每次不重样，带你收获最新测试技术！

微信扫一扫关注我们

投稿邮箱：editor@51testing.com

35 岁危机，职场人的生死劫

◆ 作者：呼叫 2019

“夜深知雪重，时闻折竹声”。雪折，一种在雪的载荷下，植物（多指树）的躯干或枝条被不断堆积的雪花压断的现象。我的朋友阿聪刚刚经历了人生的第一次“雪折”。

阿聪是一个有点聪明且勤奋好学的人，从考入省重点大学起，一直是家长口中别人家的孩子、全村人希望。出身普通家庭的他更懂得本领的重要，大学期间他拼命学习，不放过任何一个机会锻炼自己。毕业后加入国内 top3 的互联网公司，从事产品设计。刚工作那会，他认为着工作是一直上升的。幻想着只要努力用不了多久，就能升职加薪，迎娶白富美，走上人生巅峰。所以，他很珍惜这份工作，也很感谢公司给的 offer。他想为公司奉献到老，从一而终不辜负公司栽培。遗憾的是这只是他的一厢情愿，公司可没有这样的想法。

工作六年，尽管历经三次业务方向调整。但是，凭借自身的勤奋和主动，绩效年年优秀，生活过的忙碌且充实。曾几何时，互联网行业刮起了一波一波的裁员潮。起初，他毫不在意，慢慢的他开始留意那些同事今天起没有上线。在不安中度过 2 年左右。去年 11 月，他还是收到了裁员通知，连同他一起被裁的还有整个部门。公司战略调整，整条业务线没了。人到中年突然失业，阿聪顿时感到了压力。每月定期的还款，日见衰老的父母，逐渐长大的孩子，需要花钱的地方是日渐增多，奈何收入却没有了。说到这，阿聪苦笑的摇摇头，端起桌上的酒杯，顿～顿～的灌了几大口，叹气道：“只有刀子落在自己身上，才知道有多疼。”但是生活还要继续，当下的痛苦只能苦涩的挨着，无奈终究是成年人世界的主基调。现在，喝酒成了他当下为数不多的解脱瞬间。

犹记得十年前，35 岁还是年富力强、志存高远、意气风发的代名词。现在却变成了人人谈虎色变的 35 岁危机。35 岁危机的迹象之一就是工作不再像当初那样“甜美”。表现在开始觉得手上的工作枯燥乏味，没有工作激情。工作获得感低，没有成就感。向上

晋升渠道受阻，存在晋升障碍，职业遭遇“天花板”。工作变得固化单调，一眼看到了几十年后的自己。

迹象之二就是赖以生存的工作变得不那么可靠了。表现在当初加入的行业，发展变得滞缓或者沦为夕阳产业，随时可能被淘汰。寄身的公司前途变得风雨飘摇，部门朝不保夕，担忧自己随时可能失业。行业内竞争加剧，996、007 内卷严重。公司不断涌入年轻人，一群精力旺盛，干劲十足职场牛犊，开始超越前辈向上收割老员工。

当出现以上两种情况之一时，35 岁危机就来临了。但是，35 岁危机并不是只在 35 岁才会爆发。对于不同人时间有先后，遭遇不可避。有些人是 32 岁，有些人则是 40 岁，不尽相同。出现 35 危机的本质原因在于，随着工龄的增加，打工人都会走到人生的三个拐点：价值拐点、生理拐点和角色拐点。不幸的是这三个拐点通常都发生在 35 岁附近。

价值拐点是指随着工作年限的增长，职级的提升，使用你的成本不断增高。但是，你的价值创造能力在经历一段抛物线式的增长后不可抗拒的会出现衰减，这削弱你的性价比。在市场化大环境下，对公司而言，就意味着从你身上可获取的剩余价值不断减少，甚至无法从你身上获取剩余价值。当剩余价值太少时，公司就开始考虑优化掉你方案。再加上你所使用的技术开始变得老旧，跟不上新业务发展需求，被需要程度降低。另外就是，岁数大了锐意进取的精神就少了，攻坚能力下滑，守着自己的一亩三分地，没有给公司开疆拓土，重要程度下滑。到这个份上，裁员就成为公司的考虑选项。毕竟公司也要生存，也会被淘汰，它不这么做，死的就是它。

