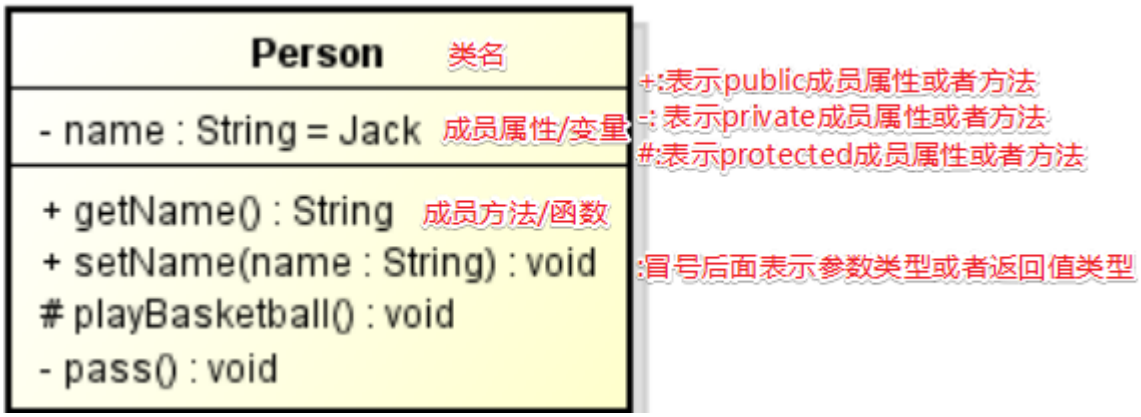
UML类图说明

- 参考<<UML类图.pdf>>文件说明

```
+: 表示public成员属性或者方法
-: 表示private成员属性或者方法
'#': 表示protected成员属性或者方法
```

冒号后面表示参数类型或者返回值类型



数据传输过程中存在的问题:

报文编解码函数说明

- 常用API函数

结构体定义:

```
typedef struct ITCAST_ANYBUF_ {

    unsigned char     *pData;    //编码之后的字符串内容
    ITCAST_UINT32     dataLen;   //编码之后字符串的长度

    ITCAST_UINT32     unusedBits;   /* for bit string */
    ITCAST_UINT32     memoryType;
    ITCAST_UINT32     dataType;   //数据类型
    struct ITCAST_ANYBUF_ *next;  //指向链表的下一个节点的指针
    struct ITCAST_ANYBUF_ *prev;  //指向链表的上一个节点的指针

}ITCAST_ANYBUF;
```

相关函数说明:

整形数据的编码函数:

```
    ITCAST_INT DER_ItAsn1_WriteInteger(ITCAST_UINT32 integer, ITASN1_INTEGER **ppDerInteger);
```

函数说明: 将整形数进行编码

参数说明：
    --integer：输入参数，要编码的整形值
    --ppDerInteger：输出参数，应传入一个一级指针的地址

整形数据的解码函数：
ITCAST_INT DER_ItAsn1_ReadInteger(ITASN1_INTEGER *pDerInteger, ITCAST_UINT32 *pInteger);
函数说明:对整形值进行解码
参数说明：
    --pDerInteger：输入参数，要解码的节点
    --pInteger：输出参数，保存解码之后的整形值

数据类型转换函数:将char *------>ITCAST_ANYBUF
ITCAST_INT DER_ITCAST_String_To_AnyBuf(ITCAST_ANYBUF **pOriginBuf, unsigned char * strOrigin, int strOriginLen);
函数说明：将char *转换为ITCAST_ANYBUF
参数说明：
    -- pOriginBuf：输出参数，保存char *类型数据转换为ITCAST_ANYBUF类型之后的数据，使用的时候应该传递一个一级指针的地址
    -- strOrigin：要转换的char *的字符串
    -- strOriginLen：strOrigin字符串的长度，这个长度应该包含字符串结束标记\0，strlen(strOrigin)+1

字符串类型数据编码函数：
ITCAST_INT DER_ItAsn1_WritePrintableString(ITASN1_PRINTABLESTRING *pPrintString, ITASN1_PRINTABLESTRING **ppDerPrintString);
函数说明：将ITASN1_PRINTABLESTRING进行编码，若是对字符串数据进行编码，应该先调用DER_ITCAST_String_To_AnyBuf将char * 转换为ITASN1_PRINTABLESTRING，然后在调用该函数进行编码操作.
参数说明：
    --pPrintString：输入参数，表示要编码的数据
    --ppDerPrintString：输出参数，表示编码之后数据保存的位置，应该传入一个一级指针的地址

字符串类型数据数据解码函数：
ITCAST_INT DER_ItAsn1_ReadPrintableString(ITASN1_PRINTABLESTRING *pDerPrintString,ITASN1_PRINTABLESTRING **ppPrintString);
函数说明:解码字符串类型数据，解码pDerPrintString数据到ppPrintString中，最后通过ppPrintString->pData获取解码之后的字符串，解码之后的字符串长度为ppPrintString->dataLen
参数说明：
    --pDerPrintString：输入参数，要解码的数据
    --ppPrintString：输出参数，解码后保存数据的内存地址

上述几个函数将需要编码的数据组成一个链表，头结点用phead表示，然后再将phead指向的这个链表进行序列化，最后序列化成字符串.

