一、I2C通信协议

1.1 概述

- I2C总线是由Philips公司开发的一种简单、**双向二线制同步串行总线。**它只需要两根线（串行数据线SDA，串行时钟线SCL）即可在连接于总线上的器件之间传送信息。SDA（串行数据线）和SCL（串行时钟线）都是双向I/O线，**接口电路为开漏输出**，需通过**上拉电阻**保证总线空闲时两根线都是高电平。
- 总线上扩展的器件数量主要由电容负载来决定
- 主器件用于启动总线传送数据，并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件。
- 传输速率在标准模式下可以达到100kb/s，在快速模式下可以达到400kb/s，在高速模式下可以达到3.4Mb/s
- 支持7bit或10bit地址
- 支持SMBus协议

1.2 SMBus协议简介

SMBus，系统管理总线，两线接口，基于I2C操作原理。利用系统管理总线，设备可提供制造商信息，告诉系统它的型号/部件号，保存暂停事件的状态，报告不同类型的错误，接收控制参数，和返回它的状态。SMBus为系统和电源管理相关的任务提供控制总线。

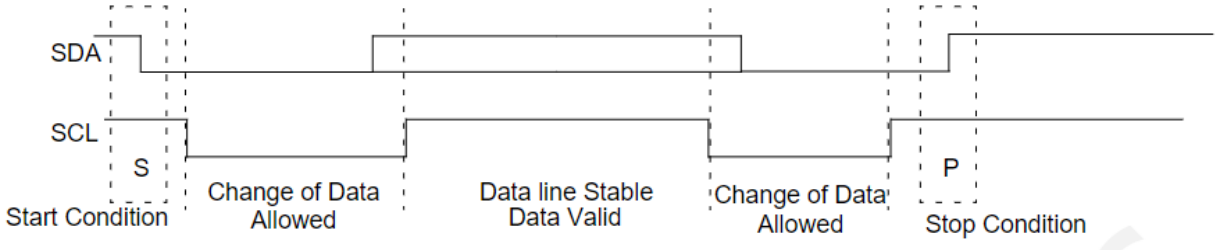
1.2.1 SMBus与I2C的不同点

~	SMBus	I2C
传输速度	10K-100K	0-3.4M
时钟超时	35ms时钟低超时	无
数据保留时间要求	300ns	无
总线协议	不同的总线协议（快速命令、处理呼叫等）	无总线协议

1.3 I2C协议

1.3.1 开始信号与停止信号

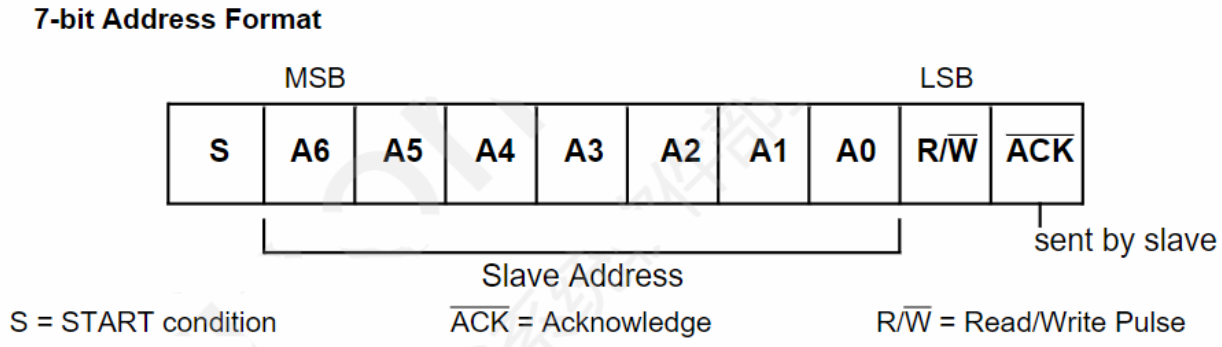
总线空闲时，SCL和SDC都被上拉电阻拉高；当主机希望开始一次数据传输时，需要发出一个开始信号（SCL为高时SDA由高到低）；当主机希望停止一次数据传输时，需要发出一个停止信号（SCL为高时SDA由低到高）。