生理拐点指随着年龄的增长，健康就像一节被持续放电的干电池，在日夜操劳中不断被损耗。过了“35 岁”的一个重要感觉就是病痛不再像从前一样忍一下就恢复了。开始接受不了熬夜通宵，担忧这个疼痛会持久化。身体素质从巅峰开始不断走下坡路，损伤变得不可逆，疼痛不时发生，开始关注死亡问题。这种生理上的变化，会进一步改变你之前的职业观。工作变得不再是唯一，生活、健康也并列在考虑选项中，拼搏奋斗的昂扬斗志开始消散。比如做事前不再像从前那样奋不顾身。精力不足，对自身技能学习提升投入减少。没有持续学习动力，缺乏持久的提升岗位技能和领域技能规划。

现实也确实如此，我们的身体冥冥之中遵守着不可违的自然规律。斯坦福大学医学院科学团队发表在《自然 • 医学》的论文显示：通过血液检测可以预测生理年龄，人体衰老有明显进展的 3 个转折时期。这意味着人类很可能不是通常以为的慢慢地、匀速地变老，而是在三个特定的年龄阶段，突然加速衰老，呈现动态非线性状。这三个衰老转

折点发生在 34 岁、60 岁以及 70 岁左右。其中 34 岁是第一个也是与职场最相关、最有意义的转折点。这就是为什么有人过了某个年龄后，会迅速的变胖、衰老、意志消沉，这是本就是事物发展的正常阶段。

角色拐点是指随着社会发展和技术进步，行业对某一类岗位的需求会动态变化，或多或少。这种变化基本遵循着萌芽、发展、高峰、衰退四个阶段，每个阶段所需要的的人力资源不同。不幸的时，我们漫长的生命不可避免的会经历这些部分或者全周期。眼看他起高楼，眼看他宴宾客，眼看他楼塌了，时刻发生在我们身边。尤其是你初入的行业，正处于前两个周期时，在经历行业从蓬勃发展进入持续萎缩，直至沦为夕阳产业。其中，巨大的落差感受就更明显。比如，我们的土木老哥，从三项五总变成拎桶跑路，也不过短短 10 年的时间。

尤其是，高考选专业时，80、90 后的父母们的经验在那个剧烈变化的时代已经严重落伍。许多人糊里糊涂的选了专业，选了大学。毕业找工作也缺乏深思熟虑，长远考虑。大部分人进入一个发展前景不好的行业，虽然个人很努力，奈何抵不住行业的大衰退。一顿操作下来，被当日的同学、朋友原因摔在身后。这就是吃亏在前期规划缺乏系统性分析，缺少前瞻性规划，导致选择的职业行业门槛低，很容易被替代。在初入职场时，没有选择良好企业文化的公司，被当做人力资源消耗。所从事的业务类型简单，缺乏技术含量，可替代门槛低。

再加上大部分人随着工作时间变长，学习能力下降，接受、适应新事物能力变弱。两周掌握一个技能，一月学门语言变得越来越难，自身岗位技能、领域技能不断退化，丧失竞争力。比如常被提到的程序员、财务、设计师这些技术门槛不高的行业。一个初中级程序员，公司找个新人集中培训几个月，试炼两个项目就能做到替换。这样看，即是作为高级程序员的你，被替代也就是一两年的时间成本。

35 岁是道坎，有危也有机。近期大火的电视剧《狂飙》，就呈现一众大器晚成的老戏骨。尤其是演员张颂文（饰演黑社会大哥高启强），演技炸裂，角色细腻，给观众留下难以忘怀的印象。明明演的是个坏人，却也恨不起来。张颂文本人就是一个大器晚成的人，前半生普普通通，平平无奇。生活也不是一帆风顺，工作更是居无定所。为了生计，装过空调、糊过日历、端过盘子、干过导游。最穷困时，一年才能工作 4 个月，拿到微薄的几千元工资，窘迫到在顺义租个农家院，种菜过活。一干就是 14 年，多少个青春都耗干了，搁一般人身上，早就认命了，干嘛和自己过不去。但是，他硬是生生扛过低谷

