struct 协议生成工具

版本：1.0.4

版本历史：

版本	说明
1.0.1	
1.0.2	修改中文版Windows系统下字符集设置错误导致生成文件中中文注释乱码的问题。
1.0.3	支持变长struct的list，例如：`Goos[] goodsList;`
1.0.4	支持位段；支持嵌套定义struct；支持尾部注释；encode/decode的参数类型改为java.io.OutputStream/java.io.InputStream。

struct是一个专门针对Java程序与C语言开发程序通讯的应用层协议生成工具，网络通讯字节流采用C语言开发习惯的方式，方便C语言程序进行decode处理。

struct规定了一套简单的语法来定义协议，协议定义存放在一个文本文件中，利用struct工具可以编译协议定义文件，生成Java POJOs或者HTML格式的协议说明文档，以及C语言的头文件。

struct让开发者从雷同、繁琐、易错的协议encode/decode编码中脱身，站在更高的层次上应用层维护通讯协议。

1. struct协议定义语法

1.1 支持的数据类型

> 以下说明中`C语言映射`都使用C语言标准头文件`stdint.h`中的定义。

struct和bitfield(位段)

协议定义的基本单元是struct，一个struct与C语言的struct类似，可以包含多个字段，在Java中是一个POJO。字段的数据类型可以是基本数据类型、定长字符串以及struct和bitfield。

从1.0.4开始支持bitfield(位段)，对字段数据的定义可以细化到bit(位)的级别。例如下面定义了一个位段Color，其中用头4位表示颜色的透明度，后4位表示颜色的值。

```
// 颜色
// 位0-3 颜色透明度
// 位4-7 颜色值
bitfield Color {
    transparency:4; // 透明度, 0为不透明, 15为全透明。
    color:4; // 颜色值
}
```

bitfield的所有字段都是无符号整数型(unsigned int)，取值范围由字段所占的位数决定，上例中的4位，取值范围是0~15。

bitfield在encode/decode时，会存储在字节中，如果bitfield中各字段的位数之和不能被8整除，则自动扩充位数到8的整数倍。

基本数据类型

类型	字节数	Java语言映射	C语言映射
byte	1	byte	unsigned char
int8	1	byte	int8_t
uint8	1	byte	uint8_t
short	2	short	short
int16	2	short	int16_t
uint16	2	short	uint16_t
int	4	int	int
int32	4	int	int32_t
uint32	4	int	uint32_t
long	8	long	int64_t
int64	8	long	int64_t
uint64	8	long	uint64_t
float	4	float	float
double	8	double	double

字符串

类型	字节数	Java语言映射	C语言映射
string	N	java.lang.String	char[N]

仅支持固定长度的字符串。

嵌套struct

一个struct定义中可以嵌套包含其他struct，可以包含单个struct，也可以包含struct的数组。

数组

基本数据类型和嵌套struct都可以定义为数组。**不支持定义字符串类型的数组。**

1.2 协议定义文件

协议定义存放在文本文件中，建议以 `.struct` 作为后缀名，以下协议定义文件简称为 `struct文件` 。**可以以任何合法的操作系统后缀名作为协议定义文件的后缀名， `.struct` 只是建议不是强制。**

下例是一个示例定义 `demo.struct` 内容，**以下的说明中都以此文件内容为例。**

```
1. ##########################################
2. # 商品陈列协议定义
3. # 版本：1.0.1
4. # 修改人：Tony
5. ##########################################
6. import "base.struct";
7. import "extension.struct";
8.
9. /* 商品 */
10. struct Goods {
11.     // 商品ID
12.     int id = 1;
13.
14.     // 商品名称
15.     string[64] name = "烟台苹果";
16.
17.     // 单价(每500克)
18.     double unitPrice = 15.05;
19. }
20.
21. // 货架
22. struct Shelf {
23.     // 货架编号
24.     int id;
25.
26.     // 货架上陈列的商品数量
27.     int displayedGoodsNum;
28.
29.     // 货架上陈列的商品
30.     Goods[displayedGoodsNum] displayedGoods;
31. }
32.
33. // 礼品
34. struct Gift {
35.     // 礼品ID
36.     int id;
37.
38.     // 礼品赠送的商品
39.     Goods goods;
40. }
```

包含其他struct文件

第6,7行，一个struct文件可以用一个或多个 `import` 指令包含其他struct文件，对于大型的、功能模块较多的系统，开发者可以根据需要把协议定义分开放在多个struct文件中，在最顶层的struct文件中，用import指令把它们都包含进来。

