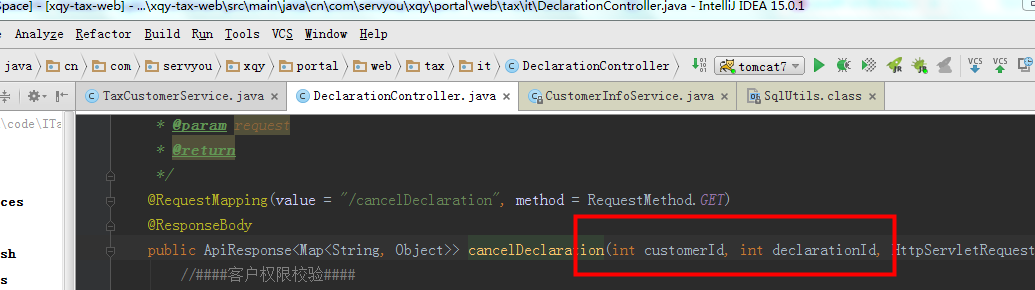
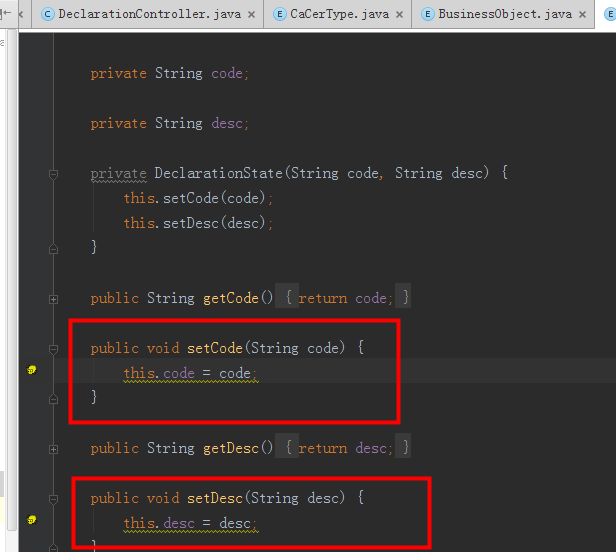
## 1、review结果汇总

### 1.1、基本类型int与其包装类型Integer



int的默认值为0，而Integer的默认值为null，即Integer可以区分出未赋值和值为0的区别，int则无法表达出未赋值的情况.

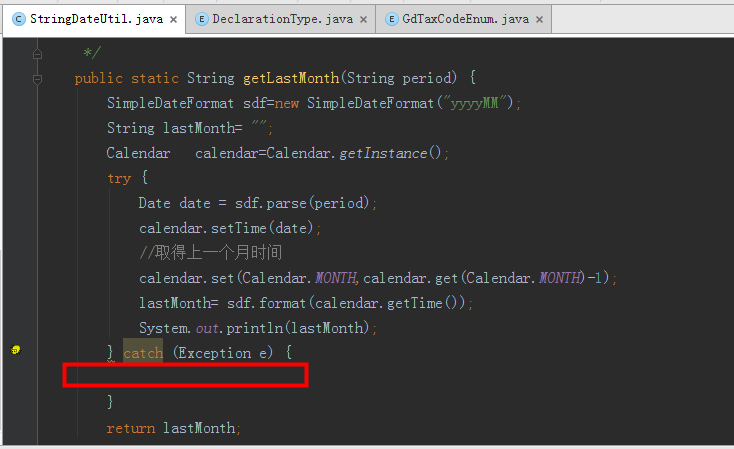
### 1.2、枚举类型Enum 暴露 public 权限的set Field



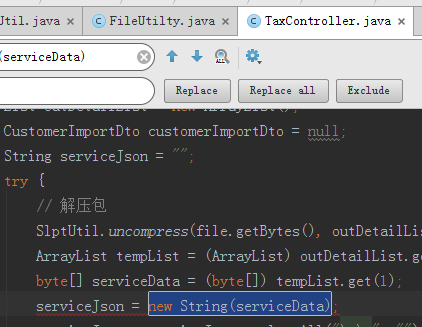
枚举类型Enum 暴露 public 权限的set Field 一方面无意义，另一方面也不安全。set Field方法改成default或者private权限