期。因为没有向生活妥协，因为长达十几年的蛰伏磨炼，为他今天的爆火打下扎实的底蕴。所以，面对危机，不服输才有希望。努力不一定就能成功，但是努力决定了你人生下限不会太低。于 47 岁杀青的张颂文，74 岁出道的姜太公相比，我们都还年轻，还有时间和机会。

大器晚成适合需要底蕴积累的行业，比如演艺、医疗、科研这类行业，纯熟的技能和深厚的经验在业内是稀缺品，是硬通货。攒齐这两点，即是不能出人头地，也是衣食无忧。遗憾的是并不是所有行业都是如此，有些行业吃的就是青春饭。年轻人就像矿产资源，扔进炉子里一顿锤炼，然后吐出一堆矿渣。年轻人青春没了，精力没了，再利用再重塑的可能性就微小了。比如工厂、客服、销售、码农等，不需要太复杂的领域技能或岗位技能。员工只需要像一颗螺丝一样，日复一日重复简单的动作。这种行业需要的是那些成本低、有活力的年轻人，但是也是对年轻人伤害最大的行业。进去的是年轻人，出来的基本就废了。

所以，选择行业和环境很重要，尤其是要选那些逆工业化的行业。如果有条件，选择技术门槛高的行业。这些行业的岗位被替代的可能性才比较低，培养一个岗位高手的时间更长，你与别人拉开的距离就越大。然后，就是保持学习，不断提升岗位技能和领域技能。很多人觉得这是老生常谈毫无新意，然而这才是更本质的东西。创造价值越多，你就更被需要。市场化是没有感情的，优胜劣汰。公司追求的是更多的剩余价值，这样才能在自由市场的腥风血雨中存活下来。所以，你的性价比越高，剩余价值越多，那不论什么时候走到那里，你都是香饽饽。

但是，需要强调的是不要把平台赋予的能量当本领。很多人错把平台影响力看成自己的能力，认为自己就是能力非凡，坐拥一堆资源，怎么走都没问题。一旦脱离平台，现实很快就教会他重新做人。这时才发现自己啥都不是，平时绕着自己的资源也不好使了，干啥啥不成，做啥啥失败。承受能力差的人，经这么一锤子暴击，大概率就不正常了。我的朋友就遇到这样一个被告人，一即将奔五的中年男子。职前是某为区域副总，在北京两套房，攥着一千万的股票，觉得自己可以更进一步。然后出来搞创业，业务还没稳定，先有新欢。原配闹离婚，大气放弃两套房。疫情来了，业务也不顺利，结果背了不少债务。现在整个人都精神不太稳定，拖着病躯到法院应诉。这就是典型的认知失衡，能力匹配不上欲望。

通过危机，成功转型的都是少数人的事迹，我们大部分人都不具备化危为机的能力，

盲目的折腾只会酿成更大的危机。但是，这并不妨碍我们预防危机、规避危机，尤其是那些还不足 33 岁的职场人和还未步入职场的年轻人。早一点认识自己当前的处境，认识自己的行业发展前景，早一点准备，多一点规划总胜过毫无准备。

一个好工作的特点就是工作中能学到知识，获得成长。有发挥自己的特长空间，可以源源不断获得快乐。能让自己学会思考，生活过的充实且安稳。相反，坏工作就是把你当做干电池，没有锻炼，没有培养，工作就像机器一样重复着简单的事情。我们可以根据这些衡量下自己当前的工作是否是值得投入、值得长期耕耘的，如果不是，尽早更换职业赛道。人生就像马拉松，一时的快代表不了最终的结果。趁着年轻，多埋下些种子，趁着还有时间，多储备些技能，趁着还有机会，尽早跳槽。说不定那一刻，某一点就发挥了作用。执迷于一时的得失，沉浸在当下的舒适，终究要面对危机的集中爆发。

