二进制基础全解析

一、二进制的定义与核心本质

二进制是计算机领域的基础计数制，仅用0和1两个数字表示所有数值，遵循“逢二进一”的计数规则，与我们日常使用的十进制（逢十进一）、前文提到的八进制（逢八进一）、十六进制（逢十六进一）同属进位计数制。

其核心价值源于计算机硬件的工作原理——计算机底层由大量晶体管组成，晶体管的“导通”与“截止”两种状态可直接对应0和1，这是二进制成为计算机底层唯一计数方式的根本原因，也是所有数字设备（手机、电脑、服务器等）数据存储与运算的基础。

二进制的数位从右至左依次为第0位、第1位、第2位……，每位的权重为2的对应次方，例如：二进制数1011，从右至左的权重分别为2^0、2^1、2^2、2^3。

二、二进制与常见计数制的转换

（一）二进制转十进制

转换规则：将二进制的每一位数字乘以对应位的权重（2^n，n为当前数位从右开始的序号，起始为0），再将所有结果相加，得到十进制数值。

示例1：将二进制1011转为十进制

计算过程：$1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 8 + 0 + 2 + 1 = 11$，即二进制1011对应十进制11。

示例2：将二进制11001转为十进制

计算过程：$1\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 = 16 + 8 + 0 + 0 + 1 = 25$，即二进制11001对应十进制25。