### 1.2、不要忽略异常

Catch语句中不要忽略异常，将异常信息打印到日志文件中方便排查。如果你觉得必须要忽略(加上忽略的原因)。代码风格要统一，该加空格就要加空格。

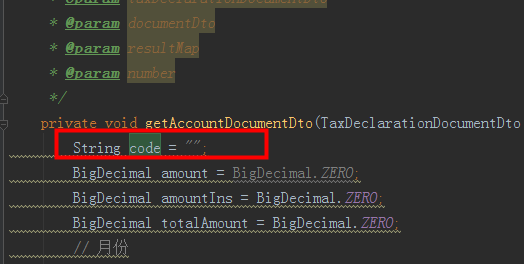


### 1.3、字节数组转换成String需要指定编码



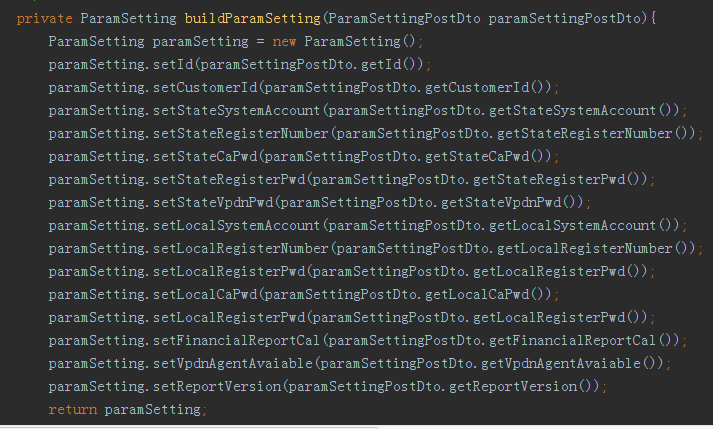
### 1.4、变量作用域尽量小

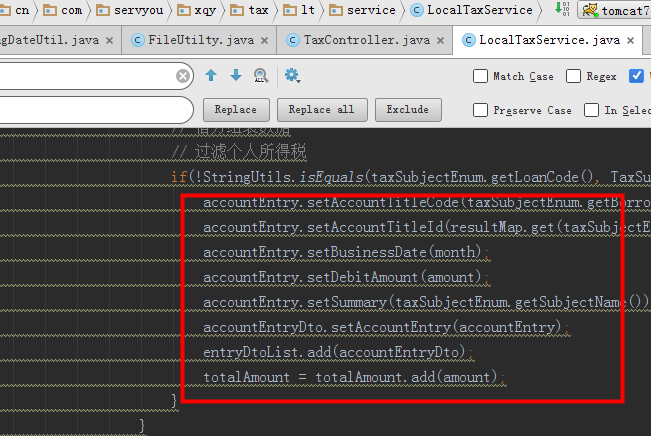
变量Code在方法体很后面才会使用到，为增强可读性和可维护性，最好是在第一次使用它的地方进行申明，减少出错的概率



### 1.5、JavaBean参数过多可以考虑使用Builder模式

JavaBean参数过多可以考虑使用Builder模式，参考Effective Java。





### 1.6、没有逻辑的代码都去掉

private String swithFeedbackState(String feedbackState) {  
 if("0".equals(feedbackState)){  
 FeedbackState.*UNDECLARED*.getCode();//1  
 }  
 if("1".equals(feedbackState)){  
 FeedbackState.*DECLARING*.getCode();//1  
 }  
 if("2".equals(feedbackState)){  
 FeedbackState.*UNDECLARED*.getCode();//0  
 }  
 if("3".equals(feedbackState)){  
 FeedbackState.*DECLARED\_SUCCESSFUL*.getCode();//2  
 }  
 if("4".equals(feedbackState)){  
 FeedbackState.*DECLARED\_FAILED*.getCode();//3  
 }  
 if("5".equals(feedbackState)){  
 FeedbackState.*DECLARING*.getCode();//1  
 }  
 if("6".equals(feedbackState)){  
 FeedbackState.*UNDECLARED*.getCode();//0  
 }  
 if("7".equals(feedbackState)){  
 FeedbackState.*UNDECLARED*.getCode();//0  
 }  
 return null;  
}

