) {
        cout << "发生了致命错误! 流无法继续使用。" << endl;
    }

    // 最终结果
    if (cin.good()) {
        cout << "重新输入的整数是: " << number << endl;
```

```
    }

    return 0;
}
```

4. 代码解析

1. **输入读取**：
   - 用户输入一个整数，如果输入过程中有错误（如输入字母而不是数字），系统会捕捉到这个错误。
2. **状态检查**：
   - 首先检查流的状态，如果 `good()` 返回true，则正常输出数字。
   - 如果 `eof()` 返回true，说明读取已到达流末尾。
   - 如果 `fail()` 返回true，说明输入出现了非法数据，调用`cin.clear()`将状态重置为正常状态，然后用 `cin.ignore()` 处理输入缓冲区，直到下一个换行符，确保后续输入不会受影响。
   - 如果 `bad()` 返回true，表示出现了不可恢复的错误。
3. **重新尝试输入**：
   - 在处理完输入错误后，程序允许用户再次输入一个整数。

总结

- **流状态管理** 是确保程序健壮性的重要部分，可以有效处理用户输入可能导致的各种状态。
- 理解并使用`good()`、`eof()`、`fail()`和`bad()`可以显著提高程序的容错能力，并创建更友好的用户体验。

第二部分 -- 文件输入输出流

1. 文件输入输出流

- 文件流使得C++可以读写文件数据。主要有`ifstream`（输入文件流）、`ofstream`（输出文件流）和`iofstream`（输入输出文件流）。
- 依照“打开—>读写—>关闭”原语进行操作

- 文件读写所用到的很多常数都在基类ios中被定义出来

2. 文件打开与关闭

文件打开使用open()函数，关闭使用close()函数。

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream outfile("example.txt");
    if (outfile.is_open()) {
        outfile << "写入示例" << endl;
        outfile.close();
    }
    return 0;
}
```

3. 文件的读写

对于文本文件，使用ifstream来读取和ofstream来写入。对于二进制文件，需使用ios::binary模式，主要是通过read( )、write( )函数完成读写二进制文件功能。

```
#include <fstream>
using namespace std;

int main() {
    // 写入二进制文件
    ofstream out("example.dat", ios::binary);
    int x = 10;
    out.write(reinterpret_cast<char*>(&x), sizeof(x));
    out.close();

    // 读取二进制文件
    ifstream in("example.dat", ios::binary);
    in.read(reinterpret_cast<char*>(&x), sizeof(x));
    in.close();
    return 0;
}
```

4. 定位输入输出流

当然，这里是关于C++中流定位的详细介绍，包含了 `seekg` 和 `seekp` 的用法，以及更全面的代码示例。

流的定位

在C++中，输入输出流可以通过以下标识来进行定位：

- `ios::beg`: 定位到流的开始位置。
- `ios::cur`: 定位到流的当前位置。
- `ios::end`: 定位到流的结束位置。

函数 `seekg` 和 `seekp`

- `seekg`: 用于设置输入流（`ifstream`）的读位置。
- `seekp`: 用于设置输出流（`ofstream`）的写位置。

这两个函数的原型如下：

```
streampos seekg(streamoff off, ios_base::seekdir way);
streampos seekp(streamoff off, ios_base::seekdir way);
```

- `off`: 文件中的偏移量。
- `way`: 定义偏移的起点，可以是 `ios::beg`, `ios::cur`, 或 `ios::end`。

代码示例

以下是一个完整的示例，其中演示了如何使用 `seekg` 和 `seekp` 来定位流：

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

int main() {
    // 创建并写入示例文件
    ofstream outFile("example.dat", ios::binary);
    const char *text = "Hello, World!";
    outFile.write(text, strlen(text)); // 写入字符串
    outFile.close(); // 关闭文件

    // 打开文件进行读取
    ifstream inFile("example.dat", ios::binary);

    // 定位到文件结尾并获取文件大小
    inFile.seekg(0, ios::end); // 移动到文件结尾
    streampos fileSize = inFile.tellg(); // 获取当前位置（即文件大小）
```

```
    inFile.seekg(0, ios::beg); // 定位回文件开头

    cout << "文件大小: " << fileSize << " 字节" << endl;

    // 读取文件内容
    char *buffer = new char[fileSize + 1]; // 创建缓冲区以存储文件内容
    inFile.read(buffer, fileSize); // 读取文件内容
    buffer[fileSize] = '\0'; // 设置字符串结束符

    cout << "文件内容: " << buffer << endl; // 显示文件内容

    // 再次读取，使用 seekg 定位到某个位置
    inFile.seekg(7, ios::beg); // 定位到文件开头后7个字节的位置
    char character;
    inFile.get(character); // 读取该位置的字符
    cout << "第8个字符: " << character << endl; // 显示读取的字符