拓展学习

[1] 【Python 自动化测试学习交流群】学习交流

咨询：微信 atstudy-js 备注：学习群

[2] ISTQB 认证基础级培训（含考试）

http://testing51.mikecrm.com/BPY6sJs

ADAS 执行器性能测试范例

◆作者：狼图腾

概述：
笔执行器性能分为横向性能和纵向性能，横向性能主要指方向盘转向的响应性能，纵向主要包括油门加速性能及刹车减速性能。其中横向性能在 ADAS 中涉及的功能包括 LKA、LDW，跟纵向加/减速性能相关的功能主要是 ACC（自适应巡航），纵向减速相关的主要是 AEB（自动紧急制动）功能。

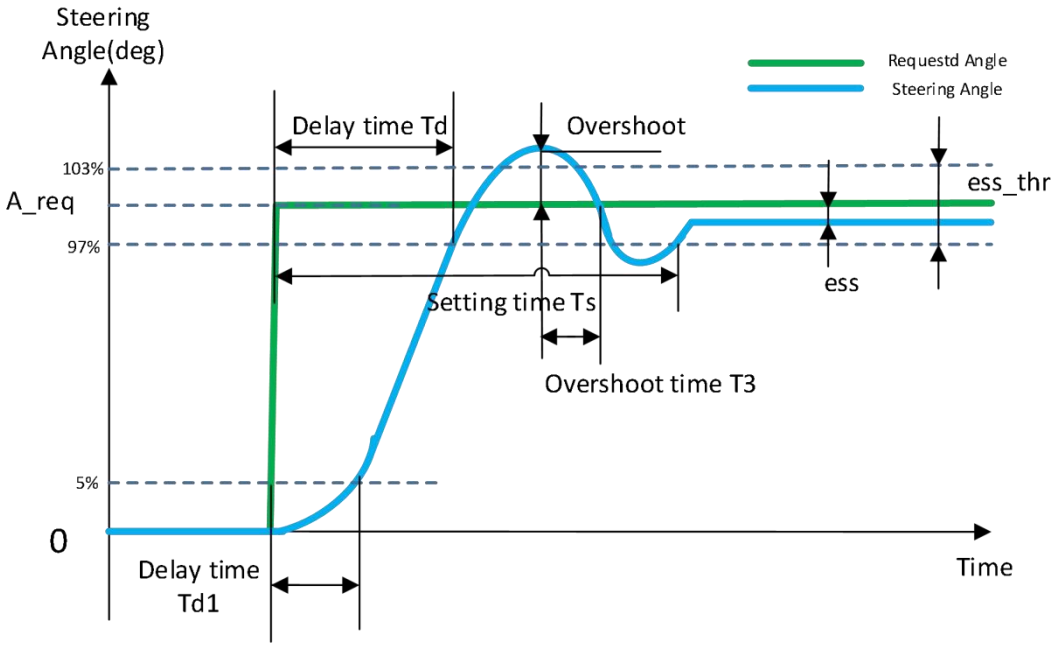
需求描述（以横向为例）

性能要求

性能要求针对阶跃输入、正弦输入、斜坡输入三种工况。

①阶跃响应

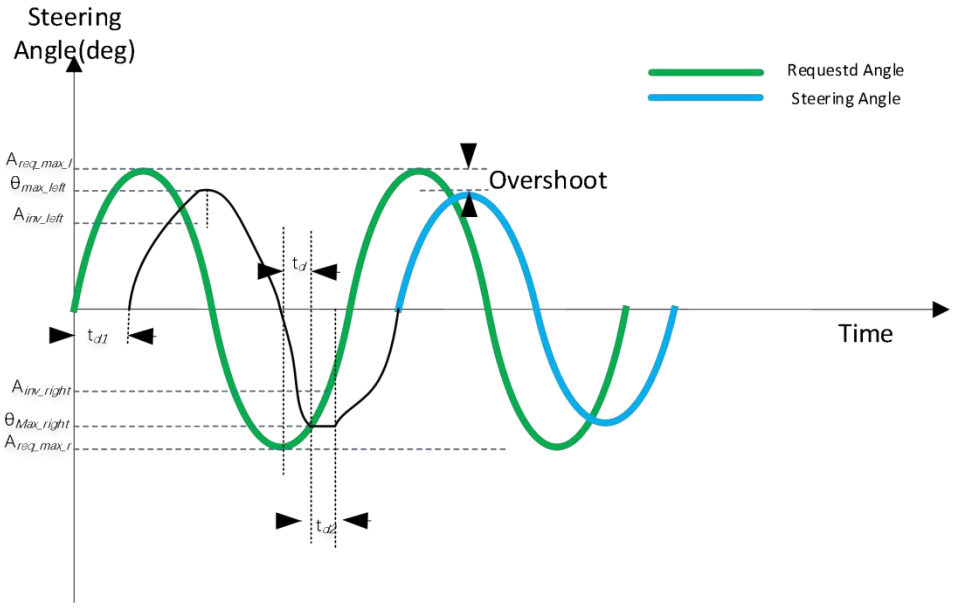
角度请求输入示意图如下：



图中绿色线条表示发送的请求角度值，蓝色线条代表实际相应的角度变化，关键性能要求参数说明及指标如下表所示：

参数名称	解释说明&性能指标
A_req	转向角度请求值。ADAS 发出的角度请求稳态值
Ess(Steady state error)	稳态误差，实际转角平稳后与角度请求值得偏差 计算公式：（响应结果稳态值-请求值）÷请求值 性能指标：Ess≤3%
Delay time——Td1	初始响应延迟时间 请求发送的时刻到响应值达到请求值 5%所用的时间 性能指标：Td1<50ms
Delay time——Td	请求值响应延迟时间，从角度请求达到稳定值的时刻，到方向盘实际转角达到目标值 97%的时间 性能指标：Td≤100ms
Setting time——Ts	稳定时间，从角度请求达到目标值的时刻到实际响应达到稳态误差极限之间的时间 性能指标：Ts≤200ms
Overshoot	超调量，实际响应角度峰值与稳态角度请求的差值 性能指标：Overshoot≤A_req*10%
T3	超调时间 性能指标：T3≤50ms