包含的协议定义文件路径名应用双引号括住，可以指定路径，例如:

```
# 指定绝对路径
import "/workspace/project/protocol/base.struct";

# 指定相对于当前协议定义文件所在目录的相对路径
import "../model1/base.struct";

# 不指定路径，则表示与当前协议定义文件在同一目录
import "base.struct";
```

以下是被包含的 `base.struct` 文件的内容， `base.struct` 演示基本数据类型。 `extension.struct` 演示**位段**和**struct中嵌套定义新的struct**，这部分内容会专门在 `1.3` 章节详细介绍。

base.struct

```
// 演示基本数据类型
struct Base {
    // 字符串类型,长度16字节。
    string[16] stringValue = "Hello World!";

    // 字节(byte)类型
    byte byteValue = 0;

    // 短整型
    short shortValue = 1;

    // 整型
    int intValue = 2;

    // 长整型
    long longValue = 3;

    // 浮点数
    float floatValue = 1.11;

    // 双精度浮点数
    double doubleValue = 3.1415926;
}
```

注释

struct支持三种注释格式，井号 `#` 和双斜杠 `//` 都是单行注释， `/* ... */` 之间可以注释多行的内容。

井号 `#` 单行注释在编译时会被完全忽略，而 `//` 和 `/* ... */` 注释都会被记录并输出到最终创建的

Java POJO class中，也被用于生成HTML格式的协议定义文档。

定义一个struct

第9~19行定义了一个简单的struct，struct定义应以 `struct StructName {` 开始， `}` 结束。

```
 9. /* 商品 */
10. struct Goods {
11.     // 商品ID
12.     int id = 1;
13.
14.    // 商品名称
15.    string[64] name = "烟台苹果";
16.
17.    // 单价(每500克)
18.    double unitPrice = 15.05;
19. }
```

第10行定义这个struct的名字为 `Goods` 。第11~18行给Goods这个struct定义了三个字段，数据类型分别为整数(int)、长度64字节的字符串(string[64])和浮点数(double)。

指定字段的初始值

可以为**基本数据类型**和**字符串**类型的字段指定初始值，如第11~18行所示。

每个struct最终都会被生成一个Java POJO，没有指定初始值的基本数据类型字段，在POJO实例化时都会被赋予初值 `0` ，没有指定初始值的字符串类型字段，在POJO实例化时会被赋予初值 `null` 。

如果在struct文件中指定了初始值，则在POJO实例化时，会赋予struct文件中定义的初始值。

引用其他struct或者struct的数组

第21~31行定义了一个复合的struct `Shelf` ，这个struct中包含了另外一个struct `Goods` 的变长数组，数组元素个数引用另外一个字段 `displayedGoodsNum` 。

```
// 货架
struct Shelf {
    // 货架编号
    int id;

    // 货架上陈列的商品数量
    int displayedGoodsNum;

    // 货架上陈列的商品
    Goods[displayedGoodsNum] displayedGoods;
}
```

这个 `Goods[displayedGoodsNum] displayedGoods` 字段在生成的Java POJO中会被定义为 `java.util.List<Goods> displayedGoods;` ，因为List是可变长度的，所以开发者在编码时，只需要往这个List中添加元素，无需关注元素的个数，在最终encode为网络字节流时，struct会计算List的个数，自动给 `displayedGoodsNum` 这个字段赋值。

> **注意**: 作为变长数组下标的字段会在struct encode成网络字节流时被赋值，因此代码中对该字段的赋值无意义。

当然，struct可以简单的只包括另外一个struct，例如第33~40行定义的struct `Gift` 。

1.3 struct内部嵌套定义新的struct

从1.0.4开始支持在struct内部嵌套定义新的struct和bitfield。

```
#
# 演示位段和内嵌定义struct
#

// 颜色
// 位0-3 颜色透明度
// 位4-7 颜色值
bitfield Color {
    transparency:4; // 透明度, 0为不透明, 15为全透明。
    color:4; // 颜色值
}

// 图层
struct ImageLayer {
    // 坐标
    struct Point {
        int x; // x轴坐标
        int y; // y轴坐标
    } leftTop; // 左上角坐标

