

37-实战二（下）：重构ID生成器项目中各函数的异常处理代码

平时进行软件设计开发的时候，我们除了要保证正常情况下的逻辑运行正确之外，还需要编写大量额外的代码，来处理有可能出现的异常情况，以保证代码在任何情况下，都在我们的掌控之内，不会出现非预期的运行结果。程序的bug往往都出现在一些边界条件和异常情况下，所以说，异常处理得好坏直接影响了代码的健壮性。全面、合理地处理各种异常能有效减少代码bug，也是保证代码质量的一个重要手段。

在上一节课中，我们讲解了几种异常情况的处理方式，比如返回错误码、NULL值、空对象、异常对象。针对最常用的异常对象，我们还重点讲解了两种异常类型的应用场景，以及针对函数抛出的异常的三种处理方式：直接吞掉、原封不动地抛出和包裹成新的异常抛出。

除此之外，在上一节课的开头，我们还针对ID生成器的代码，提出了4个有关异常处理的问题。今天，我们就用一节课的时间，结合上一节课讲到的理论知识，来逐一解答一下这几个问题。

话不多说，让我们正式开始今天的内容吧！

重构generate()函数

首先，我们来看，对于generate()函数，如果本机名获取失败，函数返回什么？这样的返回值是否合理？

```
public String generate() {  
    String substrOfHostName = getLastFiledOfHostName();  
    long currentTimeMillis = System.currentTimeMillis();  
    String randomString = generateRandomAlphameric(8);  
    String id = String.format("%s-%d-%s",  
        substrOfHostName, currentTimeMillis, randomString);  
    return id;  
}
```

ID由三部分构成：本机名、时间戳和随机数。时间戳和随机数的生成函数不会出错，唯独主机名有可能获取失败。在目前的代码实现中，如果主机名获取失败，substrOfHostName为NULL，那generate()函数会返回类似“null-16723733647-83Ab3uK6”这样的数据。如果主机名获取失败，substrOfHostName为空字符串，那generate()函数会返回类似“-16723733647-83Ab3uK6”这样的数据。

在异常情况下，返回上面两种特殊的ID数据格式，这样的做法是否合理呢？这个其实很难讲，我们要看具体的业务是怎么设计的。不过，我更倾向于明确地将异常告知调用者。所以，这里最好是抛出受检异常，而非特殊值。

按照这个设计思路，我们对generate()函数进行重构。重构之后的代码如下所示：

```
public String generate() throws IdGenerationFailureException {  
    String substrOfHostName = getLastFiledOfHostName();  
    if (substrOfHostName == null || substrOfHostName.isEmpty()) {  
        throw new IdGenerationFailureException("host name is empty.");  
    }  
    long currentTimeMillis = System.currentTimeMillis();  
    String randomString = generateRandomAlphameric(8);  
    String id = String.format("%s-%d-%s",
```

```
        substrOfHostName, currentTimeMillis, randomString);  
    return id;  
}
```

重构getLastFiledOfHostName()函数

对于getLastFiledOfHostName()函数，是否应该将UnknownHostException异常在函数内部吞掉（try-catch并打印日志），还是应该将异常继续往上抛出？如果往上抛出的话，是直接把UnknownHostException异常原封不动地抛出，还是封装成新的异常抛出？

```
private String getLastFiledOfHostName() {  
    String substrOfHostName = null;  
    try {  
        String hostName = InetAddress.getLocalHost().getHostName();  
        substrOfHostName = getLastSubstrSplittedByDot(hostName);  
    } catch (UnknownHostException e) {  
        logger.warn("Failed to get the host name.", e);  
    }  
    return substrOfHostName;  
}
```

现在的处理方式是当主机名获取失败的时候，getLastFiledOfHostName()函数返回NULL值。我们前面讲过，是返回NULL值还是异常对象，要看获取不到数据是正常行为，还是异常行为。获取主机名失败会影响后续逻辑的处理，并不是我们期望的，所以，它是一种异常行为。这里最好是抛出异常，而非返回NULL值。