### 1.7、内部类与内部静态类问题

如果你不需要内部类对象与其外部类对象之间有联系，那你可以将内部类声明为static。如果省略static，则每个内部类实例都包含一个指向外部对象的引用，保存这份引用要消耗时间和空间，并且会导致外部类实例在符合JVM垃圾回收时仍然不能被回收。

*/\*\*  
 \* key以HDXX-NSRSBH  
 \* 存入到缓存里面的核定信息对象  
 \** ***@author*** *kangzs  
 \*  
 \*/*public class HDXXCacheDto extends DTO{  
  
 private static final long *serialVersionUID* = 20160504132730L;  
   
 */\*\*  
 \* 增值税类型：  
 \* 1 -> 一般纳税人  
 \* 2 -> 小规模纳税人  
 \*/* private String addedTaxType;  
 */\*\*  
 \* 税种列表  
 \*/* private List<SBZLDto> sbzlDto;  
   
 public List<SBZLDto> getSbzlDto() {  
 return sbzlDto;  
 }  
 public void setSbzlDto(List<SBZLDto> sbzlDto) {  
 this.sbzlDto = sbzlDto;  
 }  
 public String getAddedTaxType() {  
 return addedTaxType;  
 }  
 public void setAddedTaxType(String addedTaxType) {  
 this.addedTaxType = addedTaxType;  
 }  
 public void addSBZLDto(String code, String name) {  
 if (null==sbzlDto) {  
 sbzlDto = new ArrayList<SBZLDto>();  
 }  
 SBZLDto zl = new SBZLDto(code, name);  
 sbzlDto.add(zl);  
 }  
   
 */\*\*  
 \* key以HDXX-NSRSBH  
 \* 存入到缓存里面的核定信息对象  
 \** ***@author*** *kangzs  
 \*  
 \*/* public class SBZLDto extends DTO{  
  
 private static final long *serialVersionUID* = 20160504140311L;  
 */\*\*  
 \* 税种编码，对应到xml节点：SBZLCODE  
 \*/* private String code;  
 */\*\*  
 \* 税种名称，对应到xml节点：SBZLMC  
 \*/* private String name;  
   
 public SBZLDto(String code, String name) {  
 super();  
 this.code = code;  
 this.name = name;  
 }  
 public String getCode() {  
 return code;  
 }  
 public void setCode(String code) {  
 this.code = code;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 }  
}

### 1.8、Map遍历推荐使用EntrySet

Map遍历推荐使用EntrySet，性能一般会好一点

for (Entry<String, String> entry: map.entrySet()) {

    key = entry.getKey();

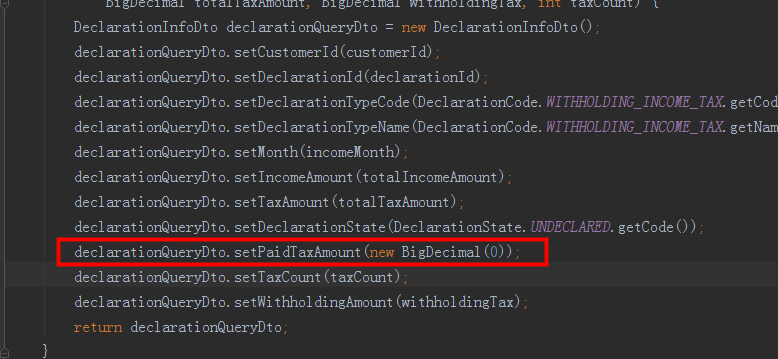
}

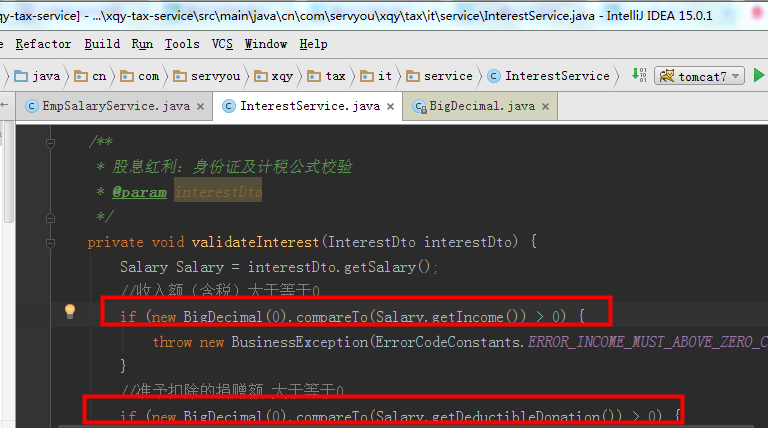
public class DecodeResult {  
  
 private Map<String, String> map = new HashMap<String, String>();  
  
