**protobuf:**

**Messages API:**

// required string name = 1;  
public boolean hasName();  
public String getName();  
// required int32 id = 2;  
public boolean hasId();  
public int getId();  
// optional string email = 3;  
public boolean hasEmail();  
public String getEmail();  
// repeated .tutorial.Person.PhoneNumber phones = 4;  
public List<PhoneNumber> getPhonesList();  
public int getPhonesCount();  
public PhoneNumber getPhones(int index);

**Builder API:**

// required string name = 1;  
public boolean hasName();  
public java.lang.String getName();  
public Builder setName(String value);  
public Builder clearName();  
// required int32 id = 2;  
public boolean hasId();  
public int getId();  
public Builder setId(int value);  
public Builder clearId();  
// optional string email = 3;  
public boolean hasEmail();  
public String getEmail();  
public Builder setEmail(String value);  
public Builder clearEmail();  
// repeated .tutorial.Person.PhoneNumber phones = 4;  
public List<PhoneNumber> getPhonesList();  
public int getPhonesCount();  
public PhoneNumber getPhones(int index);  
public Builder setPhones(int index, PhoneNumber value);  
public Builder addPhones(PhoneNumber value);  
public Builder addAllPhones(Iterable<PhoneNumber> value);  
public Builder clearPhones()

Builder才能新建或者修改相关的数据

通过类方法newBuilder获得bulder对象,设置好新的数据后,调用builder的方法build()

值得注意的是,builder设置数据的方法返回值都是builder所以可以直接使用,就像这样

Person p = Person.newBuilder().setName(“wk”).addPhones(value).build()

解析和序列化:

byte[] toByteArray();: serializes the message and returns a byte array containing its raw bytes.

static Person parseFrom(byte[] data);: parses a message from the given byte array.

void writeTo(OutputStream output);: serializes the message and writes it to an OutputStream.

static Person parseFrom(InputStream input);: reads and parses a message from an InputStream.

**继承:**

不可以多继承,但是可以多重继承.(b是a的父类,c是b的父类)

**final关键字**:表示最终类,不能再被继承

### **super 与 this 关键字**

super关键字：我们可以通过super关键字来实现对父类成员的访问，用来引用当前对象的父类。

this关键字：指向自己的引用。

### **implements关键字**

使用 implements 关键字可以变相的使java具有多继承的特性，使用范围为类继承接口的情况，可以同时继承多个接口（接口跟接口之间采用逗号分隔）。

## 构造器

子类是不继承父类的构造器（构造方法或者构造函数）的，它只是调用（隐式或显式）。如果父类的构造器带有参数，则必须在子类的构造器中显式地通过 super 关键字调用父类的构造器并配以适当的参数列表。

如果父类构造器没有参数，则在子类的构造器中不需要使用 super 关键字调用父类构造器，系统会自动调用父类的无参构造器。

## 抽象类:

虽然抽象不能进行实例.

Father f = new Father(); // 不可行

但是可以声明一个该抽象类,但是传递的值是子类的实例化.

Father f = new Son(); //这个可行

序列化: 就是可以将类实例化,转换成二进制并保存到一个文件中(写入磁盘).需要用的时候可以从文件中读取(反序列化)

需要继承java.io.Serializable (满足可以序列化的条件)

* @NonNull：空值是非法值
* @Nullable：允许使用空值，且必须是预期的值
* @NonNullByDefault：缺少空注释的方法特征符的类型被视为非空。

**Arrays.copyOf() 返回值是新的对象,原数组不变**

****System.arraycopy() 必须传递一个已经生成的目标对象,并会修改目标数组的值****

String 类

isEmpty() 判断是否为空(其实就是length == 0)

但是有可能是string指向null

所以如果要判断String类型的是否为空的话可以用这个判断

String s

(s== null || s.isEmpty())

**自定义类数组:**

在声明了自定义类的数组之后，对每一个数组元素的初始化，都要为其new一个对象出来使得指针指向该对象，Java语言本身是不提供在自定义类数组声明时候自动创建新对象的方式的。

如:

Course[] courses = new Course[3];

courses[0] = new Course();

初始化

Course[ courses = new Course[]{1,2,3};

**JNI:**

JNIEnv

以下代码只起说明作用，真实的实现是在一个文件中，所以实际代码还包含一些预编译技术(用于兼容C与C++)，以及其它一些东西。

C实现方式：

typedef const struct JNINativeInterface\_ \*JNIEnv;

struct JNINativeInterface\_ {

       jclass (JNICALL \*FindClass)(JNIEnv \*env, const char \*name);

