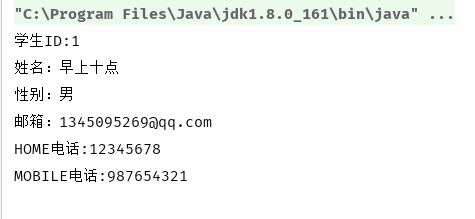
###### 周报

**工作时间**  
  
    2018.04.16 - 2018.04.20  
  
**本周工作内容**  
  
    1.主要搭建并完成了较大部分的基于spring boot框架的项目。项目我已经放到了github上了。认识到了dao层，到service层，再到controller层的开发是怎么进行的。目前的话应该就是具体的可能不是很熟，但是主要的流程什么的算是明白了。可以实现对数据库的增删查改。  
  
**项目地址：https://github.com/ZHEROLGIT/spring-boot.git**  
  
    2.写了一下关于对protobuf的理解。认识到这是一个比json更加快捷，轻量的结构化数据存储格式，可以做数据存储和数据交换格式。  
  
    3.对应着项目的内容，看了下项目里面自己还模棱两可的东西，比如为什么要去用super，以前的状态是会用。现在是知道为什么要用。还有泛型数组列表等其他基础的部分。

###### protobuf的总结

1. protobuf的原理  
   它是一种轻便高效的结构化数据存储格式，可以用于结构化数据串行化，或者说序列化。跟json类似，可以做数据存储或者数据交换格式
2. .protobuf的使用方法  
   源码上下载对应的protoc-3.5.1-win32，以及protobuf-java-2.5.0-sources.jar包，之后解压缩protoc-3.5.1-win32，找到protoc.exe编辑器。  
   具体的例子  
       syntax = "proto2";  //采用proto2版本的语言  
       package tutorial;  
       option java\_package = "com.test.sell.Protobuf";  
       option java\_outer\_classname = "ProtobufDemo";  
       message Person {  
        required string name = 2; //必要字段  
         required int32 id = 1;  
         optional string email = 3; //非必要字段  
         enum PhoneType {  
       MOBILE = 0;  
       HOME = 1;  
       WORK = 2;  
         }  
         enum Sex{  
       MAN = 0;  
       WOMAN = 1;  
         }  
         optional Sex sex = 4 [default = MAN];  
         message PhoneNumber {  
       required string number = 1;  
       optional PhoneType type = 2 [default = HOME];  
         }  
         repeated PhoneNumber phone = 5;  
       }
3. 解释：  
   option java\_package，导出的Java文件的包名  
   option java\_outer\_classname，proto编译后生成的java类的名字  
   message Person，对应到生成的java文件就是里面的一个类。所以上述文件会产生一个类Person，以及Person类的内部类PhoneNumber。

name = 2，id = 1，其中id，name对应到java类就是Person的属性名，而标签数字1和2.则表示不同的字段在序列化后的二进制数据中的布局位置，此值不能重复。

1. 运行该文件  
   直接打开命令行，进入到.proto文件的所在位置，执行  
       protoc -I=proto文件所在目录 --java\_out=生成的java文件的存放地址 proto文件所在目录/xxx.proto
2. protobuf的总结  
    类似跟json差不多的一种数据交换格式，自带了一个编辑器。给我的感觉就是，你可以直接按照他的语法写好消息格式，然后通过命令行直接生成了java类文件，（也支持c++，python）。然后这些类直接放到项目中，我们可以再创个类来完成消息的序列化和反序列化的工作。  
   测试上面的类  
       public class TestProtobuf {  
       public static void main(String[] args){  
               //序列化  
               ProtobufDemo.Person.Builder builder = ProtobufDemo.Person.newBuilder();  
               builder.setName("早上十点");  
               builder.setId(1);  
               builder.setEmail("1345095269@qq.com");  
               builder.setSex(ProtobufDemo.Person.Sex.MAN);  
     
               ProtobufDemo.Person.PhoneNumber.Builder builder1 = ProtobufDemo.Person.PhoneNumber.newBuilder();  
               builder1.setNumber("12345678");  
               builder1.setType(ProtobufDemo.Person.PhoneType.HOME);  
               ProtobufDemo.Person.PhoneNumber phoneNumber = builder1.build();  
               builder.addPhone(phoneNumber);  
     
               ProtobufDemo.Person.PhoneNumber phoneNumber1 = ProtobufDemo.Person.PhoneNumber.newBuilder()  
                       .setNumber("987654321")  
                       .setType(ProtobufDemo.Person.PhoneType.MOBILE).build();  
               builder.addPhone(phoneNumber1);  
               ProtobufDemo.Person person = builder.build();  
               byte[] personByte = person.toByteArray();  
     
               //反序列化  
               try{  
                   ProtobufDemo.Person person1 = ProtobufDemo.Person.parseFrom(personByte);  
                   System.out.println("学生ID:"+person1.getId());  
                   System.out.println("姓名："+person1.getName());  
                   System.out.println("性别："+(person1.getSex().getNumber()==0?"男":"女"));  
                   System.out.println("邮箱："+person1.getEmail());  
     
                   List<ProtobufDemo.Person.PhoneNumber> phList = person1.getPhoneList();  
                   for (ProtobufDemo.Person.PhoneNumber p : phList) {  
                       System.out.println(p.getType().toString()+"电话:"+p.getNumber());  
                   }  
               } catch (InvalidProtocolBufferException e) {  
                   e.printStackTrace();  
               }  
           }  
       }  
     
   我理解的大致就是上面这个过程。最后测试出来的结果如图：

###### java基础总结

1.构造器

特点：构造器与类同名。  
每个类可以有一个以上的构造器。  
构造器可以有0个，1个或多个参数。  
构造器没有返回值。  
构造器总是伴随着new操作一起调用。  
当类没有提供任何构造器的时候，系统才会提供一个默认的构造器。  
    class boy{  
        private String sex;  
        private int age;  
        private String name;  
        //构造器  
        public boy(String name,String sex,int age){  
            name = "张三"  
            sex = "男";  
            age = 18;  
        }  
        //Getter AND Setter 方法  
        ...  
    }  
2.super关键字

调用超类（父类）的方法。  
    调用超类的构造器。  
思路过程：  
    public double getSalary(){  
        return salary + bonus;  
        //不起作用,因为子类的方法不能直接地访问父类的私有域。  
    }  
    public double getSalary(){  
        double baseSalary = getSalary();  
        //不起作用，因为外面也有getSalary方法，这样写会导致此语句无限调用自己，直到程序崩溃为止。  
    }  
    public double getSalary(){  
        double baseSalary = super.getSalary();  
        //可以起作用  
    }  
3.抽象类(absract)  
        public abstract class Person{  
        public abstract String getDescription();  
        private String name;  
        public Person(String name){  
            this.name = name;  
        }  
        public String getName(){  
            return name;  
        }  
        }  
特点：  
除了抽象方法之外，抽象类还可以包含具体数据和具体方法。抽象方法充当着占位的角色，他们的具体实现方法在子类中。  
抽象类不能被实例化，也就是说将一个类声明为abstract，就不能创建该类的对象。但是它可以调用非抽象子类的对象。  
        new Person("Z L");//error，错误  
        Person p = new Student("Z L","L Z");//可以实现  
4.用泛型数组列表(ArrayList)代替数组  
不必指出数组的大小  
使用add将任意多的元素添加到数组中  
使用size()替代length计算元素的数目  
使用a.get(i)替代a[i]访问元素