至于是直接将UnknownHostException抛出，还是重新封装成新的异常抛出，要看函数跟异常是否有业务相关性。getLastFiledOfHostName()函数用来获取主机名的最后一个字段，UnknownHostException异常表示主机名获取失败，两者算是业务相关，所以可以直接将UnknownHostException抛出，不需要重新包裹成新的异常。

按照上面的设计思路，我们对getLastFiledOfHostName()函数进行重构。重构后的代码如下所示：

```
private String getLastFiledOfHostName() throws UnknownHostException{  
    String substrOfHostName = null;  
    String hostName = InetAddress.getLocalHost().getHostName();  
    substrOfHostName = getLastSubstrSplittedByDot(hostName);  
    return substrOfHostName;  
}
```

getLastFiledOfHostName()函数修改之后，generate()函数也要做相应的修改。我们需要在generate()函数中，捕获getLastFiledOfHostName()抛出的UnknownHostException异常。当我们捕获到这个异常之后，应该怎么办呢？

按照之前的分析，ID生成失败的时候，我们需要明确地告知调用者。所以，我们不能在generate()函数中，

将UnknownHostException这个异常吞掉。那我们应该原封不动地抛出，还是封装成新的异常抛出呢？

我们选择后者。在generate()函数中，我们需要捕获UnknownHostException异常，并重新包裹成新的异常IdGenerationFailureException往上抛出。之所以这么做，有下面三个原因。

- 调用者在使用generate()函数的时候，只需要知道它生成的是随机唯一ID，并不关心ID是如何生成的。也就是说，这是依赖抽象而非实现编程。如果generate()函数直接抛出UnknownHostException异常，实际上是暴露了实现细节。
- 从代码封装的角度来讲，我们不希望将UnknownHostException这个比较底层的异常，暴露给更上层的代码，也就是调用generate()函数的代码。而且，调用者拿到这个异常的时候，并不能理解这个异常到底代表了什么，也不知道该如何处理。
- UnknownHostException异常跟generate()函数，在业务概念上没有相关性。

按照上面的设计思路，我们对generate()的函数再次进行重构。重构后的代码如下所示：

```
public String generate() throws IdGenerationFailureException {
    String substrOfHostName = null;
    try {
        substrOfHostName = getLastFiledOfHostName();
    } catch (UnknownHostException e) {
        throw new IdGenerationFailureException("host name is empty.");
    }
    long currentTimeMillis = System.currentTimeMillis();
    String randomString = generateRandomAlphameric(8);
    String id = String.format("%s-%d-%s",
        substrOfHostName, currentTimeMillis, randomString);
    return id;
}
```

重构getLastSubstrSplittedByDot()函数

对于getLastSubstrSplittedByDot(String hostName)函数，如果hostName为NULL或者空字符串，这个函数应该返回什么？

```
@VisibleForTesting
protected String getLastSubstrSplittedByDot(String hostName) {
    String[] tokens = hostName.split("\\.");
    String substrOfHostName = tokens[tokens.length - 1];
    return substrOfHostName;
}
```

理论上讲，参数传递的正确性应该有程序员来保证，我们无需做NULL值或者空字符串的判断和特殊处理。调用者本不应该把NULL值或者空字符串传递给getLastSubstrSplittedByDot()函数。如果传递了，那就是code bug，需要修复。但是，话说回来，谁也保证不了程序员就一定不会传递NULL值或者空字符串。那我们到底该不该做NULL值或空字符串的判断呢？

如果函数是private类私有的，只在类内部被调用，完全在你自己的掌控之下，自己保证在调用这个private函数的时候，不要传递NULL值或空字符串就可以了。所以，我们可以不在private函数中做NULL值或空字符串的判断。如果函数是public的，你无法掌控会被谁调用以及如何调用（有可能某个同事一时疏忽，传递进了NULL值，这种情况也是存在的），为了尽可能提高代码的健壮性，我们最好是在public函数中做NULL值或空字符串的判断。