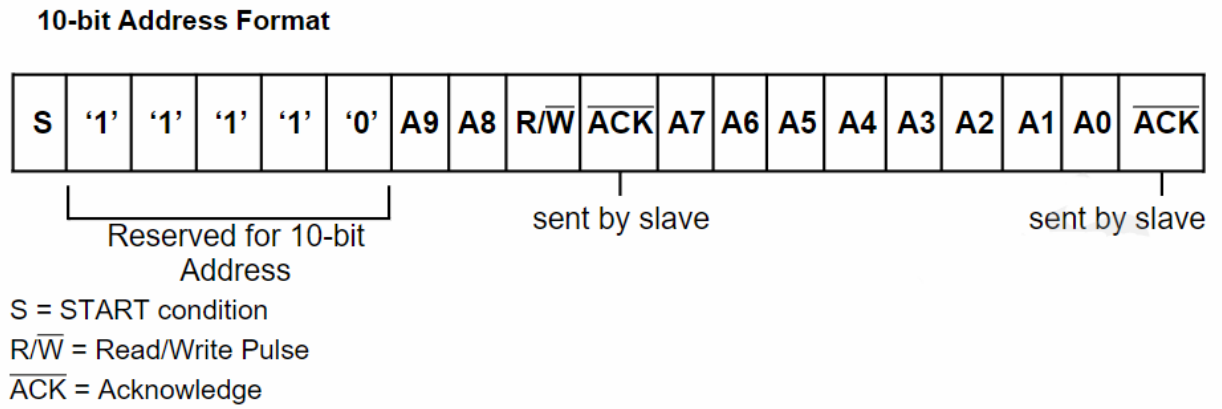
1.3.2 从机地址

有两种从机地址格式，7位地址和10位地址。

- 在7位地址的数据帧中，第一个字节的前7位是从机地址，最后一位是读写标志位，0代表主机向slave发数据，1代表主机从slave中读取数据。发送端每次发出一个字节，接收端都要发一个ACK信号回应（低电平有效）。

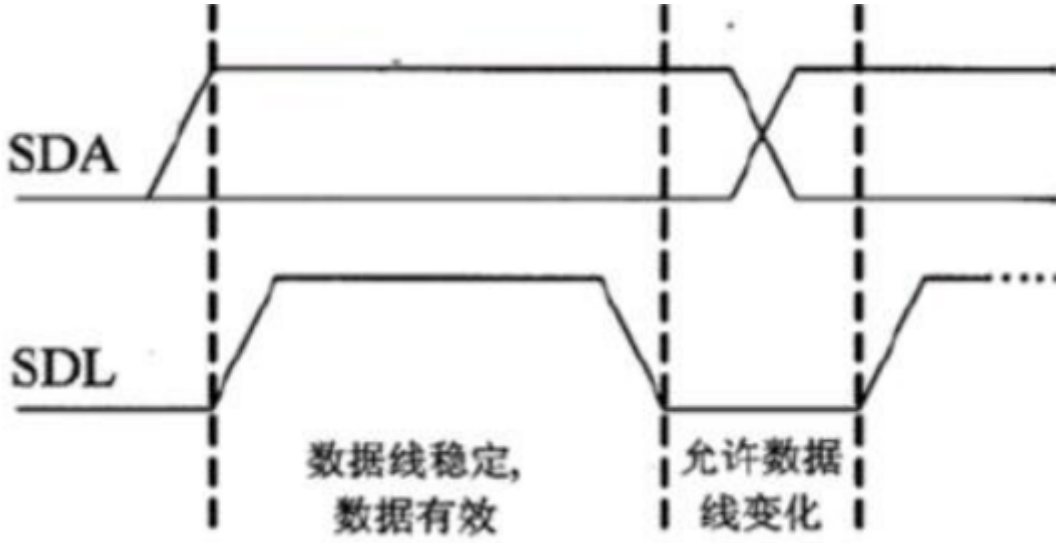


- 在10位地址的数据帧中，需要**用两个字节**来传递从机地址。第一个字节的高五位通知从机这是一个10位地址；接下来两位是从机地址的高两位；最后一位代表读写。发送完第一个字节后，接收端同样要发送一个ACK信号。之后主机继续发送第二个字节，第二个字节的8位就是地址的低8位，之后从机再发一个ACK信号，地址传输完毕。