```

```
21.     // 尺寸
22.     struct Size {
23.         int width; // 宽度
24.         int height; // 高度
25.     } size; // 图层大小
26.
27.     // 图层颜色
28.     Color color;
29. }
30.
31. // 任务
32. struct Task {
33.     // 任务控制字
34.     bitfield CtrlAndSID {
35.         ctrl:3; // 任务类型标识
36.         SID:4; // Session ID
37.         reserved:17; // 保留
38.     } ctrlAndSID; // 任务控制字和SessionID
39.
40.     // 混杂模式
41.     struct MixedGroup {
42.         uint16 power; // 发射功率
43.         uint8 channel; // 通讯信道;
44.         uint16 num; // 后续数据包(Packet)个数
45.         // 数据包
46.         struct Packet {
47.             byte[4] eslId; // ESL ID
48.             byte[26] data; // 数据
49.         } [num] packets; // 由num指定个数的多个数据包
50.     } mixedGroup;
51.
52.     // CCITT-CRC16
53.     uint16 crc16;
54. }
```

第15~25行，在ImageLayer的定义中，又定义了Point和Size两个新的struct。

> - 在作用域上，所有struct和bitfield都是平级的。嵌套定义的struct和bitfield仍然可以在其他struct定义时进行引用。
> - 一个struct或者bitfield只能被定义一次。

可以不断嵌套定义，第46~49行，Task中嵌套定义了MixedGroup，MixedGroup中又嵌套定义了Packet。

2. 编译struct文件生成代码

struct发行一个 `struct-<version>.jar` 的可执行jar文件，从命令行运行这个jar可以编译struct文件，生

成Java代码、HTML文档以及C语言头文件。

2.1 命令行参数

```
$ java -jar build/libs/struct-1.0.1.jar  -h
usage: StructCompiler [-h] -t {java,c,html} [-e ENCODING] [-p PACKAGE]
                      -d DIRECTORY -f FILE

Compile C style struct to generating Java C or HTML.

optional arguments:
  -h, --help             show this help message and exit
  -t {java,c,html}, --target {java,c,html}
                         Specify target language
  -e ENCODING, --encoding ENCODING
                         Specify charset  encoding  of  struct  description
                         file and generated files (default: utf8)
  -p PACKAGE, --package PACKAGE
                         Specify package of generated Java classes
  -d DIRECTORY, --directory DIRECTORY
                         Specify destination directory
  -f FILE, --file FILE   Specify struct description file
```

`-h, --help` 显示命令行参数说明。

`-t {java,c,html}, --target {java,c,html}` **必选**，指定生成Java代码、HTML文档还是C语言头文件。

`-e ENCODING, --encoding ENCODING` **可选**，指定生成代码(文档)的字符集编码，缺省UTF8。

`-p PACKAGE, --package PACKAGE` **可选**，仅生成Java代码时有用，指定生成Java POJO类的package。

`-d DIRECTORY, --directory DIRECTORY` **必选**，指定生成文件存放到哪个目录下。如果指定生成Java代码，同时-p选项指定了package，则Java代码会生成到这个目录下的package路径中。

`-f FILE, --file FILE` **必选**，指定要编译的struct文件。

2.2 生成Java代码

编译 `demo.struct` 生成Java代码。

```
> java -jar struct-1.0.1.jar -t java -e utf8 -d src/test/java -p demo.test -f test
/demo.struct
```

`-t java` 指定生成Java代码， `-e utf8` 指定生成代码源文件的字符集编码为utf8， `-d src/test/java` 指定源代码目录为src/test/java， `-p demo.test` 指定生成Java POJO类的package为demo.test。

命令运行后，会在 **`src/test/java/demo/test`** 目录下生成5个Java源代码文件： `Struct.java` , `Base.java` , `Goods.java` , `Shelf.java` , `Gift.java` 。

其中 `Base.java` , `Goods.java` , `Shelf.java` , `Gift.java` 都是在 `demo.struct` 和它import的 `base.struct` 中定义的struct。这些类的完整代码见附录。

`Struct.java` 这个类是所有struct类的基类，定义了struct类的基本方法，包括网络字节序和encode/decode方法。

2.3 生成HTML格式的协议说明文档

```
> java -jar struct-1.0.1.jar -t html --encoding=utf8  -d test/html  -f test/demo.s
truct
```

`-t html` 指定了生成HTML格式文档， `-d test/html` 指定把HTML文档生成到test/html目录下。

命令运行后，会在 `-d` 选项指定的test/html目录下生成三个HTML文件： `index.html` , `Navigation.html` , `Struct.html` 。用浏览器打开 `index.html` 即可查看文档内容。

