基于路径比较的变异测试方法

姚香娟[1,2,3],巩敦卫[1,3]
(1.中国矿业大学信息与电气工程学院,江苏徐州 221116; 2.中国矿业大学理学院,江苏徐州 221116;
3.武汉大学软件工程国家重点实验室,湖北武汉 430072)

摘 要: 提出基于路径比较的变异测试方法.首先,通过比较穿越路径判定变异体是否被杀死;然后,建立基于路径覆盖的变异测试数据生成模型,该模型把杀死变异体作为目标,把满足特定路径覆盖作为约束;最后,采用遗传算法求解该模型.将本文方法应用于典型被测程序,结果表明,该方法可以降低变异测试的难度,并提高测试数据的生成效率.

关键词: 变异测试; 路径比较; 测试数据; 遗传算法
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2012) 01-0103-05
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.01.017

Mutation Testing Based on Comparison of Paths

YAO Xiang-juan[1,2,3], GONG Dun-wei[1,3]
(1. *School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou, Jiangsu* 221116, *China*;
2. *College of Science, China University of Mining and Technology, Xuzhou, Jiangsu* 221116, *China*;
3. *State Key Laboratory of Software Engineering, Wuhan University, Wuhan, Hubei* 430072, *China*)

Abstract: This study proposed a mutation testing method based on comparison of paths. First, whether a mutation is killed is determined by comparing the traversed paths; then, a model of test data generation for mutation testing based on path coverage is proposed. In the mathematic model, the objective is to kill the mutation and the constraint is to traverse the chosen path. Finally, we applied the proposed method to some typical programs. The experimental results show that the proposed method can not only reduce the difficulty of mutation testing, but also improve the efficiency of generating test data.

Key words: mutation testing; path comparison; test data; genetic algorithm

1 引言

变异测试(Mutation Test)最早由 Hamlet 等人[1]提出,基本思想是:首先,通过变异算子作用源程序,产生一组变异体;然后,采用相同的测试数据分别运行源程序和变异体,若两者输出不同,则认为变异体被杀死.

变异测试具有排错能力强,方便灵活,以及自动化程度高等优点,既可以用来生成测试数据,又可以用来衡量测试数据集的检错能力.但是,已有的变异测试方法需要消耗大量的计算资源[2].

为减少变异测试的开销,Howden 提出弱变异测试[3],通过观察程序执行变异节点后变量是否变化,决定变异体是否被杀死.尽管弱变异测试在一定程度上降低了变异测试的难度,但没有解决测试数据生成问题.

实际上,程序的缺陷或错误有很多表现形式,而程序输出和变量状态异常只是其中两种.何种检测方式更容易发现缺陷和错误,值得进一步研究.另外,路径覆盖是一种常用的软件测试技术[4].如果能够把路径覆盖和变异测试相结合,对提高变异测试问题的求解效率肯定会起到积极作用.

鉴于此,本文提出基于比较穿越路径查找缺陷的变异测试方法,基本思想是:对于相同的测试数据,如果运行源程序和变异体后穿越的路径不同,则认为变异体被杀死.此外,基于路径覆盖建立了变异测试数据生成问题的含约束优化模型,优化目标是杀死变异体,约束条件是满足特定路径覆盖.最后,利用遗传算法[5]对模型进行求解.该方法不需要考虑变异算子的语法结构和涉及变量的状态变化,因此简单易行,且适用于任何变异算子和程序.将本文方法应用于典型被测程序,结果证明本文方法的有效性.

收稿日期:2011-01-07;修回日期:2011-05-05
基金项目:国家自然科学基金(No.61075061); 高等学校博士学科点专项科研基金(博士生导师类)(No.201000951100006); 江苏省自然科学基金(No.BK2010187); 武汉大学软件工程国家重点实验室开放基金(No.SKLSE2010-08-19);中国矿业大学青年科技基金(No.2008A034)

2 相关工作

变异测试最重要的用途之一是生成测试数据.Offutt 提出约束法来生成杀死变异体的测试数据[6].实验表明,该方法比较有效,但是,该方法难以处理依赖于输入变量的循环条件、数组元素下标和模块调用等情况.为克服上述方法的不足,Offutt 等人又提出动态域削减法,该方法的动态特性有助于改进对循环、数组,以及表达式等的处理,并且动态地沿着控制流图移动使得路径约束立即被求解,从而在时间和空间上都更有效[7].Zhang 等人给出动态符号执行法,基本思想是:把每一个要满足的约束都转化为一个条件语句,再生成覆盖这些条件语句真分支的测试数据[8].