    // 清理
    delete[] buffer; // 释放动态分配的内存
    inFile.close(); // 关闭文件

    return 0;
}
```

代码解析

1. **写入文件**:
   - 首先创建一个二进制输出文件 `example.dat` 并写入字符串 `Hello, World!`。
2. **定位文件大小**:
   - 使用 `seekg(0, ios::end)` 将读取位置移动到文件的结尾，接着使用 `tellg()` 获取当前的位置（即文件的字节大小）。
3. **读取文件内容**:
   - 将流位置重置回文件的开始位置（`seekg(0, ios::beg)`）。
   - 创建一个动态缓冲区以存储文件内容，读取文件数据，并确保在末尾添加空字符以形成有效的字符串。
4. **再次使用 `seekg` 定位**:
   - 使用 `seekg(7, ios::beg)` 将位置设置为开头后的第八个字节，然后读取该字符并输出。
5. **内存管理**:
   - 释放动态分配的内存并关闭文件流。

第三部分 -- 字符串输入输出流

1. 字符串的输入输出流

字符串也可以被视为流，通过`stringstream`类来处理字符串流。

- `istringstream`：输入流，提供读string功能
- `ostringstream`：输出流，提供写string功能
- `stringstream`：输入输出流，读写string功能

2. 代码示例

反向解字符串

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

int main() {
    // 假设我们有一个字符串，内容为 "42 John 3.14"
    string input = "42 John 3.14";

    // 利用 stringstream 来解析字符串
    stringstream ss(input);

    // 定义变量来存储解析结果
    int age;
    string name;
    double pi;

    // 从 stringstream 中提取各个值
    ss >> age >> name >> pi;

    // 输出结果
    cout << "年龄: " << age << endl;
    cout << "名字: " << name << endl;
    cout << "PI: " << pi << endl;

    return 0;
}
```

代码解析

1. **输入字符串**:
    - 假设我们有一个字符串 `input`，其中包含多个变量，用空格分隔。
2. **使用 `stringstream`**:
    - 包含字符串的 `input` 被传递给 `stringstream`，允许我们像操作流一样处理它。
3. **提取变量**:
    - 使用 `>>` 操作符从 `stringstream` 中提取变量。在这个例子中，从字符串中提取了一个整数 `age`、一个字符串 `name` 和一个浮点数 `pi`。
4. **输出结果**:
    - 使用 `cout` 输出各个变量的值。

可以使用 C++ 的 `ostringstream` 类来实现将不同类型的数据合并到一个字符串中。以下是一个示例代码，展示如何将整型、浮点型和字符串合并为一个单一的字符串。

合并不同类型的数据到字符串

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

int main() {
    // 定义要合并的数据
    int age = 25;
    string name = "Alice";
    double height = 5.5;

    // 使用 ostringstream 把数据合并到字符串中
    ostringstream oss;
    oss << "Name: " << name << ", Age: " << age << ", Height: " << height << "
feet";

    // 获取合并后的字符串
    string result = oss.str();

    // 输出合并后的字符串
    cout << result << endl;
```

```
    return 0;
}
```

代码解析

1. **定义变量**:
    - 定义一个整型 `age`，一个字符串 `name`，和一个浮点型 `height`，它们包含需要合并的数据。
2. **使用 `ostringstream`**:
    - 创建一个 `ostringstream` 对象 `oss`。这个对象用于将不同类型的数据以流的方式写入，最终形成一个字符串。
3. **数据合并**:
    - 使用输出操作符 `<<` 将不同类型的数据添加到 `oss` 中。可以适当地添加文字描述，也可以格式化输出。
4. **获取结果字符串**:
    - 使用 `oss.str()` 方法获取合并后的字符串，并保存到 `result` 中。
5. **输出结果**:
    - 使用 `cout` 将最终的合并字符串输出到控制台。

综合示例

在C++中，可以通过重载 `<<` 操作符来实现自定义对象的输出格式。以下是一个示例，展示如何定义一个 `Student` 类，包括学生的属性，然后通过重载 `<<` 操作符从键盘输入属性值并在屏幕上显示。

示例代码

```
#include <iostream>
#include <string>
using namespace std;

class Student {
private:
    string name;
    int age;
```

```
    double grade;

public:
    // 构造函数
    Student() : name(""), age(0), grade(0.0) {}

    // 输入学生属性值
    void input() {
        cout << "请输入学生姓名: ";
        getline(cin, name);
        cout << "请输入学生年龄: ";
        cin >> age;
        cout << "请输入学生成绩: ";
        cin >> grade;
        cin.ignore(); // 清除输入缓冲区中的换行符
    }

    // 重载<<运算符
    friend ostream& operator<<(ostream& os, const Student& student) {
        os << "姓名: " << student.name << ", 年龄: " << student.age << ", 成绩: " <<
student.grade;
        return os;
    }
};

int main() {
    Student student;

    // 输入学生属性
    student.input();

    // 输出学生属性
    cout << student << endl;

    return 0;
}
```

代码解析

1. **定义 `Student` 类**:
    - 包含三个私有属性 `name`、`age` 和 `grade`，表示学生的姓名、年龄和成绩。
2. **构造函数**:
    - 默认构造函数初始化这些属性。
3. **输入方法 `input()`**:
    - 使用 `getline` 来读取学生姓名，以允许输入包含空格的字符串。
    - 使用 `cin`读取年龄和成绩。注意，在读取整型和浮点型后调用 `cin.ignore()`，以清除输入缓冲区中的换行符，以防止后续输入出错。

4. **重载 `<<` 操作符**:
    - 友元函数 `operator<<` 被定义为接收一个输出流和 `Student` 对象的引用。它将学生的信息格式化并输出到流中。
5. **主函数**:
    - 创建 `Student` 对象并调用 `input()` 方法从键盘获取输入。
    - 使用 `cout` 输出学生信息，利用重载的 `<<` 操作符格式化输出。