那你可能会说，getLastSubstrSplittedByDot()是protected的，既不是private函数，也不是public函数，那要不要做NULL值或空字符串的判断呢？

之所以将它设置为protected，是为了方便写单元测试。不过，单元测试可能要测试一些corner case，比如输入是NULL值或者空字符串的情况。所以，这里我们最好也加上NULL值或空字符串的判断逻辑。虽然加上有些冗余，但多加些检验总归不会错的。

按照这个设计思路，我们对getLastSubstrSplittedByDot()函数进行重构。重构之后的代码如下所示：

```
@VisibleForTesting
protected String getLastSubstrSplittedByDot(String hostName) {
    if (hostName == null || hostName.isEmpty()) {
        throw IllegalArgumentException("..."); //运行时异常
    }
    String[] tokens = hostName.split("\\.");
    String substrOfHostName = tokens[tokens.length - 1];
    return substrOfHostName;
}
```

按照上面讲的，我们在使用这个函数的时候，自己也要保证不传递NULL值或者空字符串进去。所以，getLastFiledOfHostName()函数的代码也要作相应的修改。修改之后的代码如下所示：

```
private String getLastFiledOfHostName() throws UnknownHostException{
    String substrOfHostName = null;
    String hostName = InetAddress.getLocalHost().getHostName();
    if (hostName == null || hostName.isEmpty()) { // 此处做判断
        throw new UnknownHostException("...");
    }
    substrOfHostName = getLastSubstrSplittedByDot(hostName);
    return substrOfHostName;
}
```

重构generateRandomAlphameric()函数

对于generateRandomAlphameric(int length)函数，如果length < 0或length = 0，这个函数应该返回什么？

```
@VisibleForTesting
protected String generateRandomAlphameric(int length) {
    char[] randomChars = new char[length];
```

```

int count = 0;
Random random = new Random();
while (count < length) {
    int maxAscii = 'z';
    int randomAscii = random.nextInt(maxAscii);
    boolean isDigit= randomAscii >= '0' && randomAscii <= '9';
    boolean isUppercase= randomAscii >= 'A' && randomAscii <= 'Z';
    boolean isLowercase= randomAscii >= 'a' && randomAscii <= 'z';
    if (isDigit|| isUppercase || isLowercase) {
        randomChars[count] = (char) (randomAscii);
        ++count;
    }
}
return new String(randomChars);
}
}

```

我们先来看length < 0的情况。生成一个长度为负值的随机字符串是不符合常规逻辑的，是一种异常行为。所以，当传入的参数length < 0的时候，我们抛出IllegalArgumentException异常。

我们再来看length = 0的情况。length = 0是否是异常行为呢？这就看你自己怎么定义了。我们既可以把它定义为一种异常行为，抛出IllegalArgumentException异常，也可以把它定义为一种正常行为，让函数在入参length = 0的情况下，直接返回空字符串。不管选择哪种处理方式，最关键的一点是，要在函数注释中，明确告知length = 0的情况下，会返回什么样的数据。

重构之后的RandomIdGenerator代码

对RandomIdGenerator类中各个函数异常情况处理代码的重构，到此就结束了。为了方便查看，我把重构之后的代码，重新整理之后贴在这里了。你可以对比着看一下，跟你的重构思路是否一致。

```

public class RandomIdGenerator implements IdGenerator {
    private static final Logger logger = LoggerFactory.getLogger(RandomIdGenerator.class);

    @Override
    public String generate() throws IdGenerationFailureException {
        String substrOfHostName = null;
        try {
            substrOfHostName = getLastFiledOfHostName();
        } catch (UnknownHostException e) {
            throw new IdGenerationFailureException("...", e);
        }
        long currentTimeMillis = System.currentTimeMillis();
        String randomString = generateRandomAlphameric(8);
        String id = String.format("%s-%d-%s",
            substrOfHostName, currentTimeMillis, randomString);
        return id;
    }

    private String getLastFiledOfHostName() throws UnknownHostException{
        String substrOfHostName = null;
        String hostName = InetAddress.getLocalHost().getHostName();
        if (hostName == null || hostName.isEmpty()) {
            throw new UnknownHostException("...");
        }
        substrOfHostName = getLastSubstrSplittedByDot(hostName);
        return substrOfHostName;
    }
}