除了直接生成测试数据,变异测试还用于提高已有测试数据集的质量.Smith 和 Williams 利用变异测试来实现测试数据扩充,以提高测试数据集的检错能力[9].

此外,变异测试还用于回归测试.Do 和 Gregg 利用变异测试解决测试数据优先级问题.在他们的工作中,测试数据的优先级由其杀死变异体的速度决定[10].

Jia 等人认为,实际程序中存在的错误往往要复杂的多,只对程序作一处修改并不能真正模拟真实的错误或缺陷,因此,他们提出了对程序作两处或更多修改的高阶变异测试方法,但是,相应的变异体个数会成倍地增加[11].鉴于此,Langdon 等提出一种高阶变异体选择方法,该方法把变异体的选择问题看成含有两个目标的优化问题进行求解[12].

Sugeta 等把变异测试应用于基于有限状态机模型的软件测试.该方法把程序的每个状态转移看作一个可实现的功能,再对各项功能进行变异[13].Hierons 等又将上述方法推广到依概率有限状态机和依概率-随机有限状态机模型中,对具有一定概率或包含随机时间的状态转移模式,利用统计方法区分源程序和变异体[14].

目前,变异测试在单元测试[15]、系统测试[16]、面向对象软件的测试[17]等问题中都得到了广泛应用.但是,变异测试的计算代价问题仍然没有很好解决,寻找高效的变异测试方法,一直是人们研究的热点问题之一.

3 基于路径比较的变异测试

只依靠程序的运行结果很难发现程序中存在的异常.测试数据穿越的路径可以清晰地显示程序的执行流程,因此,可以通过比较穿越的路径,判定变异体是否被杀死.

3.1 基本概念

控制流图 程序的控制流图是程序控制结构的图形表示,是一种具有如下结构的有向图 $G=(N,E,s,e)$,其中,N 的元素称为 G 的节点,对应程序的某一条语句;E 的元素 $e_{ij}=(n_i,n_j)$ 称为 G 的边,表示从语句 n_i 到语句 n_j 存在控制流.每个程序的控制流图还包含一个惟一的入口节点 s 和出口节点 e.

路径 记为 p,是一个节点序列 $n_1,n_2,\cdots,n_k$,满足从节点 n_i 到 n_{i+1} 有边存在,$i=1,2,\cdots,k-1$.

顺次相同节点序列 对于两条路径 p_1 和 p_2,设 $p_1=n_{11}n_{12}\cdots n_{1k_1}$,$p_2=n_{21}n_{22}\cdots n_{2k_2}$,如果存在 t 使得 $n_{11}=n_{21},\cdots,n_{1t}=n_{2t}$,但是 $n_{1,t+1}\neq n_{2,t+1}$,则称 $n_{11}n_{12}\cdots n_{1t}$ 为 p_1 和 p_2 的顺次相同节点序列,记为 $s(p_1,p_2)$.

分叉点 如果 $s(p_1,p_2)=p_1=p_2$,则 p_1 和 p_2 为同一条路径;否则,p_1 和 p_2 为不同路径,此时,称 $s(p_1,p_2)$ 中最后一个节点为 p_1 和 p_2 的分叉点.

3.2 基于路径比较的变异测试准则

变异测试准则:以测试数据 x 为输入分别运行源程序 Φ 和变异体 M,记穿越 Φ 和 M 的路径分别为 $p(\Phi,x)$ 和 $p(M,x)$,如果 $p(\Phi,x)\neq p(M,x)$,则称 x 依路径杀死变异体 M.

该方法和常规变异测试方法的区别在于:常规变异测试检查的是程序的输出,而本文方法检查的是测试数据所穿越的路径.为加以区别,称本文方法为依路径变异测试.对于给定的变异体,依路径变异测试可以查找到常规方法难以发现的缺陷或错误.