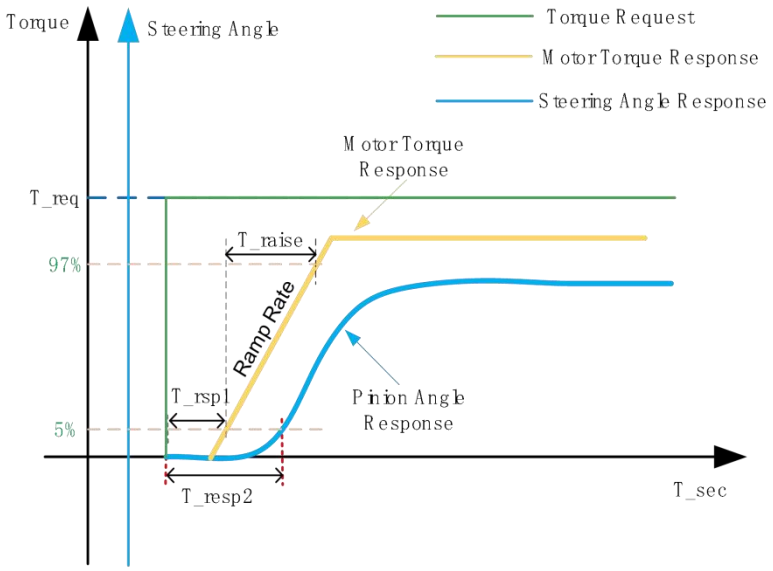
②正弦角度响应



性能要求参数说明及指标如下表所示：

参数名称	解释说明&性能指标
Areq_max_l	向左请求角度峰值
Areq_max_r	向右请求角度峰值
θ max_left	左转角峰值，角度正弦输入下向左转角的响应峰值
θ max_right	右转角峰值，角度正弦输入下向右转角的响应峰值
Ainv_left	左回转角度，方向盘转角达到左转角峰值后开始回转的对应的角度请求值
Ainv_right	右回转角度，方向盘转角达到由转角峰值后开始回转的对应的角度请求值
Td1	转角响应延迟时间 请求发送的时刻到响应值达到请求值 5%所用的时间 性能指标：Td1<50ms
Overshoot	超调量，实际响应角度峰值与稳态角度请求的差值 性能指标：Overshoot≤幅值*10%
Td	实际转角峰值响应时间差值，请求角度峰值到响应角度峰值时间差 性能指标：Td≤100ms

③斜坡响应



性能要求参数说明及指标如下表所示：

参数名称	解释说明&性能指标
T_req	转向扭矩请求值。ADAS 发出的扭矩请求稳态值
Delay Time T_rsp1	从扭矩输出到电机端扭矩响应达到请求值的 5%的延迟时间 性能指标：T_rsp1<50ms
Delay Time T_rsp2	从扭矩输出到转向角响应到请求值的 5%的延迟时间 性能指标：T_rsp2<120ms
T_rasie	电机端扭矩从响应到请求值的 5%到最大值的 97%的时间 性能指标：T_raise<200ms

测试用例

工况	用例 ID	车速 /kph	目标角度 /°	测试步骤
	1	20	-2	1.确认测试环境安全，连接测试设备。 2.车辆保持要求车速行驶，设定目标角度，开始记录数据。 3.发送转向请求。 4.横向控制开关发送指令 3s 后，停止测试，存储数据。
	2		-5	
	3		-10	
	4		-15	
	5		-50	
	6		-80	
	7		-100	
	8		2	
	9		5	
	10		10	
	11		15	
	12		50	
	13		80	
	14		100	
	15	30	-2	
	16		-5	
	17		-10	
	18		-15	
	19		-50	
	20		-80	
	21		-100	
	22		2	
	23		5	
	24		10	
	25		15	
	26		50	
	27		80	

阶跃输入	28		100	
	29	40	-2	
	30		-5	
	31		-10	
	32		-15	
	33		-50	
	34		-80	
	35		-100	
	36		2	
	37		5	
	38		10	
	39		15	
	40		50	
	41		80	
	42		100	
	43	50	-2	
	44		-5	
	45		-10	
	46		-15	
	47		-50	
	48		-80	
	49		2	
	50		5	
	51		10	
	52		15	
	53		50	
	54		80	
	55	60	-2	
	56		-5	
	57		-10	
	58		-15	
	59		-50	
	60		2	
	61		5	
	62		10	
	63		15	
	64		50	
	65	80	-2	
	66		-5	
	67		-10	
	68		-20	
	69		2	
	70		5	

	71		10	
	72		20	
	73		-2	
	74		-5	
	75		-10	
	76		-15	
	77	100	2	
	78		5	
	79		10	
	80		15	

工况	用例 ID	车速 /kph	幅值 /°	频率 /Hz	测试步骤
正弦输入	1	100	5	0.2	1.确认测试环境安全，连接测试设备。 2.车辆保持要求车速行驶，设定目标角度，开始记录数据。 3.发送转向请求。 4.横向控制开关发送指令 3s 后，停止测试，存储数据。
	2	90	5	0.2	
	3	80	5	0.2	
	4	70	20	0.1	
	5	60	30	0.1	
	6	50	40	0.1	
	7	40	40	0.1	
	8	30	80	0.1	
	9	20	80	0.1	
	10	10	80	0.1	