（二）十进制转二进制

常用两种方法，分别适用于整数和小数，此处重点讲解整数转换（小数转换多用于高精度场景，日常开发较少用到）。

1. 除2取余法（整数转换）

转换规则：将十进制整数反复除以2，每次记录余数（0或1），直到商为0，最后将所有余数从后往前排列，即为对应的二进制数。

示例：将十进制18转为二进制

计算过程：

18 ÷ 2 = 9 余数0

9 ÷ 2 = 4 余数1

4 ÷ 2 = 2 余数0

2 ÷ 2 = 1 余数0

1 ÷ 2 = 0 余数1

余数逆序排列：10010，即十进制18对应二进制10010。

2. 小数转换（补充）

转换规则：将十进制小数反复乘以2，每次记录整数部分（0或1），直到小数部分为0（或达到所需精度），整数部分顺序排列即为二进制小数。

示例：将十进制0.625转为二进制

0.625×2=1.25 整数部分1

0.25×2=0.5 整数部分0

0.5×2=1.0 整数部分1

结果：0.101，即十进制0.625对应二进制0.101。

（三）二进制与八进制、十六进制转换

前文提到八进制、十六进制在开发中比二进制更易读写，三者转换可借助二进制作为中间桥梁，效率更高。

1. 二进制转八进制

规则：从二进制右边开始，每3位为一组（不足3位补0），每组对应1位八进制数（0-7）。

示例：二进制110101转为八进制

分组：011 101（右边开始分组，不足3位补0），每组对应八进制3、5，结果为65（八进制）。

2. 二进制转十六进制

规则：从二进制右边开始，每4位为一组（不足4位补0），每组对应1位十六进制数（0-9、A-F，A=10、F=15）。

示例：二进制11010110转为十六进制

分组：1101 0110，每组对应十六进制D、6，结果为D6（十六进制）。

三、二进制的运算规则

二进制运算分为算术运算和逻辑运算，是计算机运算的核心基础，也是C++等编程语言中位运算的底层原理。

（一）算术运算

1. 加法（逢二进一）

核心规则：0+0=0；0+1=1；1+1=10（向高位进1）；1+1+1=11（向高位进1）。

示例：1011 + 0101 = 10000

2. 减法（借一当二）

核心规则：0-0=0；1-0=1；1-1=0；0-1=1（向高位借1，借1当2）。

示例：1000 - 0101 = 0011

3. 乘法与除法

二进制乘法类似十进制乘法，可简化为“加法+移位”；除法可简化为“减法+移位”，是计算机底层优化运算效率的核心逻辑。

（二）逻辑运算（重点，C++位运算基础）

逻辑运算针对二进制的每一位独立运算，不涉及进位/借位，是C++中位运算（&、|、^、~、<<、>>）的基础，日常开发中高频使用。

1. 与运算（&）

规则：两位均为1时，结果为1；否则为0。

用途：用于保留指定位、清除无关位（例如保留二进制数的低4位）。

示例：1011 & 0110 = 0010

2. 或运算（|）

规则：两位中至少有1位为1时，结果为1；否则为0。

用途：用于设置指定位为1（例如将二进制数的低2位置1）。

示例：1011 | 0110 = 1111

3. 异或运算（^）

规则：两位不同时，结果为1；两位相同时，结果为0。

用途：用于翻转指定位、两个数交换（无需临时变量）。

示例：1011 ^ 0110 = 1101

4. 非运算（~）

规则：对每一位取反，1变0，0变1（属于单目运算）。

注意：在C++中，非运算会影响符号位（补码表示），需注意溢出问题。

示例：~1011（假设为4位）= 0100

5. 移位运算（<<、>>）

左移（<<）：将二进制数整体左移n位，右边补0，等价于乘以2^n；

右移（>>）：将二进制数整体右移n位，正数左边补0，负数左边补1（取决于编译器），等价于除以2^n（向下取整）。

示例：1011 << 1 = 10110（等价于11×2=22）；1011 >> 1 = 0101（等价于11÷2=5）。

四、二进制的实际应用场景

（一）计算机底层存储与运算

所有数据（文本、图片、音频、视频）在计算机中均以二进制形式存储：文本通过ASCII码/Unicode码映射为二进制，图片通过像素点的RGB值转换为二进制，音频/视频通过采样、编码转为二进制流。计算机的CPU运算、内存读写，本质上都是二进制的运算与传输。

（二）C++开发中的位运算场景

二进制是C++位运算的基础，位运算比普通算术运算效率更高，常用于以下场景：

1. 底层硬件操作：嵌入式开发中，通过位运算配置寄存器（设置/清除指定位）；

2. 数据压缩与加密：通过位运算优化数据存储体积，或实现简单的加密逻辑；

3. 高效运算：替代乘除2的幂运算（左移/右移），提升代码执行效率；

4. 状态标记：用二进制的每一位表示一个状态（例如用1字节8位表示8个开关状态），节省内存。

示例（C++位运算设置状态）：

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

int main() {
    // 用二进制每一位表示一个状态（1表示开启，0表示关闭）
    int status = 0; // 初始状态：0000 0000
    status |= 1 << 0; // 第0位置1（开启状态0），结果：0000 0001
    status |= 1 << 2; // 第2位置1（开启状态2），结果：0000 0101
    cout << "当前状态：" << status << endl; // 输出5（十进制），对应二进制0000 0101
    return 0;
}

```

（三）网络传输

网络数据传输的最小单位是比特（bit，即二进制的1位），字节（Byte）是基本单位（1Byte=8bit）。数据在网络中传输时，会被拆分为二进制比特流，传输完成后再重组为原始数据，二进制是网络通信的底层载体。

五、学习二进制的注意事项

1. 区分二进制与其他计数制的表示：C++中，二进制无直接表示法（C++14起可通过0b前缀表示，如0b1011），八进制以0前缀表示，十六进制以0x前缀表示，避免混淆；

2. 理解补码机制：计算机中负数以补码形式存储，二进制的非运算、移位运算需结合补码理解，避免出现逻辑错误；

3. 重点掌握逻辑运算：二进制的逻辑运算与C++位运算直接关联，是开发中最常用的二进制相关知识，需熟练掌握；

4. 无需死记硬背转换公式：日常开发中可借助工具（如计算器、代码片段）完成进制转换，重点理解转换逻辑。

六、总结

二进制是计算机技术的基础，虽然日常开发中很少直接编写二进制数，但它是理解计算机底层运算、C++位运算、数据存储与传输的核心。对于C++学习者而言，无需深入钻研二进制的复杂转换，重点掌握“逢二进一”规则、逻辑运算及位运算的应用，就能满足绝大多数开发场景的需求，同时为后续学习底层开发、嵌入式开发打下基础。

（注：文档部分内容可能由 AI 生成）