将链表序列化成字符串函数：
ITCAST_INT DER_ItAsn1_WriteSequence(ITASN1_SEQUENCE *pSequence, ITCAST_ANYBUF **ppDerSequence);
函数说明:将整个链表进行序列化，并将最后结果保存在第二个参数执行的内存中
参数说明：
    --pSequence：输入参数，链表头节点指针
    --ppDerSequence：输出参数，保存序列化之后的数据，该参数应该是一个一级指针的地址
    使用方法：
        第一步：将整个链表(有pHead头指针指向的节点)编码成一个ITCAST_ANBYBUF类型
            ITCAST_ANYBUF *pTemp;

```
            DER_ItAsn1_WriteSequence(pHead, &pTemp);
        第二步：pTemp指向的ITCAST_ANYBUF类型的pData作为编码之后的数据，将dataLen作为编码数据的长度.
            编码之后的数据：pTemp->pData;
            编码之后的数据长度：pTemp->dataLen;

反序列化函数：
ITCAST_INT DER_ItAsn1_ReadSequence(ITCAST_ANYBUF *pDerSequence, ITASN1_SEQUENCE **ppSequence);
函数说明:对字符串数据进行反序列化，在调用这个函数之前应该先调用DER_ITCAST_String_To_AnyBuf函数将char *
转换成ITCAST_ANYBUF类型.
参数说明：
    --pDerSequence：传入参数，表示要反序列化的数据
    --ppDerSequence：传出参数，保存反序列化之后的数据
    使用方法：
        例如要解码的数据为char *inData;
        第一步:将char *----->ITCAST_ANYBUF类型
            DER_ITCAST_String_To_AnyBuf(&pTemp, inData, inLen);
        第二步：解码数据到链表中，获得链表头指针pHead
            DER_ItAsn1_ReadSequence(pTemp, &phead);
        第三步：释放pTemp指向的内存
            DER_ITCAST_FreeQueue(pTemp);

-------------------------------------------------------------------------------------------------
关于编码和解码的函数，还有两个比较简单的函数：
1 编码函数：
int EncodeChar(char *pData, int dataLen, ITCAST_ANYBUF **outBuf);
函数说明：将字符串类型数据进行编码，保存到outBuf指向的内存中
参数说明：
    --pData：输入参数，要编码的数据
    --dataLen：输入参数，pData数据的长度，通常应该加上字符串结束标记\0，strlen(pData)+1
    --outBuf：输出参数，编码之后数据保存的内存地址

2 解码函数：
int DecodeChar(ITCAST_ANYBUF *inBuf, char **Data, int *pDataLen);
函数说明：对ITCAST_ANYBUF数据解码
参数说明：
    --inBuf：输入参数，要解码的数据
    --Data：输出参数，保存解码之后的数据，一般可以char *p，然后将&p作为参数使用.
    --pDataLen：输出参数，解码之后的数据的长度

注意：Data这个参数只能是char *指针，数组不行，如char name[64]; 因为name是数组，是数组名是常量，
      所以不能作为二级指针的传出参数使用.
-------------------------------------------------------------------------------------------------
释放内存函数：
ITCAST_INT DER_ITCAST_FreeQueue(ITCAST_ANYBUF *pAnyBuf)
函数说明：释放一个序列(链表)，pAnyBuf为链表的头结点
函数参数：
    --pAnyBuf：输入参数，链表头节点指针
```

解决编译报错的问题

在项目名称上点击右键, 选择**[属性]**菜单, 然后选择**[C/C++]**下的**[预处理器]**, 最后在右侧的**[预处理器定义]** 处添加_CRT_SECURE_NO_WARNINGS(点击最右侧的下拉框, 然后点击**[编辑]**, 并将***__CRT_SECURE_NO_WARNINGS***添加到最后一行即可)

练习使用itcast_asn1_der.c中的接口函数:

下面是teacher.c的代码

```
/*
    typedef struct _Teacher
    {
        char name[64];
        int age;
        char *p;
        long plen;
    }Teacher;
*/
#include "teacher.h"
#include "itcast_asn1_der.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

函数说明：给Teacher结构体变量编码
参数说明：
    p:       输入参数，需要在外部先给Teacher结构体变量赋值
    outData: 输出参数，该指针指向的内存由该函数内存分配
    outlen:  输出参数，保存编码之后的数据长度
int encodeTeacher(Teacher * p, char ** outData, int * outlen)
{
    ITCAST_ANYBUF* temp = NULL;
    ITCAST_ANYBUF* head = NULL;
    ITCAST_ANYBUF* next = NULL;

    //编码name
    EncodeChar(p->name, strlen(p->name)+1, &head);
    next = head;

    //编码age
    DER_ItAsn1_WriteInteger(p->age, &next->next);
    next = next->next;

    //编码p
    EncodeChar(p->p, strlen(p->p)+1, &next->next);
    next = next->next;

    //编码plen
    DER_ItAsn1_WriteInteger(p->plen, &next->next);

    //序列化
    DER_ItAsn1_WriteSequence(head, &temp);
```