1.3.3 数据变化时序

I2C一共有两根总线，时钟线SDL用来控制传输速率，数据线SDA用来传递数据。SDL的一个周期内完成SDA的变化与读取。在SDL低电平时允许SDA变化，在SDL高电平时SDA保持不变，主机或从机读取数据。

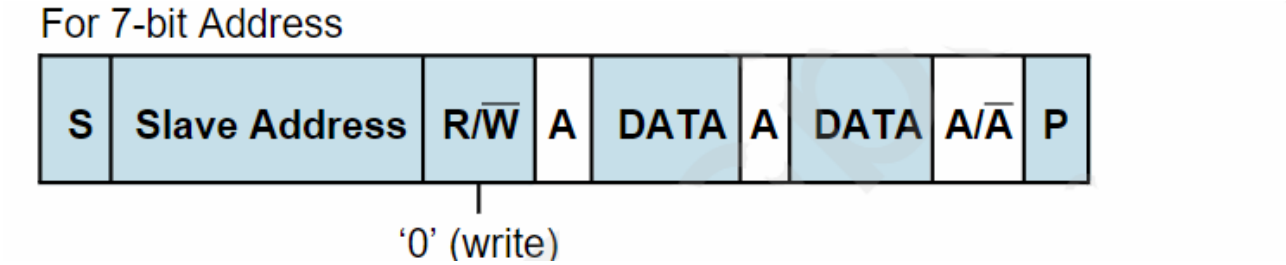


1.3.4 发送和接受协议

主机可以初始化数据传输，作为主机发送者或者主机接收者，最多只能有一个从机响应主机，作为与主机相对应的从机接收者或从机发送者。所有数据都是通过字节发送的，每次数据传输过程没有字节数的限制。主机每发送一个字节从机必须回应一个ACK信号。当不回应时，主机发送停止信号来停止传输，

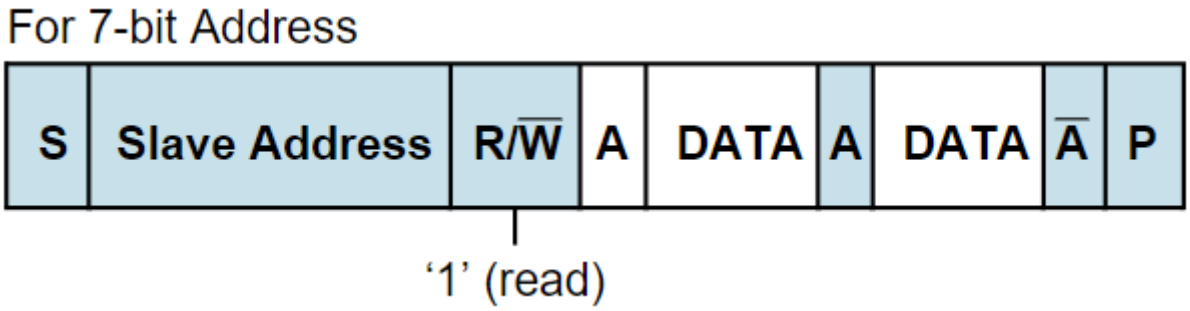
主机发送从机接收

主机向送机发送数据流程如下图所示。主机发送开始信号->主机发送从机地址+W->从机ACK->主机发送数据（如果从机有子地址，第一个数据就是子地址）->从机ACK->主机发数据->从机ACK->所有数据发送完毕后，主机发送停止信号，释放总线（最后一个数据从机回应不回应都无所谓，因为控制权在主机，主机知道什么时候结束）。