```

```

}

@VisibleForTesting
protected String getLastSubstrSplittedByDot(String hostName) {
    if (hostName == null || hostName.isEmpty()) {
        throw new IllegalArgumentException("...");
    }

    String[] tokens = hostName.split("\\.");
    String substrOfHostName = tokens[tokens.length - 1];
    return substrOfHostName;
}

@VisibleForTesting
protected String generateRandomAlphameric(int length) {
    if (length <= 0) {
        throw new IllegalArgumentException("...");
    }

    char[] randomChars = new char[length];
    int count = 0;
    Random random = new Random();
    while (count < length) {
        int maxAscii = 'z';
        int randomAscii = random.nextInt(maxAscii);
        boolean isDigit= randomAscii >= '0' && randomAscii <= '9';
        boolean isUppercase= randomAscii >= 'A' && randomAscii <= 'Z';
        boolean isLowercase= randomAscii >= 'a' && randomAscii <= 'z';
        if (isDigit|| isUppercase || isLowercase) {
            randomChars[count] = (char) (randomAscii);
            ++count;
        }
    }
    return new String(randomChars);
}
}

```

重点回顾

好了，今天的内容到此就讲完了。我们一块来总结回顾一下，你需要重点掌握的内容。

今天的内容比较偏实战，是对上节课学到的理论知识的一个应用。从今天的实战中，你学到了哪些更高层的软件设计和开发思想呢？我这里抛砖引玉，总结了下面3点。

- 再简单的代码，看上去再完美的代码，只要我们下功夫去推敲，总有可以优化的空间，就看你愿不愿把事情做到极致。
- 如果你内功不够深厚，理论知识不够扎实，那你就很难参透开源项目的代码到底优秀在哪里。就像如果我们没有之前的理论学习，没有今天我给你一点一点重构、讲解、分析，只是给你最后重构好的 RandomIdGenerator的代码，你真的能学到它的设计精髓吗？
- 对比[第34节课](#)最初小王的IdGenerator代码和最终的RandomIdGenerator代码，它们一个是“能用”，一个是“好用”，天壤之别。作为一名程序员，起码对代码要有追求啊，不然跟咸鱼有啥区别！

课堂讨论

我们花了4节课的时间，对一个非常简单的、不到40行的ID生成器代码，做了多次迭代重构。除了刚刚我

在“重点回顾”中讲到的那几点之外，从这个迭代重构的过程中，你还学到哪些更有价值的东西？

欢迎在留言区写下你的思考和想法，和同学一起交流和分享。如果有收获，也欢迎你把这篇文章分享给你的朋友。

精选留言：

● Jxin 2020-01-27 15:38:41

还学到什么：

1.一下子想搞个例子讲这些真的太难了，拍着脑子想demo。栏主这个demo背景简单，也将要讲的内容串起来了，实属不易，辛苦栏主了。

个人见解：

1.按我的习惯，我会尽量把入参和中间不可靠变量的异常校验都放在public方法，所有私有方法都以契约的方式不再做参数校验。也就是说 public方法干 1.参数校验 2. 系统一级流程编排 3.统一异常处理 这三件事。所以对private方法的提炼会和栏主有点出入。

2.如果这个id生成器还要带有业务key，比如分表路由key之类的东西。那么这个实现就还得大动干戈。但凡这种涉及持久数据的玩意，很可能需要考虑新老版本兼容的问题，也就是如何平滑过度老数据。所以需要在id生成算法上引入版本或者类型的标记，把标记打在持久化的数据上，以备平滑过度老数据。[4赞]