工况	用例 ID	车速 /kph	叠加扭矩/Nm	测试步骤
斜坡输入	1	50	-3	1.确认测试环境安全，连接测试设备。 2.车辆保持要求车速行驶，设定控制模式为 EMA，设定目标叠加扭矩，开始记录数据。 3.发送转向请求。 4.横向控制开关发送指令 3s 后，停止测试，存储数据。
	2	50	3	
	3	50	-1	
	4	50	1	
	5	80	-2	
	6	80	2	
	7	80	-1	
	8	80	1	
	9	100	-1	
	10	100	1	
	11	100	-2	
	12	100	2	
	13	60	-3 至 3 斜率为 5Nm/s	
	14	60	-3 至 3 斜率为 5Nm/s	

工况	用例 ID	车速 /kph	目标加减速度 /ms-2	测试步骤
纵向加速	1	怠速	0.5	1.确认测试环境安全，连接测试设备。 2.车辆保持要求车速行驶，设定目标加/减速度值，开始记录数据。 3.发送加/减速度请求。 4.车辆停止后，暂停测试，存储数据。
	2	怠速	1	
	3	怠速	2.2	
	4	30	0.5	
	5	30	1	
	6	30	2.2	
	7	60	0.5	
	8	60	1	
	9	60	2.2	
	10	80	0.5	
	11	80	1	
	12	80	2.2	
纵向减速	13	100	-0.5	
	14	100	-1	
	15	100	-2	
	16	100	-3	
	17	100	-4	
	18	30	-5	
	19	30	-6	

工况	用例 ID	车速 /kph	目标减速度 /ms-2	测试步骤
AEB 减速	1	10	-5	1.确认测试环境安全，连接测试设备。 2.车辆保持要求车速行驶，设定目标减速度值，开始记录数据。 3.发送减速请求。 4.车辆停止后，暂停测试，存储数据。
	2	10	-10	
	3	20	-5	
	4	20	-10	
	5	20	-12	
	6	60	-5	
	7	60	-10	
	8	60	-12	
	9	100	-5	
	10	100	-10	
	11	100	-12	

总结

执行控制是自动驾驶真正落地的基础，感知定位如同驾驶员的眼睛，规划决策相当于驾驶员的大脑，而执行器就好比驾驶员的手脚，并且规划决策无法和执行控制剥离，对执行器缺乏了解，决策就无从做起。

如文章开头所说，执行器包括横向控制（即转向控制）和纵向控制（即速度控制），当前研究的比较多也是横向控制，通俗地讲，横向控制为给定一个车速，通过控制转向达到车辆沿着预定轨迹行驶的目的，而纵向控制的目的是为了满足车辆行驶过程中的速度要求，有时还需要配合横向控制达到满足车辆在轨迹跟踪的同时，还需要满足安全性、稳定性和舒适性的目的。因为车辆是一个特别复杂的系统，横向、纵向都有耦合关系的存在，因此就需要对自动驾驶车辆进行横纵向及其协同控制测试。

拓展学习

[1] 自动驾驶测试用例数量太大怎么办？

https://www.atstudy.com/course/1012021

[2] 整车测试/座舱域测试/ADAS 测试系统学习

https://mp.weixin.qq.com/s/XWYX7-g8l6vzykNLAFw6vA

DevSecOps 安全测试探索实践与思考

◆ 作者：曹怡佳

一、背景

DevSecOps 是伴随 DevOps 标准体系的建设而延展出的一种全新的安全理念与模式。在 DevSecOps 标准要求下，安全是整个团队每个人的责任，需要贯穿业务从开发到运维全生命周期的每一个环节。它强调的是安全测试左移，从需求设计初期就尽早的将安全考虑在内，从源头处提升安全能力，这样能更有效、更低成本地解决目前面临的安全问题，全面提升应用服务的整体安全能力，助推数字化转型。

二、DevSecOps 二级标准要求

DevSecOps 标准属于 DevOps 标准体系中安全及风险管理能力域的要求内容，涵盖了通用、开发、交付、运营 4 个能力子域，细分了 17 个能力项。二级贯标实施主要针对通用、开发、交付 3 个能力子域共 9 个能力项，其中涉及测试的标准要求主要包括开发、交付 2 个能力子域中的需求管理和测试管理 2 个能力项。具体要求如下：