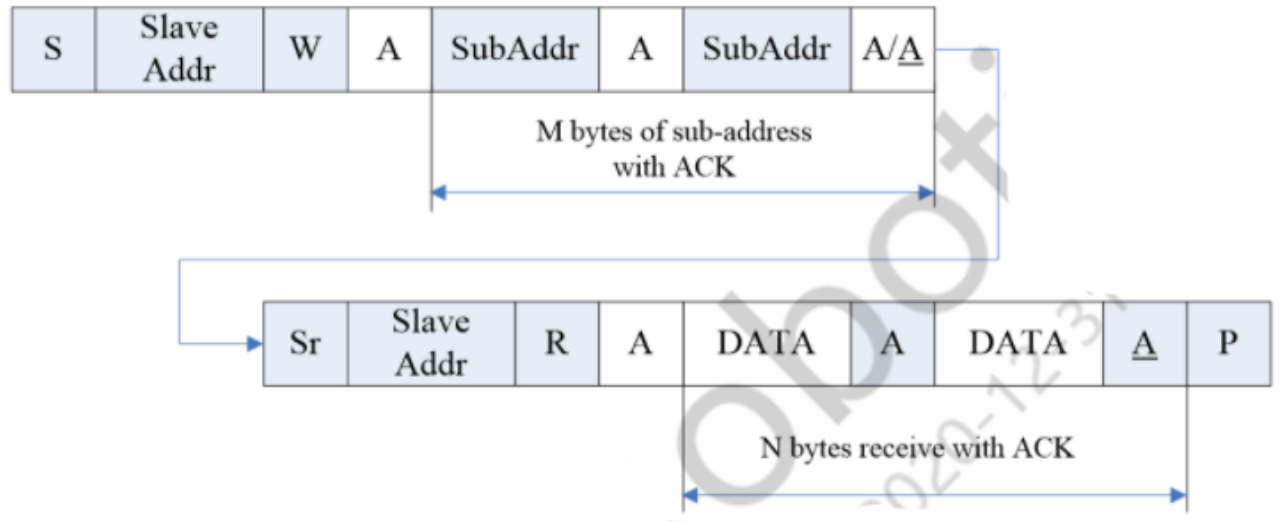


主机接收从机发送

主机向从机读取数据流程如下图所示。在没有子地址的情况下，和发送数据流程基本一样。主机发送开始信号->主机发送从机地址+R->从机ACK->从机发送数据->主机ACK->从机发数据->主机ACK->所有数据发送完毕后主机发送停止信号，释放总线（最后一个数据主机必须不回应，让从机释放SDA总线）



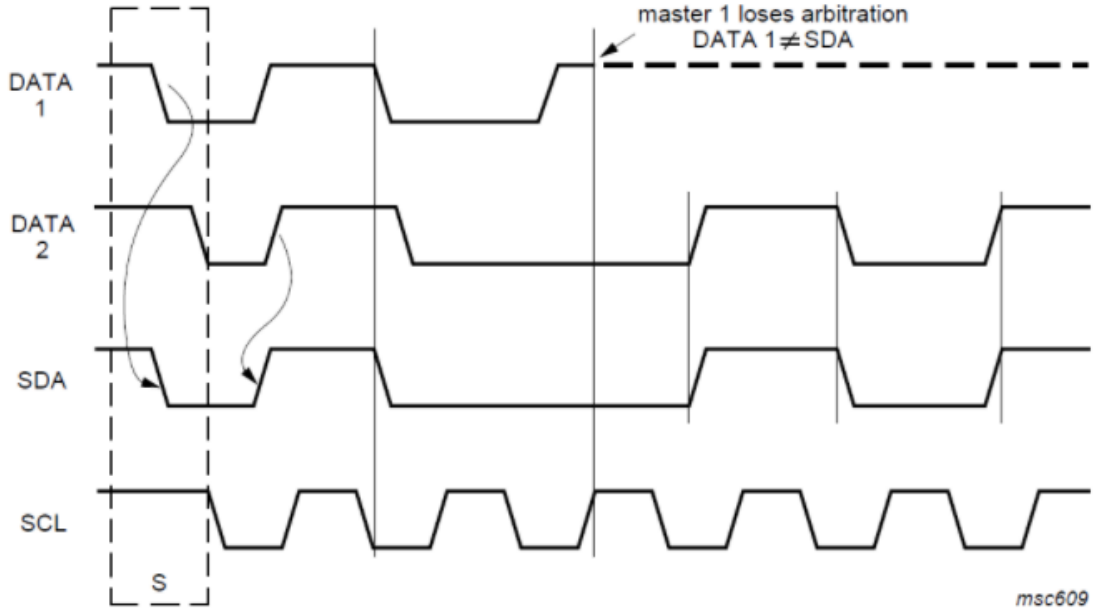
有子地址的情况下主机读数据要稍微复杂一些。因为子地址要主机写，数据是主机读，要涉及读与写之间的转变。主机发送开始信号->主机发送从机地址与+W->从机ACK->主机发送从机子地址->从机ACK->主机发送重新开始信号（同开始信号）->主机发送从机地址+R->从机ACK->从机发数据->主机ACK->重复上面过程->所有数据发送完毕后，主机发送停止信号，释放总线。