  
 */\*\*  
 \* 获取申报主文件  
 \*   
 \** ***@param*** *filename  
 \** ***@return*** *\*/* public String getBody() {  
  
 for (String key : map.keySet()) {  
 if (!key.equals("SB\_HEAD.xml")) {  
 return map.get(key);  
 }  
 }  
  
 return map.get("SB\_HEAD.xml");  
 }  
  
}

### 1.9、BigDecimal已存在的几个默认值

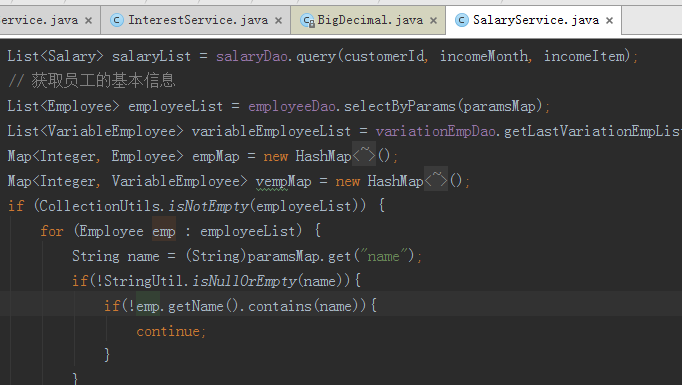
BigDecimal.*ZERO*

BigDecimal已经包含ZERO值，没必要自己new BigDecimal(0)





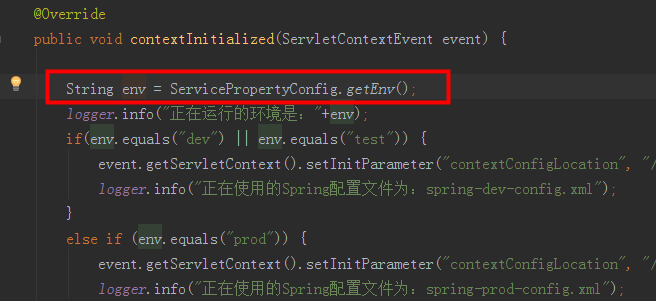
### 1.10、几个if条件可根据实际情况进行合并



减少条件嵌套，代码会更优雅一些。

### 1.11、局部变量只赋值一次可以用final修饰

局部变量只赋值一次可以用final修饰，final String env=…..。这样代码看起来会更优雅。



### 1.12、字符串拼接性能问题

多字符串拼接，改用StringBuilder builder = new StringBuilder();

builder.append().append()….

builder.toString()

String与StringBuilder、StringBuffer区别

String: final关键字修饰(类不能被继承，并且它的成员方法都默认为final方法)

对String对象的任何改变都不影响到原对象，相关的任何修改操作都会生成新的对象。

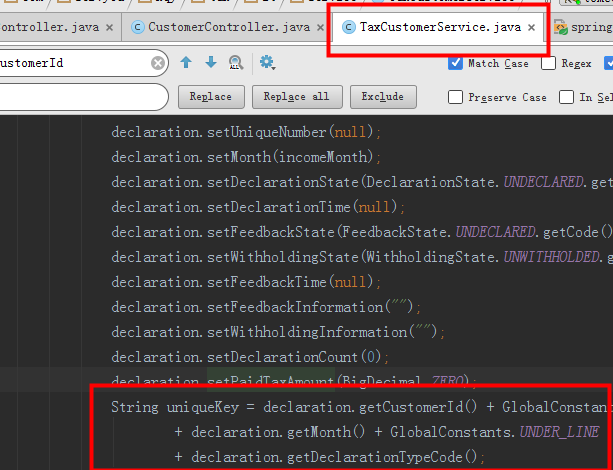
StringBuilder : append操作是在原有对象的基础上进行的。可变对象，不是线程安全