以图 1 所示的 C 语言程序片断为例.程序的输入为整数 i 和 j,输出为二者的最小值.如果第 3 个语句变异为"if $i>j$-1",那么,对于任何 i 和 j,执行源程序和变异体后的输出都一样,所以,只考察程序的输出检测不到该错误.但是,当 $i=j$ 时,执行源程序和变异体后穿越的路径不同,即测试数据依路径杀死了该变异体.

```
1  int i,j;
2  minval = i;
3  if i > j
4  minval = j;
5  printf("%d",minval)
```

图 1 示例程序

3.3 基于路径比较的变异测试数据生成

传统方法生成杀死变异体测试数据时,一般需要考虑变异算子的语法结构和涉及变量的状态变化,这无疑会耗费大量的计算资源.鉴于此,本文提出一种新的变异测试数据生成方法,基本思想是:首先,随机选择穿越变异语句的路径作为目标路径,把覆盖目标路径作为约束条件,杀死变异体作为目标来生成测试数据;然后,依一定方式改变目标路径,并生成相应测试数据,直到变异体被杀死或满足其他终止条件为止.

(1)选择目标路径

设穿越变异语句的路径共有 t 条,分别记为 $p_1,p_2,\cdots,p_t$.随机选择其中的一条作为目标路径.最初,每

条路径被选中的概率都是一样的,即 $1/t$.如果某路径被选中 q 次,且都没能生成杀死变异体的测试数据,那么,可以认为穿越该路径的测试数据不能杀死变异体或杀死变异体的能力很弱,因此,该路径被后续选中的概率降为 $1/qt$.

对目标路径 p_1,生成随机数 $r_1\in[0,1]$,如果 r_1 小于 p_1 的选中概率,则选择其作为目标路径;否则,考察目标路径 p_2,依相同的方式确定其是否选为目标路径.如此重复进行,直到某条路径被选中为止.

(2)建立数学模型

为了采用遗传算法生成测试数据,需要把测试数据生成问题转化为优化问题.记:

$$f(x)=\begin{cases}0 & x\text{ 杀死 }M\\ 1 & \text{否则}\end{cases}$$

容易看出,当 x 杀死变异体 M 时,$f(x)$的取值为 0;当 x 不能杀死变异体 M 时,$f(x)$的取值为 1.这样,就把杀死变异体 M 的测试数据生成问题转化为函数 $f(x)$的最小化问题,即 $\min f(x)$.

但是,$f(x)$的取值只有 0 和 1 两种.众所周知,这样的函数表示难以有效引导遗传算法搜索到最优解.通过限定 x 的取值可以在一定程度上提高找到最优解的效率,这可以通过路径覆盖实现.

不失一般性,记选择的目标路径为 p_i,那么,把 x 穿越该路径作为优化问题所要满足的约束条件.记 x 穿越的路径为 $p(x)$,定义 x 对路径 p_i 的层接近水平为 $p(x)$和 p_i 的分叉点到 p_i 的最后一个条件语句之间的条件语句数,记为 $l(x,P_i)$;另外,定义 x 对路径 p_i 的分支距离为 $p(x)$和 p_i 产生分叉时,对期望分支的偏离程度,记为 $d(x,p_i)$.不同条件语句的分支距离定义请参考文献[18].令:

$$g_i(x)=l(x,P_i)+1-1.001^{-d(x,P_i)} \tag{1}$$

那么,x 穿越路径 p_i 的充要条件为 $g_i(x)=0$,并且 $g_i(x)$越小,$p(x)$和 p_i 的接近程度就越高.这样一来,变异测试数据生成问题可以转化为如下优化问题:

$$\begin{aligned}&\min \quad f(x)\\ &\text{s.t.} \quad g_i(x)=0\\ &\qquad x\in D\end{aligned} \tag{2}$$

其中,D 是被测程序的输入域.

(3)适应值函数

采用遗传算法生成变异测试数据时,需要将决策变量(程序的输入)编码,并设计合适的适应值函数.在不引起混淆的前提下,记决策变量 x 的编码仍为 x,其适应值为 $F(x)$,现在给出 $F(x)$的表达式.由于我们的目的是生成杀死变异体的测试数据,也就是生成使 $f(x)$取最小值的 x,因此,当 $f(x)=0$ 时,我们也期望 $F(x)=0$;当 x 不能杀死变异体时,目标函数值 $f(x)=1$,此时,个体的优劣主要依靠 $g_i(x)$区分:$g_i(x)$越小,x 的适应值 $F(x)$就应该越小.基于以上考虑,适应值函数可表示为:

$$F(x)=f(x)\cdot(g_i(x)+c) \tag{3}$$

其中,c 可以是任意较小的常数,以保证括号中的值都大于 0.这样,$F(x)=0$ 当且仅当 x 杀死变异体.