作者回复2020-01-27 16:24:52

我觉得你懂我~

● your problem? 2020-01-27 09:45:30

打卡，也祝大家新年快乐，身体健康，另外我始终觉得generateRandomAlphameric这个函数里，随机获取这个写法很不利于性能测试，假如这个函数会被百万，甚至千万次的调用，不可控性也太强了，我觉得可以改成随机生成0-26的数字，对应去加到字母的位置，不知道老师和大家有什么想法吗 [2赞]

● Frank 2020-01-27 21:39:52

今天学习了异常代码处理思路。在处理到异常时，通常会将上层关心的异常直接包装成RuntimeException往上抛，没有根据业务域定义相关的自定义异常。通过今天的学习，了解到处理异常的基本思路：是往上抛还是吞掉，主要看调用者是够关心该异常。是否要包装成新的异常主要看调用者是否理解该异常，该异常是否业务相关。如果能理解、业务相关可以直接抛，否则重新包装。

在这4节课的持续迭代过程中，除了文章中提到的开发思想，自己总结了如下一些个人想法：

1. 科比说过“我现在所做的一切，都是为了追求更加完美” - 缅怀逝去的伟大的科比。我们对生活，工作都要尽量追求完美。
2. 人生是个不断重构自己的过程，自己写的代码也要不断持续重构，优化。这样自己才能不断进步。
3. 参考优秀的开发思想，方法论，不断地将之实践，总结，改进，逐渐形成合适自己的方法论。[1赞]

● 高原 2020-01-27 08:58:45

希望老师每节课举的代码有下载的地方，自己下载下来结合老师讲解的，自己理解体会其中的解决问题 [1赞]

作者回复2020-01-27 14:28:02

好的，等我俩月，我整理好，一块放到github上：

<https://github.com/wangzheng0822>

● 黄林晴 2020-01-28 13:50:47

打卡

- liu_liu 2020-01-28 12:12:44
写代码不是糊弄，写出好的有水平的代码需要下一番功夫。对代码保持敬畏之心，有追求极致的思想，才会越来越好。
- javaadu 2020-01-27 21:09:01
我再提一点自己的改进想法：修改后的代码里，generate方法还应该处理掉8这个魔法数字，如果需要让用户定制长度，则需要提供另一个不带默认值的方法，并在generate方法里处理随机方法抛出的参数非法异常
- 辣么大 2020-01-27 19:30:05
学到什么：
 - 1、动手实践。争哥的这4节看着简单，实际信息量很大，动手实践一下能有更深的体会。
 - 2、跑去研究了Java Random的源码的构造函数如何实现的。
搞清random做为变量可以放在哪里使用，以及random.rndInt()的取值范围。
 - 3、异常如何处理：什么时候吞掉、抛出去，或者包装后抛出去。
 - 4、学了理论还是要多在实际中尝试应用。
 - 5、重构时先写好测试。每次baby step的重构，refactor build commit。从《重构2》里学的。
- ちよくん 2020-01-27 18:33:50
滴滴打卡
- 小晏子 2020-01-27 16:03:12
感觉generate函数失败后报错信息是host name empty会有点奇怪，意味着用户还要去设置服务器的host name，程序和服务器设置有了依赖性了，如果用户不会设置或者是基于容器的，那还要用户花费时间去设置主机名，感觉不好。
- Harvey 2020-01-27 08:20:13
设计之所以难是因为没有标准答案，很多权衡是依赖于具体业务的。这就是DDD的思想所在，要先想清楚问题域是什么在思考解决方案。很多开发讨论问题的时候没有层次，上来就陷入技术细节，这就叫缺乏抽象。下游系统要想清楚哪些是上游系统给你提供的服务？哪些是人家的内部技术实现？比如ID生成，作为上游系统，ID生成服务提供的是有小概率重复的随机ID服务，至于随机算法，下游系统不必关心，这是上游系统的内部实现，这样上游系统才有空间更换算法而不影响下游系统。
- Jeff.Smile 2020-01-27 00:43:56
大年初三，抢占沙发！
- Geek_kobe 2020-01-27 00:35:20
果然还是看技术文章能让恐慌的心静下来