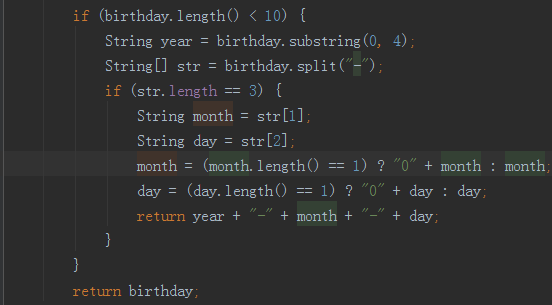
StringBuffer： 类的成员方法前面多了一个关键字：synchronized，不用多说，这个关键字是在多线程访问时起到安全保护作用的,也就是说StringBuffer是线程安全的

**StringBuilder和StringBuffer都是可变字符串，前者线程不安全，后者线程安全。**

**StringBuilder和StringBuffer的大部分方法均调用父类AbstractStringBuilder的实现。其扩容机制首先是把容量变为原来容量的2倍加2。最大容量是Integer.MAX\_VALUE，也就是0x7fffffff。**

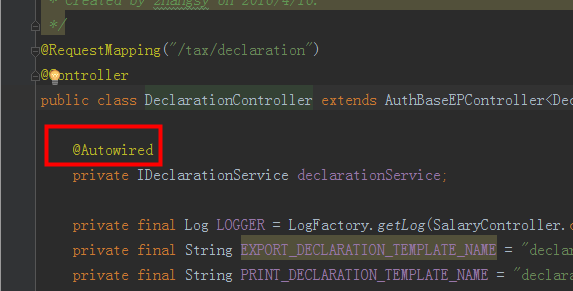
**StringBuilder和StringBuffer的默认容量都是16，最好预先估计好字符串的大小避免扩容带来的时间消耗。**