1.4 仲裁

当总线上有一个以上的主机时，协议通过仲裁的方法确定哪个主机获得总线的使用权。从机不参与仲裁的过程。当总线处于空闲状态（IDLE）时，多个主机都可能发起开始条件（START）在总线上传输数据。仲裁用来判断哪个主机的传输可以正常进行。

仲裁是按位进行的。仲裁开始时，对于每一位数据，SCL为高时，每个主机都检测SDA上的数据是否和自己发送的数据相同。可能需要进行多个位（bit）的比较，主机才开始检测到SDA上数据和自己发送的不一致。实际上，只要SDA上的数据和主机发送的数据一致，这些主机就可以将数据一致发送下去。当主机发送为HIGH，检测SDA上电平却为LOW，那么该主机就在仲裁中失去主控权，并将其SDA输出关闭。余下的主机获得总线控制权并继续数据的传输。如图，当主机1在检测到SDA数据和它自身的输出DATA1不一致时，将自动关闭DATA1的输出，停止向总线上发送数据。



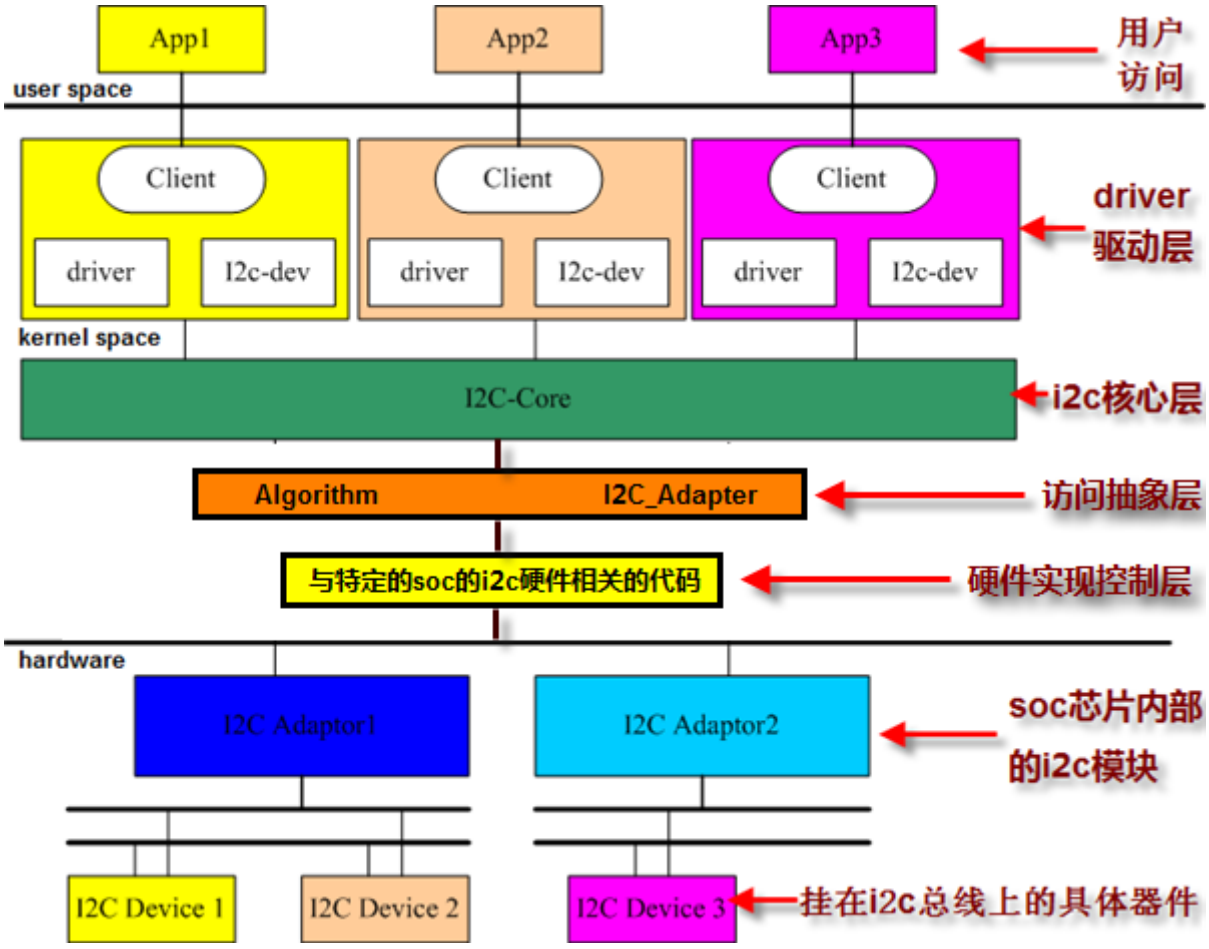
由此可见，在仲裁过程中胜出的主机是没有丢失数据的。在仲裁中失去总线控制权的主机在本次字节传输结束后继续产生时钟，并在总线空闲时开始上次数据的重传。如果同一个器件可以工作在主从两种模式，它在仲裁过程中失去总线控制权，那么有可能是仲裁胜出的主机将要访问该器件，该器件应该立即切换到从机模式。