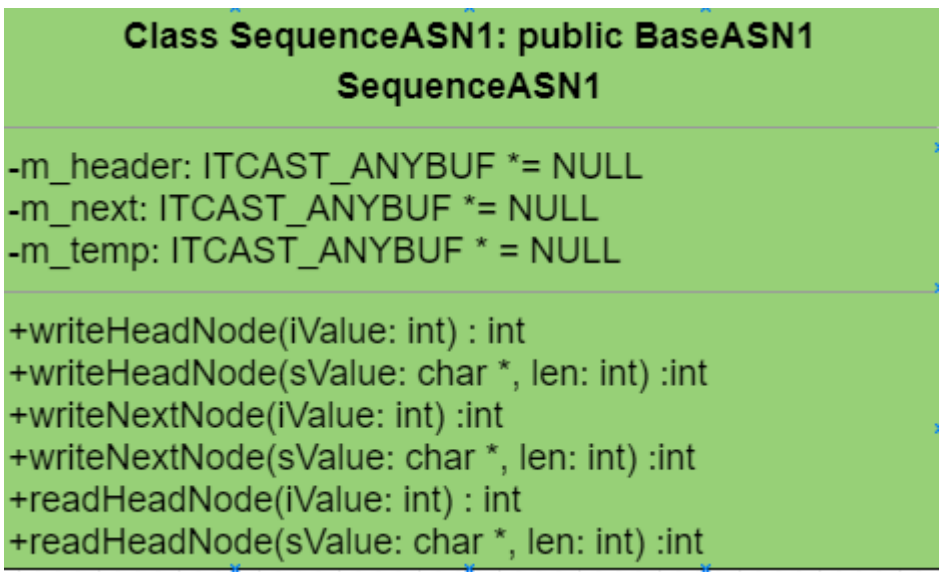
```
    //输出参数赋值
    *outData = temp->pData;
    *outlen = temp->dataLen;

    //释放内存
    DER_ITCAST_FreeQueue(head);

    return 0;
}
```

函数说明：解码inData数据到Teacher结构体指针指向的内存中
参数说明：
    inData：输入参数，要解码的字符串
    inLen： 输入参数，要解码的字符串的长度
    p：   输出参数，保存解码之后的数据

```
int decodeTeacher(char * inData, int inLen, Teacher ** p)
{
    ITCAST_ANYBUF* temp = NULL;
    ITCAST_ANYBUF* head = NULL;
    ITCAST_ANYBUF* next = NULL;   //作为遍历链表的节点，从head开始遍历

    Teacher* pt = (Teacher*)malloc(sizeof(Teacher));

    //将char *---->ITCAST_ANYBUF类型
    DER_ITCAST_String_To_AnyBuf(&temp, inData, inLen);

    //将字节流程解码到head指向的链表中
    DER_ItAsn1_ReadSequence(temp, &head);
    next = head;

    //解码name
    //name也是char *，但是name不能用DecodeChar进行解码，原因是DecodeChar的第二个
    //参数是传出参数，而name是数组，数组名是常量指针,不能作为传出参数使用.
    int len = 0;
    DER_ItAsn1_ReadPrintableString(next, &temp);
    memcpy(pt->name, temp->pData, temp->dataLen);
    DER_ITCAST_FreeQueue(temp);
    next = next->next;

    //解码age
    DER_ItAsn1_ReadInteger(next, &pt->age);
    next = next->next;

    //解码p
    len = 0;
    DecodeChar(next, &pt->p, &len);
    next = next->next;

    //解码plen
    DER_ItAsn1_ReadInteger(next, &pt->plen);

    //输出参数赋值
```

```
    *p = pt;

    //释放内存

    return 0;
}

void freeTeacher(Teacher ** p)
{
    // 1. (*p)->p
    // 2. free (*p)
    if (*p != NULL)
    {
        if ((*p)->p != NULL)
        {
            free((*p)->p);
            (*p)->p = NULL;
        }

        free(*p);
        *p = NULL;
    }
}
```

UML绘图工具: draw.io



将c源代码封装成C++类代码:

1 将宏定义------>常量const

- 主要是整形和字符串
- 一些连续的整形值可以改写成枚举类型

2 宏函数

- 简单的宏函数可以改写成内联函数
- 如果比较复杂, 可以改写成类的成员函数

3 若成员函数都用到了某个变量, 可以将这个变量设置成类的成员变量

4 通过类的访问控制权限控制类的某些成员可以对外部访问

- 一般只有public成员可以对外界访问, 不被外界访问的可以设置成private成员或者protected成员

基础类BaseASN1类解读

面向对象的思想:

- 封装
  - 将属性和方法封装在一个类中
  - 通过访问控制来限制对类中成员的访问
- 继承
  - 子类可以继承父类的成员
- 多态
  - 有继承关系
  - 父类中成员函数声明为virtual的, 且子类重写父类的成员函数
  - 有父类指针指向子类对象, 通过父类指针调用具体的子类对象的方法