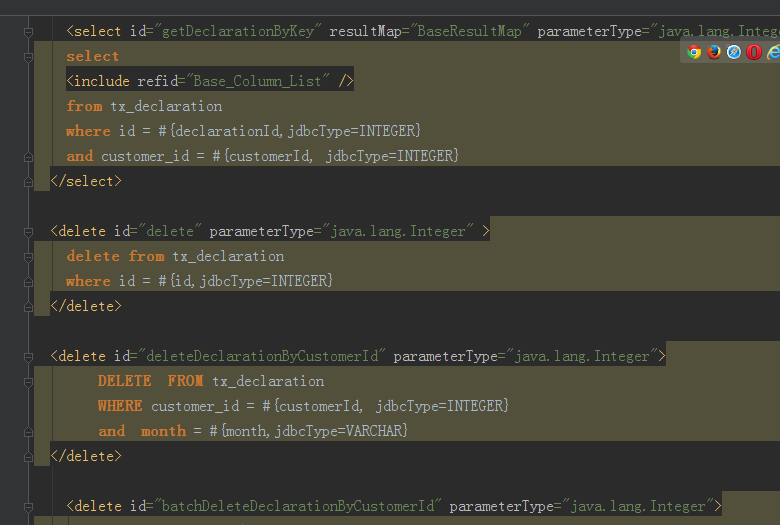
### 1.13、@Resource与@Autowired 统一

@Autowired注解属于Spring，@Resource注解属于J2ee。建议使用@Resource，减少与Spring的耦合，代码看起来会比较优雅。

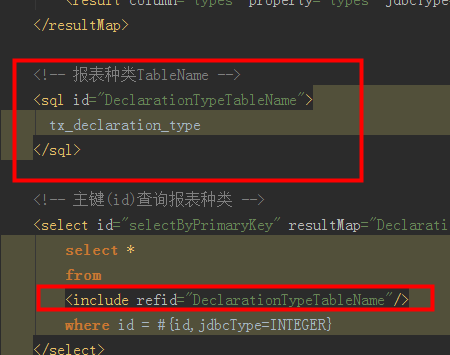


### 1.14、Mybatis Mapper文件语句注释说明

大部分Mapper.xml文件没有对sql文件进行注释说明，在编写某个SQL语句时最好增加一些简洁注释，方便其他开发人员审阅。

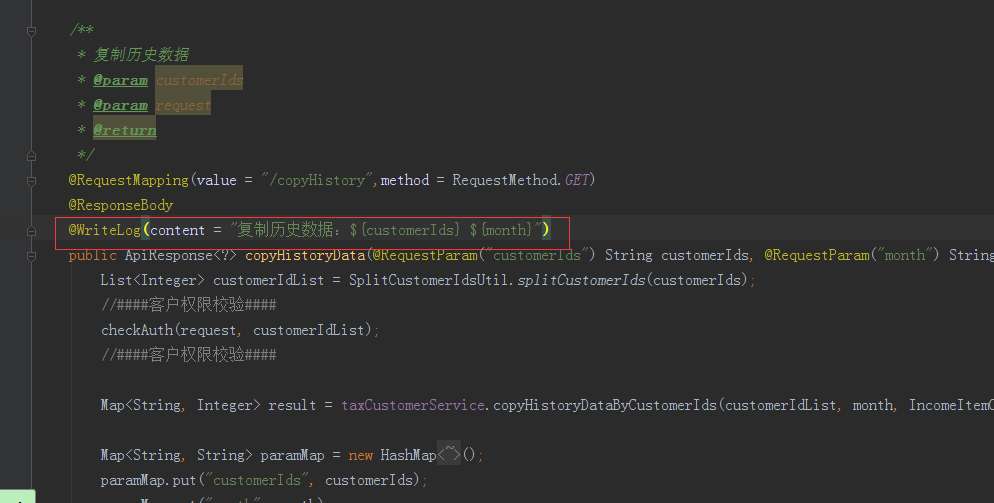
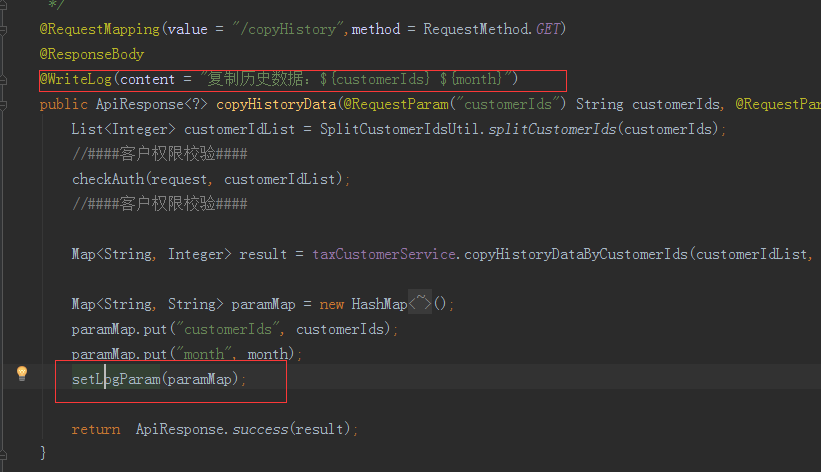


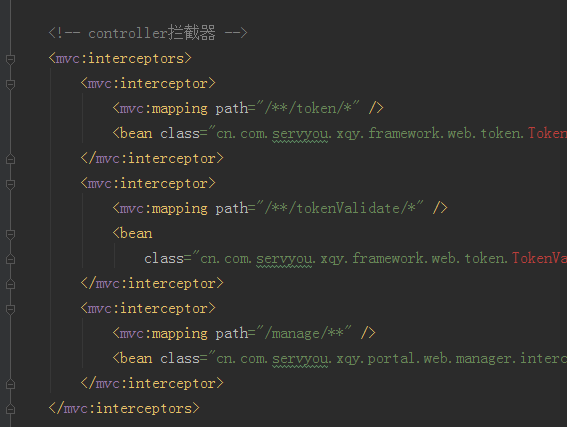
数据库表的名称可以单独抽取出来，然后利用<include refid=””>进行引用，其他很多公共的语句也可这样处理。