从上面的原理分析可知，I2C不存在核心主机，是没有优先级的概念的，谁先发送低电平谁就会掌握对总线的控制权。

二、Linux I2C体系结构

Linux的I2C体系结构分为I2C核心、I2C总线驱动、I2C设备驱动三部分。

- I2C核心(i2c-core-base.c)提供了I2C总线驱动和设备驱动的注册、注销方法、I2C通信方法(algorithm)上层的，与具体硬件无关的代码以及探测设备的上层代码等,代码由内核提供。
- I2C总线驱动(i2c-designware-platdrv.c/hobot-i2c.c)是对I2C适配器实现，主要包含I2C适配器数据结构i2c_adapter、I2C适配器的Algorithm数据结构i2c_algorithm和控制I2C适配器产生通信信号的函数。一般由芯片厂商会提供。
- I2C设备驱动(x2-camera.c/at24.c)是对I2C设备端的实现，一般挂在适配器上，通过适配器与CPU交换数据。主要包含数据结构i2c_driver和i2c_client.代码需要用户自己写。



2.1 分析i2c_adapter、i2c_algorithm、i2c_driver和i2c_client这4个数据结构的作用与关系

- **i2c_adapter与i2c_algorithm**

adapter对应物理上的一个适配器硬件，而algorithm对应一套通信方法。adapter需要algorithm提供的通信函数来控制适配器产生特定的访问周期，i2c_adapter结构体中喊喊i2c_algorithm的指针。

i2c_algorithm中的关键函数master_xfer函数用于产生I2C访问周期需要的信号，以i2c_msg为单位，包含I2C传输的方向、地址、缓冲区、缓冲区长度等信息。

```
struct i2c_adapter {
    struct module *owner;
    unsigned int class;       /* classes to allow probing for */
    const struct i2c_algorithm *algo; /* the algorithm to access the bus */
    void *algo_data;

    const struct i2c_lock_operations *lock_ops;
    struct rt_mutex bus_lock;
    struct rt_mutex mux_lock;

    int timeout;            /* in jiffies */
    int retries;
    struct device dev;      /* the adapter device */
    struct completion dev_released;
};

struct i2c_algorithm {
    int (*master_xfer)(struct i2c_adapter *adap, struct i2c_msg *msgs,
               int num);
    int (*smbus_xfer) (struct i2c_adapter *adap, u16 addr,
               unsigned short flags, char read_write,
               u8 command, int size, union i2c_smbus_data *data);
};
```

- **i2c_driver与i2c_client**

i2c_driver对应一套驱动方法，主要函数是probe、remove函数等，而i2c_client对应真实的物理设备。每个I2C设备都要一个i2c_client来描述，i2c_driver与i2c_client之间的对应关系是一对多，一个i2c_driver可以支持多个同类型的i2c_client。i2c_client的信息以前通常在BSP的板级文件中通过i2c_board_info来填充，现在通常在设备树文件下写，写在对应的适配器的子模块中。

```
struct i2c_client {
    unsigned short flags;       /* div., see below      */
    unsigned short addr;        /* chip address - NOTE: 7bit    */
    char name[I2C_NAME_SIZE];
    struct i2c_adapter *adapter;    /* the adapter we sit on    */
    struct device dev;      /* the device structure     */
    int irq;            /* irq issued by device     */
    struct list_head detected;
};
```

- **i2c_adapter与i2c_client**

三、编写I2C驱动的完整流程