       //……

};

解释：

jclass (JNICALL \*FindClass)(JNIEnv \*env, const char \*name);

函数指针FindClass，函数调用方式JNICALL，返回类型jclass，参数JNIEnv\*和const char\*。

调用方式

(\*env)->FindClass(env, "java/lang/String")

C++实现方式：

typedef JNIEnv\_ JNIEnv;

struct JNIEnv\_ {

    const struct JNINativeInterface\_ \*functions;

    jclass FindClass(const char \*name) {

        return functions->FindClass(this, name);

}

//……

};

调用方式

env->FindClass("java/lang/String")

可以注意到，在C实现中JNIEnv是指针类型，而在C++中是类。因此，才会有调用方式的不同。

**Package:**

1. 把功能相似或相关的类或接口组织在同一个包中，方便类的查找和使用。

2、如同文件夹一样，包也采用了树形目录的存储方式。同一个包中的类名字是不同的，不同的包中的类的名字是可以相同的，当同时调用两个不同包中相同类名的类时，应该加上包名加以区别。因此，包可以避免名字冲突。

3、包也限定了访问权限，拥有包访问权限的类才能访问某个包中的类。

把.java文件和.class放在package相同的路径下

**正则表达式:**

* **Pattern 类：**

pattern 对象是一个正则表达式的编译表示。Pattern 类没有公共构造方法。要创建一个 Pattern 对象，你必须首先调用其公共静态编译方法，它返回一个 Pattern 对象。该方法接受一个正则表达式作为它的第一个参数。

str = “www”;

* Pattern p = Pattern.compile(str);
* **Matcher 类：**

Matcher 对象是对输入字符串进行解释和匹配操作的引擎。与Pattern 类一样，Matcher 也没有公共构造方法。你需要调用 Pattern 对象的 matcher 方法来获得一个 Matcher 对象

* line = “www.google.com”;
* Matcher matcher = p.matcher(line);
* matcher.group(0); //返回第一个匹配的字符串
* mathcer.replaceAll(); 返回所有匹配的字符串

### **static{}静态代码块与{}非静态代码块（构造代码块）**

 静态代码块在非静态代码块之前执行(静态代码块—非静态代码块—构造方法)。静态代码块只在第一次new执行一次(不需要创建对象什么的.只要类被加载就会执行)，之后不再执行，而非静态代码块在每new一次就执行一次。 非静态代码块可在普通方法中定义(不过作用不大)；而静态代码块不行。

**内部类:**

**成员内部类:**

**(可以通过外围类实例的new方法来构造一个实例) new**

成员内部类可以创建时会获得外围类的引用,所以可以访问外围类的包括private修饰的变量

在成员内部类中要注意两点，第一：成员内部类中不能存在任何static的变量和方法；第二：成员内部类是依附于外围类的，所以只有先创建了外围类才能够创建内部类。

**局部内部类:**

有这样一种内部类，它是嵌套在方法和作用于内的，对于这个类的使用主要是应用与解决比较复杂的问题，想创建一个类来辅助我们的解决方案，到那时又不希望这个类是公共可用的，所以就产生了局部内部类，局部内部类和成员内部类一样被编译，只是它的作用域发生了改变，它只能在该方法和属性中被使用，出了该方法和属性就会失效。

**匿名内部类:**

**<https://www.cnblogs.com/chenssy/p/3390871.html>**

**静态内部类:**

     1、 它的创建是不需要依赖于外围类的。

      2、 它不能使用任何外围类的非static成员变量和方法。自己有static方法或者成员变量

枚举ENUM:

private enum TYPE {

FIREWALL("firewall", 11), //这个就是枚举的实例, 可以看做是class,new方法创建对象, 构造函数可以自己实现

SECRET("secretMac", 14),

BALANCE("f5", 12);

private String typeName;

private int num;

TYPE(String typeName, int a) {

this.typeName = typeName;

num = a;

}

/\*\*

\* 根据类型的名称，返回类型的枚举实例。

\*

\* @param typeName 类型名称

\*/

public static TYPE fromTypeName(String typeName) {

for (TYPE type : TYPE.values()) {

if (type.getTypeName().equals(typeName)) {

return type;

}

}

return null;

}

public String getTypeName() {

return this.typeName;

}

}