2.4 生成C语言头文件

```
> java -jar struct-1.0.1.jar --target=c --encoding=utf8 --directory=test  --file=t
est/demo.struct
```

`--target=c` 指定生成C语言头文件， `--directory=test` 指定头文件生成到test目录下。

2.5 编译时刻错误

如果struct中存在语法错误，则编译时会报告错误的文件和行数。例如以下的struct定义中， `struct` 关键字写错。

```
/* 商品 */
strut Goods {
    // 商品ID
    int id = 1;

   // 商品名称
   string[64] name = "烟台苹果";

   // 单价(每500克)
   double unitPrice = 15.05;
}
```

编译时会得到如下的错误信息。

```
Compiling /path/to/demo/test/demo.struct
line 10:0 extraneous input 'strut' expecting {'import', 'struct'}
Syntax error.
```

3. 生成Java代码的使用

生成的Java代码完整内容见附录。每个struct会被生成一个POJO，struct的每个字段被生成为POJO的属性，并提供了getter/setter方法。以下是针对 `demo.struct` 中定义的struct的使用示例代码：

```
import java.io.*;
import java.util.List;

public class Demo {
    public static void main(String[] args) throws Exception {
        // 设定struct序列化的字节序为Little-Endian
        Struct.byteOrder = Struct.LITTLE_ENDIAN;

        // 创建对象
        Shelf shelf = new Shelf();

        // 为基本数据类型属性赋值
        shelf.setId(8899);

        // 为List类型的属性赋值
        List<Goods> goodsList = shelf.getDisplayedGoods();
        goodsList.add(new Goods());
        goodsList.add(new Goods());

        // 把shelf对象encode成为字节数组
        byte[] bytes = encode(shelf);

```

```
23.         // decode字节数组
24.         Shelf newone = new Shelf();
25.         decode(newone, bytes);
26.     }
27.
28.     /**
29.      * 把指定struct对象encode成为字节数组
30.      * @param struct 指定的struct对象
31.      * @return struct对象encode出的字节数组
32.      * @throws IOException
33.      */
34.     private static byte[] encode(Struct struct) throws IOException {
35.         ByteArrayOutputStream baos = new ByteArrayOutputStream();
36.         DataOutputStream output = new DataOutputStream(baos);
37.         struct.encode(output);
38.         return baos.toByteArray();
39.     }
40.
41.     /**
42.      * 把字节流decode到指定的struct对象中
43.      * @param struct 指定的struct对象
44.      * @param bytes 字节流
45.      * @throws IOException
46.      */
47.     private static void decode(Struct struct, byte[] bytes) throws IOException
 {
48.         DataInputStream input = new DataInputStream(new ByteArrayInputStream(b
ytes));
49.         struct.decode(input);
50.     }
51. }
52.
```

3.1 对象创建和属性赋值

每个struct生成的POJO类都有缺省无参数的构造方法，如第10行、第17行、第18行所示，用缺省的构造方法创建struct对象。

对于List类型的属性，不提供setter方法，开发者只能通过getter方法获得List属性，然后再向其中添加元素。

3.2 struct对象encode/decode

每个struct类都有encode和decode方法。encode方法把struct对象及其递归包含的所有其他struct对象序列化到实现了DataOutput的对象中。decode方法从实现了DataInput的对象中读出struct的内容，包括其递归包含的其他struct对象内容，装配成一个struct对象。

第28~39行演示了如何encode一个struct对象成为字节数组，**struct中包含的每个字段依照定义顺序从上到下依次序列化到字节流中，字段与字段之间没有填充，紧邻着存放。**

第41~50行演示了如何把字节数组中的内容decode到struct对象中。

3.3 指定encode/decode的字节序

第4行，给Struct类的静态属性 `byteOrder` 赋值可以指定encode/decode时的字节序，缺省不赋值时为 `LITTLE_ENDIAN` 字节序。还可以设置为 `BIG_ENDIAN` 。