(4)算法步骤

基于路径比较的变异测试数据生成方法的步骤如下:

步骤 1　依设定的概率选择路径 p_i 作为目标路径;

步骤 2　将测试数据生成问题转化为式(2)表示的优化问题,并采用遗传算法求解,个体适应值依据式(3)确定;

步骤 3　当算法满足终止准则时,如果找到问题的最优解,则输出测试数据,结束变异测试;

步骤 4　修改路径被选中的概率,转步骤 1.

算法的流程图如图 2 所示.其中,第一个虚线框是变异测试的主体部分,即判别所生成的测试数据是否能够杀死变异体;第二个虚线框是遗传算法的主体部分,即用于解决式(2)所描述的优化问题.依次循环,直到满足结束条件.在本文,变异测试的结束条件是所生成的测试数据能够杀死变异体,或者已经生成相当数量的能够覆盖目标路径的测试数据.

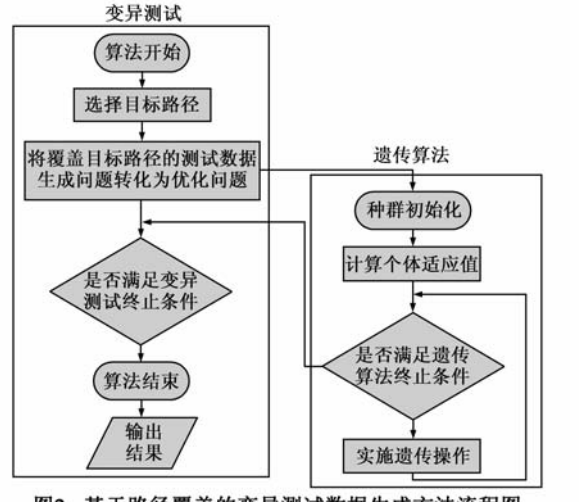

图2　基于路径覆盖的变异测试数据生成方法流程图

4　实验

实验目的有两个:一是验证本文提出的依路径变异测试的合理性;二是验证本文提出的基于路径比较的变异测试数据生成方法的有效性.

4.1　被测程序

我们选择 8 个不同类型和难度的程序,这些程序被广泛应用于验证各种软件测试方法.这 8 个被测程序的结构信息如表 1 所列.

表 1 被测程序的结构信息

程序	代码行数	If 语句数	循环语句数
Bubble Sort	24	2	2
Anomaly Detector	55	5	2
Array Difference	31	4	2
Postcode	170	25	0
Sliding Window	137	72	5
Sortcode	95	13	0
Vending Machine	114	12	4
Tcas	173	7	0

4.2 实验设计

为了达到第一个目的,分别采用传统的变异测试和依路径变异测试,生成杀死变异体的测试数据,比较生成测试数据的效率.为了达到第二个目的,比较本文提出的变异测试数据生成方法和随机法.因为传统方法需要静态分析,而静态分析消耗的时间难以衡量,因此,本文只与随机法比较.

对每个被测程序,定义若干不同类型的变异体,个数从 20 到 40 不等.使用路径法和随机法两种方法生成变异测试数据,并分别采用依路径比较和依输出比较准则,判定变异体是否被杀死.对每个被测程序,在相同条件下独立实验 30 次,记录每次实验需要评价的个体数和运行时间,并取其平均值,结果如表 2 所列.

4.3 实验结果及分析

首先,比较不同变异测试准则对生成测试数据需要评价的个体数和运行时间的影响.

路径法生成变异测试数据时,依路径比较准则生成测试数据需评价的个体数和运行时间明显低于依输出比较准则.差别最大的是 Bubble Sort 程序,依输出比较需要评价的个体数和运行时间分别为 2164325 和 124.7s,是依路径比较的 3.61 倍;差别最小的是 Array Difference 程序,依输出比较需要评价的个体数和运行时间分别为 447642 和 32.8s,是依路径比较的 1.07 倍和 1.11 倍.