### 1.15、SpringMVC请求日志可以考虑使用SpringMVC拦截器

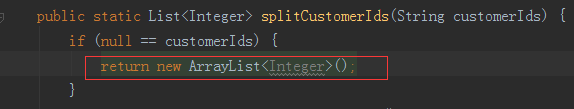
利用SpringMVC 拦截器HandlerIntercptorAdaptor可以根据配置设置哪些请求需要拦截，拦截过程中可以获取请求参数、记录当前操作人等信息，减小与业务代码的耦合度。





### 1.16、充分了解并使用Java类库

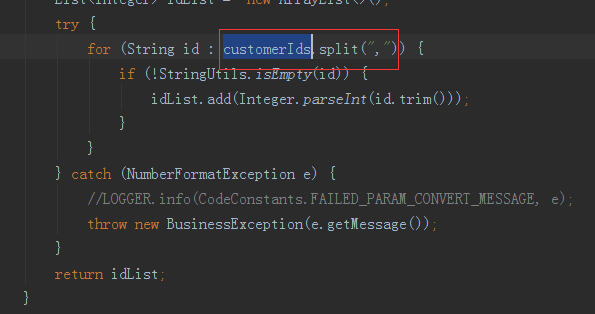
充分了解并使用Java类库



Return空的List语句可以改成**return** Collections.*emptyList*();

EmptyList是空的集合，不能进行add等操作。

### 1.17、循环终止判断条件



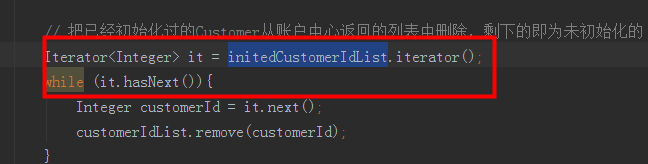
与循环无关的计算最好放到外层，性能会有所提高，可改为:

String ids = customerIds.split(“,”);

for (String id : ids) {

}

### 1.18、for-each循环优于Iterator



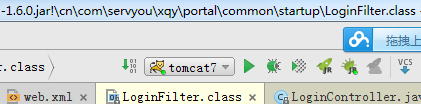
For each循环通过完全隐藏跌迭代器或者索引变量，避免了混乱和出错的可能，这种模式同样适用于集合和数组。

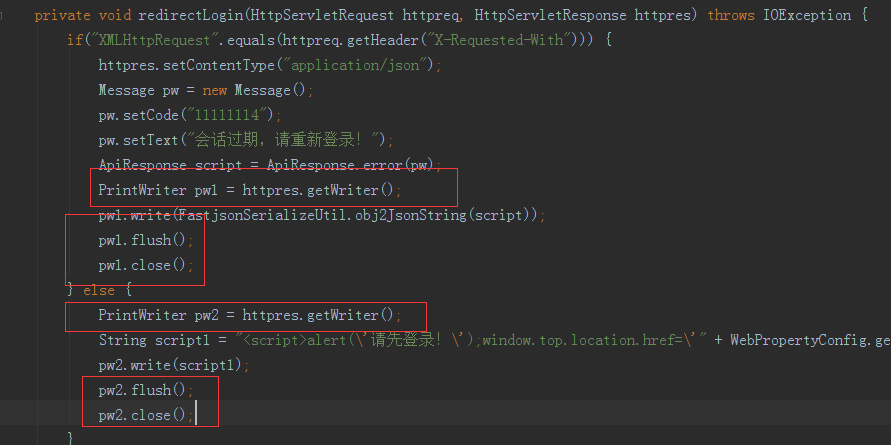
建议改成如下for-each形式，简洁明了不容易出错：

|  |
| --- |
| **for** (Integer id : initedCustomerIdList)  customerIdList.remove(id); |

如果你需要对原来的列表或者数组进行修改或者取代，就需要使用显示的迭代器。

### 1.19、IO流关闭问题





## 2、单元测试

一般开发人员拒绝单元测试原因是项目比较紧急，我没有时间。实际上这是一个错误的想法，当你在写一个包含多个复杂逻辑的service。这时候你就应该考虑编写Junit或者EasyMock来编写单元测试代码，当你回过头再看原来的代码或者其它reviewer加入项目组，总是能从单元测试中获取一些有用的逻辑信息。