4. 附录

Base.java

```
/*******************************************************************************
 * Base.java
 * GENERATED BY StructCompiler, DON'T MODIFY MANULLY.
 * Generated Time: Mon Jun 27 11:03:11 CST 2016
 */
package demo.test;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

/**
 * struct Base
 *
 * 演示基本数据类型
 */
public class Base extends Struct {
    // 字符串类型,长度16字节。
    private String stringValue;

    // 字节(byte)类型
    private byte byteValue;

    // 短整型
    private short shortValue;

    // 整型
    private int intValue;

    // 长整型
```

```
    private long longValue;

    // 浮点数
    private float floatValue;

    // 双精度浮点数
    private double doubleValue;

    public Base() {
        this.stringValue = (String) "Hello World!";

        this.byteValue = (byte) 0;

        this.shortValue = (short) 1;

        this.intValue = (int) 2;

        this.longValue = (long) 3;

        this.floatValue = (float) 1.11;

        this.doubleValue = (double) 3.1415926;

    }

    public String getStringValue() {
        return this.stringValue;
    }

    public void setStringValue(String stringValue) {
        this.stringValue = stringValue;
    }

    public byte getByteValue() {
        return this.byteValue;
    }

    public void setByteValue(byte byteValue) {
        this.byteValue = byteValue;
    }

    public short getShortValue() {
        return this.shortValue;
    }

    public void setShortValue(short shortValue) {
        this.shortValue = shortValue;
    }
```

```
public int getIntValue() {
    return this.intValue;
}

public void setIntValue(int intValue) {
    this.intValue = intValue;
}

public long getLongValue() {
    return this.longValue;
}

public void setLongValue(long longValue) {
    this.longValue = longValue;
}

public float getFloatValue() {
    return this.floatValue;
}

public void setFloatValue(float floatValue) {
    this.floatValue = floatValue;
}

public double getDoubleValue() {
    return this.doubleValue;
}

public void setDoubleValue(double doubleValue) {
    this.doubleValue = doubleValue;
}

public int calcSize() {
    int __size = 0;

    __size += 16;

    __size += 1 * 1;

    __size += 2 * 1;

    __size += 4 * 1;

    __size += 8 * 1;

    __size += 4 * 1;
```

```
        __size += 8 * 1;

        return __size;
    }

    public void encode(DataOutput dos) throws IOException {
        writeString(this.stringValue, 16, dos);

        write(dos, this.byteValue, byte.class);

        write(dos, this.shortValue, short.class);

        write(dos, this.intValue, int.class);

        write(dos, this.longValue, long.class);

        write(dos, this.floatValue, float.class);

        write(dos, this.doubleValue, double.class);

    }

    public Struct decode(DataInput dis) throws IOException {
        this.stringValue = readString(dis, 16);

        this.byteValue = read(dis, byte.class);

        this.shortValue = read(dis, short.class);

        this.intValue = read(dis, int.class);

        this.longValue = read(dis, long.class);

        this.floatValue = read(dis, float.class);

        this.doubleValue = read(dis, double.class);

        return this;
    }
}
```

Goods.java

```
/**********************************************************************
 * Goods.java
 * GENERATED BY StructCompiler, DON'T MODIFY MANULLY.
 * Generated Time: Mon Jun 27 11:03:11 CST 2016
```

```
 */
package demo.test;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

/**
 * struct Goods
 *
 * 商品
 */
public class Goods extends Struct {
    // 商品ID
    private int id;

    // 商品名称
    private String name;

    // 单价(每500克)
    private double unitPrice;

    public Goods() {
        this.id = (int) 1;

        this.name = (String) "烟台苹果";

        this.unitPrice = (double) 15.05;

    }

    public int getId() {
        return this.id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getUnitPrice() {
```

```
        return this.unitPrice;
    }

    public void setUnitPrice(double unitPrice) {
        this.unitPrice = unitPrice;
    }

    public int calcSize() {
        int __size = 0;

        __size += 4 * 1;

        __size += 64;

        __size += 8 * 1;

        return __size;
    }

    public void encode(DataOutput dos) throws IOException {
        write(dos, this.id, int.class);

        writeString(this.name, 64, dos);

        write(dos, this.unitPrice, double.class);

    }

    public Struct decode(DataInput dis) throws IOException {
        this.id = read(dis, int.class);

        this.name = readString(dis, 64);

        this.unitPrice = read(dis, double.class);

        return this;
    }
}
```

Shelf.java

```
/*******************************************************************************
 * Shelf.java
 * GENERATED BY StructCompiler, DON'T MODIFY MANULLY.
 * Generated Time: Mon Jun 27 11:03:11 CST 2016
 */
package demo.test;
```

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

/**
 * struct Shelf
 *
 * 货架
 */
public class Shelf extends Struct {
    // 货架编号
    private int id;

    // 货架上陈列的商品数量
    private int displayedGoodsNum;