表 2 实验结果

程序	依路径比较				依输出比较			
	路径法		随机法		路径法		随机法	
	评价个体数	时间(s)	评价个体数	时间(s)	评价个体数	时间(s)	评价个体数	时间(s)
Bubble Sort	598921	34.5	819734	51.2	2164325	124.7	1968723	122.9
Anomaly Detector	602153	44.2	893429	70.3	1152564	84.6	2099426	164.8
Array Difference	416673	29.6	419834	31.8	447642	32.8	1218934	92.3
Postcode	1043245	88.9	2085247	226.4	2390421	203.7	2966023	321.7
Sliding Window	1565933	102.6	1894321	194.7	3864532	253.2	5178923	532.1
Sortcode	1023543	79.3	1654321	164.8	2119322	164.2	3474321	345.7
Vending Machine	3933265	56.7	4302434	71.5	5229723	75.4	7179632	119.3
Tcas	1843246	127.8	3789522	286.8	3295622	228.5	5778312	437.2

随机法生成变异测试数据时,依路径比较准则生成变异测试数据需要评价的个体数和运行时间明显低于依输出比较准则.相差最大的是 Sliding Window 程序,依输出比较需要评价的个体数和运行时间分别为 5178923 和 532.1s,是依路径比较的 2.73 倍;相差最小的是 Postcode 程序,依输出比较需要评价的个体数和运行时间分别为 2966023 和 321.7s,是依路径比较的 1.42 倍.

这些结果充分说明,依路径比较准则比依输出比较准则更容易发现程序中的缺陷或错误,大大提高了变异测试的效率.

然后,比较不同的变异测试数据生成方法对生成测试数据需要评价的个体数和运行时间的影响.

采用依路径比较准则,路径法生成测试数据需要评价的个体数和运行时间明显低于随机法.对需要评价的个体数,差别最大的是 Tcas 程序,随机法为 1843246,是路径法的 2.06 倍;差别最小的是 Array Difference 程序,随机法为 419834,与路径法相当.对运行时间,相差最明显的是 Postcode 程序,随机法为 226.4s,是路径法的 2.55 倍;相差最小的是 Array Difference 程序,随机法为 31.8s,仅是路径法的 1.07 倍.

采用依输出比较准则,除 Bubble Sort 程序外,路径法生成测试数据需要评价的个体数和运行时间明显低于随机法.相差最明显的是 Array Difference 程序,随机法需要评价的个体数和运行时间分别为 1218934 和 92.3s,是路径法的 2.72 倍和 2.81 倍;相差最小的是 Postcode 程序,随机法需要评价的个体数和运行时间分别为 2966023 和 321.7s,是路径法的 1.24 倍和 1.58 倍.对 Bubble Sort 程序,路径法生成测试数据需要评价的个体数和运行时间分别为 2164325 和 124.7s,略高于随机法的 1968723 和 122.9s,这说明,对于特定程序,路径法生成测试数据的效率可能没有随机法高.但是,对于多数程序,路径法生成测试数据的效率远高于随机法.

这些结果充分说明,对于多数程序,本文提出的路径法生成变异测试数据的效率高于随机法.此外,本文的依路径比较准则不需要考虑变异算子的语法结构和涉及变量的状态变化,因此简单易行,适用于任何形式的变异算子和程序.

5 结束语

变异测试是一种常用的软件测试技术,既可用来生成测试数据,又可用来衡量测试数据集的揭错能力.但现有的变异测试方法通常消耗大量的计算资源.

本文给出一种基于路径比较和路径覆盖的变异测试技术.首先,给出基于路径比较的变异测试准则,即如果相同的测试数据穿越源程序和变异体的路径不同,则认为变异体被杀死;然后,给出基于路径覆盖的变异测试数据生成方法,该方法把杀死变异体作为目标,把满足特定路径覆盖作为约束建立优化模型,并采用遗传算法求解该优化问题,有效克服了已有变异测试方法的不足,提高了测试数据生成的效率.

尽管本文方法有效提高了变异测试的性能,但是,当变异体个数非常庞大时,生成杀死所有变异体测试数据的工作仍然比较繁重.因此本文下一步要研究的工作就是如何在路径变异测试中缩减变异体和测试数据的个数,进一步提高测试的效率.

