STM32硬件I2C死锁原因及解决办法

一些基础知识：

- I2C通信的两条信号线需要使用OD方式，连接上拉电阻
- I2C通信有主机、从机之分，主机即为发起通信的一方。主机未必是数据的发送方或者接收方
- I2C总线上可以有多个设备，每次仅能有一个设备控制总线
- 如何界定当前总线是由哪个设备控制？看当前哪个设备能拉低SCL\SDA信号线
  - 理论上来说，所有I2C设备都可以拉低这两条信号线
  - 但实际操作中，I2C设备会有个标志，标志当前总线是否由自己控制，如果不是自己控制，它是不会去拉低总线的
  - I2C检测总线被占用的方式也很简单，检测SCL\SDA是否都是高电平，如果都是，则为空闲

死锁的发生：

- 一般死锁发生于主机与从机通信之间，出现意外的通信错误（可能是SCL、SDA与3V3 GND意外短路，或者通信速率太高，而总线电容太大，导致丢失某些bit），或者丢失ack
- 死锁即为：
  - 主机认为此时应该由从机控制总线（即可能是主机在等待从机发送一定数量的数据，但中间丢了1个bit，没发送完）
  - 从机认为此时应该由主机控制总线（即从机已经把数据发送完了，在等待主机接管总线）
  - 双方互相等待，陷入死锁

死锁的解决方式：

- 硬件上可以为所有I2C设备设置一个统一的复位电路，一旦陷入死锁，断开I2C设备的电源，重新上电（较为麻烦，且重新上电后需要重新初始化I2C设备）
- 软件上解决方式：（STM32）
  - 将SCL SDA配置为OD模式，而不是AF OD，即手动控制这两个管脚
  - 手动发送9个SCL脉冲（期间检测SCL、SDA是否回到空闲状态（高电平），如果已经是高电平，则跳出）
  - 重新初始化SCL、SDA为AF OD，重新初始化I2C
- 死锁的复现方式：（如何测试写好的恢复函数是否真的有效）
  - 在I2C恢复函数下个断点（检测到I2C多次超时之后，应该能跳转到I2C恢复函数）
  - 使用镊子，将SCL与SDA短接，很快就能看到程序停到恢复函数的断点上，此时再执行恢复函数，看能否正常走出（可在回复函数中写个死循环，只有I2C正常才跳出，检测I2C正常的办法，可以读从设备的ID）

示例代码

```
void HAL_I2C_MspInit(I2C_HandleTypeDef *i2cHandle)
{

  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if (i2cHandle->Instance == I2C1)
  {
    /* USER CODE BEGIN I2C1_MspInit 0 */

    /* USER CODE END I2C1_MspInit 0 */

    __HAL_RCC_GPIOB_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_6 | GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_OD;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    for (int i = 0; i < 10; ++i)
    {
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
      HAL_Delay(1);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
      HAL_Delay(1);
    }

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_SET);
```

```
    HAL_Delay(1);
    i2cHandle->Instance->CR1 |= I2C_CR1_SWRST; //复位I2C控制器
    HAL_Delay(1);
    i2cHandle->Instance->CR1 = 0; //解除复位（不会自动清除）

    /**I2C1 GPIO Configuration
    PB6     ------> I2C1_SCL
    PB7     ------> I2C1_SDA
    */
    GPIO_InitStruct.Pin = GPIO_PIN_6 | GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    /* I2C1 clock enable */
    __HAL_RCC_I2C1_CLK_ENABLE();
    /* USER CODE BEGIN I2C1_MspInit 1 */

    /* USER CODE END I2C1_MspInit 1 */
  }
}


/* USER CODE BEGIN 1 */
void I2C_Reset()
{
  HAL_I2C_MspDeInit(&hi2c1);
  hi2c1.State = HAL_I2C_STATE_RESET;
  MX_I2C1_Init();
  // 硬件i2c会出现死锁，当超时次数达到一定数量，即很有可能是发生了死锁
  // 所谓死锁是指主机与从机互相等待，主机以为总线在从机手上控制，从机以为总线在主机手上控制，一直再等待对方释放总线
}
```