    // 货架上陈列的商品
    private java.util.List<Goods> displayedGoods;

    public Shelf() {
        this.id = 0;

        this.displayedGoodsNum = 0;

        this.displayedGoods = new java.util.LinkedList<Goods>();

    }

    public int getId() {
        return this.id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getDisplayedGoodsNum() {
        return this.displayedGoodsNum;
    }

    public void setDisplayedGoodsNum(int displayedGoodsNum) {
        this.displayedGoodsNum = displayedGoodsNum;
    }

    public java.util.List<Goods> getDisplayedGoods() {
        return this.displayedGoods;
    }
```

```
    public int calcSize() {
        int __size = 0;

        __size += 4 * 1;

        __size += 4 * 1;

        this.displayedGoodsNum = (int) this.displayedGoods.size();
        for (Struct __struct : this.displayedGoods) {
            __size += __struct.calcSize();
        }

        return __size;
    }

    public void encode(DataOutput dos) throws IOException {
        write(dos, this.id, int.class);

        write(dos, this.displayedGoodsNum, int.class);

        this.displayedGoodsNum = (int) this.displayedGoods.size();
        for (Struct __struct : this.displayedGoods) {
            __struct.encode(dos);
        }

    }

    public Struct decode(DataInput dis) throws IOException {
        this.id = read(dis, int.class);

        this.displayedGoodsNum = read(dis, int.class);

        for (int i = 0; i < this.displayedGoodsNum; ++i) {
            this.displayedGoods.add((Goods) new Goods().decode(dis));
        }

        return this;
    }
}
```

Gift.java

```
/**********************************************************************
 * Gift.java
 * GENERATED BY StructCompiler, DON'T MODIFY MANULLY.
 * Generated Time: Mon Jun 27 11:03:11 CST 2016
```

```
 */
package demo.test;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

/**
 * struct Gift
 *
 * 礼品
 */
public class Gift extends Struct {
    // 礼品ID
    private int id;

    // 礼品赠送的商品
    private Goods goods;

    public Gift() {
        this.id = 0;

        goods = new Goods();

    }

    public int getId() {
        return this.id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public Goods getGoods() {
        return this.goods;
    }

    public int calcSize() {
        int __size = 0;

        __size += 4 * 1;

        __size += this.goods.calcSize();

        return __size;
    }
```

```
    public void encode(DataOutput dos) throws IOException {
        write(dos, this.id, int.class);

        this.goods.encode(dos);

    }

    public Struct decode(DataInput dis) throws IOException {
        this.id = read(dis, int.class);

        this.goods.decode(dis);

        return this;
    }
}
```

demo.h

```
/*****************************************************************************
 * GENERATED BY StructCompiler, DON'T MODIFY MANULLY.
 * Generated Time: Mon Jun 27 12:45:43 CST 2016
 */
#ifndef __DEMO_H__
#define __DEMO_H__

#include <stdint.h>

#define ALIGNED_BY_BYTE __attribute__((packed,aligned(1)))

struct Struct {
    char member[0];
} ALIGNED_BY_BYTE;

/*
 * base
 *
 * 演示基本数据类型
 */
typedef struct base {
    // 字符串类型,长度16字节。
    char string_value[16];

    // 字节(byte)类型
    uint8_t byte_value;

    // 短整型
    int16_t short_value;
```

```
    // 整型
    int32_t int_value;

    // 长整型
    int64_t long_value;

    // 浮点数
    float float_value;

    // 双精度浮点数
    double double_value;

} ALIGNED_BY_BYTE base_t;

/*
 * goods
 *
 * 商品
 */
typedef struct goods {
    // 商品ID
    int32_t id;

    // 商品名称
    char name[64];

    // 单价(每500克)
    double unit_price;

} ALIGNED_BY_BYTE goods_t;

/*
 * gift
 *
 * 礼品
 */
typedef struct gift {
    // 礼品ID
    int32_t id;

    // 礼品赠送的商品
    struct goods goods;

} ALIGNED_BY_BYTE gift_t;

/*
 * shelf
```

```
 *
 * 货架
 */
typedef struct shelf {
    // 货架编号
    int32_t id;

    // 货架上陈列的商品数量
    int32_t displayed_goods_num;

    // 货架上陈列的商品
    struct goods displayed_goods[0];

} ALIGNED_BY_BYTE shelf_t;

#endif /* end of __DEMO_H__ */
```