参考文献

[1] R G Hamlet. Testing programs with the aid of a compiler[J]. IEEE Trans on Software Engineering, 1977, 3(4): 279 – 290.

[2] 单锦辉,高友峰,刘明浩,等.一种新的变异测试数据自动生成方法[J].计算机学报,2008,31(6):1025 – 1034.
J H Shan, Y F Gao. Novel evolutionary generation approach to test data for multiple paths coverage[J]. Chinese Journal of Computers, 2008, 31(6): 1025 – 1034. (in Chinese)

[3] W E Howden. Weak mutation testing and completeness of test sets[J]. IEEE Transactions on Software Engineering, 1982, 8(4): 371 – 379.

[4] 巩敦卫,张岩.一种新的多路径覆盖测试数据进化生成方法[J].电子学报,2010,38(6):1329 – 1304.
D W Gong, Y Zhang. Novel evolutionary generation approach to test data for multiple paths coverage[J]. Acta Electronica Sinica, 2010, 38(6): 1329 – 1304. (in Chinese)

[5] 任子武,伞冶.实数遗传算法的改进及性能研[J].电子学报,2007,35(2):269 – 274.
Z W Ren, Z San. Improvement of real-valued genetic algorithm and performance study[J]. Acta Electronica Sinica, 2007, 35(2): 269 – 274. (in Chinese)

[6] A J Offutt. Automatic test data generation[D]. Atlanta: Georgia Institute of Technology, 1988.

[7] A J Offutt, Z Jin, J Pan. The dynamic domain reduction procedure for test data generation[J]. Software, Practice and Experience, 1999, 29(2): 167 – 193.

[8] L M Zhang, T Xie, L Zhang, et al. Test generation via dynamic symbolic execution for mutation testing[A]. Proceedings of IEEE International Conference on Software Maintenance[C]. Timisoara: IEEE Press, 2010. 1 – 10.

[9] B H Smith, L Williams. On guiding the augmentation of an automated test suite via mutation analysis[J]. Empirical Software Engineering, 2009, 14(3): 341 – 369.

[10] H Do, G Rothermel. On the use of mutation faults in empirical assessments of test case prioritization techniques[J]. IEEE Trans on Software Engineering, 2006, 32(9): 733 – 752.

[11] Y Jia, M Harman. Higher order mutation testing[J]. Information and Software Technology, 2009, 51: 1379 – 1393.

[12] W B Langdon, et al. Efficient multi-objective higher order mutation testing with genetic programming[J]. The Journal of Systems and Software, 2010, 83(12): 2416 – 2430.

[13] T Sugeta, et al. Mutation testing applied to validate SDL specifications[A]. Proceedings of the 16th IFIP International Conference on Testing of Communicating Systems[C]. Birmingham: Springer Verlag, 2004. 193 – 208.

[14] R M Hierons, M G Merayo. Mutation testing from probabilistic and stochastic finite state machines[J]. The Journal of Systems and Software, 2009, 82: 1804 – 1818.

[15] G Fraser, A Zeller. Mutation – driven generation of unit tests and oracles[A]. Proceedings of the 19th international symposium on Software testing and analysis[C]. New York: ACM Press, 2010. 147 – 158.

[16] P R Mateo, M P Usaola, J Offutt. Mutation at system and functional levels[A]. Proceedings of Third International Conference on Software Testing, Verification, and Validation Workshops[C]. Paris: IEEE Press, 2010. 110 – 119.

[17] L Madeyski, N Radyk. Judy – a mutation testing tool for java [J]. IET Software, 2010, 4(1): 32 – 42.

[18] D W Gong, X J Yao. Testability transformation based on equivalence of target statements[J/OL]. Neural Computing & Applications, 2010. http://www.springerlink.com/content/cl3j1779107k0141/.

作者简介

姚香娟　女,1975 年 3 月出生于河北赵县,博士,副教授,硕士生导师,主要研究方向:基于搜索的软件工程.
E-mail: yxjcumt@126.com

巩敦卫　男,1970 年 3 月出生于江苏铜山,博士,教授,博士生导师,主要研究方向:基于搜索的软件工程、智能优化与控制.
E-mail: dwgong@vip.163.com