能力域	能力子域	能力项	2级 （基础级：完善规范化，工具自动化）	3级 （全面级：深度规范化、工具系统化、全面自动化、度量驱动改进）
安全及风险管理	控制开发过程风险	需求管理	1）需求包含安全内容，并纳入团队整体的需求清单 2）分析项目涉及的法律法规和行业规范要求，并制定合规和安全需求基线，如：个人隐私风险等 3）针对安全需求具有相应的用例，并明确验收标准，如：安全需求清单等 4）针对不同技术栈，制定相应的安全需求	1）同上，且需达到以下要求： 2）具有持续更新的安全需求标准库和管理平台，包括但不限于：法律法规、行业监管要求、公司的安全策略以及业界最佳实践等 3）针对应用场景特点，制定相应的安全需求与用例，如：Web 应用安全、移动应用安全等 4）安全需求与其他功能性需求同步开展测试，由测试团队和安全团队联合负责 5）安全需求既要包括功能性需求，如：认证、授权、安全日志与审计等，又要包括非功能性需求，如：健壮性、可用性、可靠性等
	控制交付过程风险	测试管理	1）在交付过程中，有明确的安全测试的要求，安全测试结果作为发布的前置条件 2）采用主流的安全工具进行安全测试和合规扫描，如：黑盒安全测试工具、静态代码安全扫描工具等 3）开发测试环境不直接使用生产数据，采用公开数据、构造出的测试数据或经过脱敏后的生产数据 4）基于安全需求，制定相应的安全测试用例，并进行验证测试 5）安全测试用例和非安全性测试用例进行统一管理	1）具有完善的安全测试流程和规范，安全测试结果作为发布的前置条件 2）安全测试结果具有明确的质量门限，如：高危漏洞数量不能大于0等 3）具备完善的端到端安全测试工具链进行安全测试和合规扫描，覆盖主要的安全测试类型，包括但不限于：黑盒安全测试工具、静态代码安全扫描工具、开源组件安全扫描工具、容器安全扫描工具等安全测试工具 4）在流水线中集成自动化安全测试，安全测试结果自动化反馈研发处理 5）引入人工渗透测试，如：针对业务逻辑、越权等漏洞进行人工测试 6）具备自动化安全测试结果汇总展示能力 7）持续优化安全测试策略，具备机制持续降低误报率与漏报率

DevSecOps 标准中涉及测试的能力项

三、安全测试探索与实践

在贯标测试实施过程中，结合现有的项目过程管理要求和项目原有的测试方法，按照贯标要求进行了补充和完善，总结出以下几个工作要点和方法：

1.安全左移，测试左移

在项目需求研制阶段，测试人员参与需求研制和评审，按照前期积累总结的领域级安全控制要点，按清单逐项检查安全需求中的各个条目，从需求源头保证其完整正确。同时，提前开展测试分析，在评审确定安全需求的过程中，同步讨论分析测试场景，形成测试方案，通过测试左移确保测试分析准确高效。

类别	安全要求	安全需求示例	需求纳入情况	裁剪说明	需求类型	研制责任主体	备注
身份鉴别	设置用户最大连续认证失败次数限制，并提供超出限制采取的措施。	用户连续登录（认证）失败次数限制为X次，超过X次后锁定用户，XX小时后自动解锁或者由管理员进行解锁。	纳入需求		功能性安全需求	产品经理主导 项目经理配合	
身份鉴别	账号被锁定时，应记录错误日志，错误日志至少包括时间、用户、IP地址等。宜立即通过短信或电话等可靠的方式通知用户或客户。	账号被锁定时，记录错误日志，错误日志至少包括时间、用户、IP地址等。	不纳入需求	现有系统已实现	功能性安全需求	产品经理主导 项目经理配合	
身份鉴别	在一些需要进行人机检测、频率限制的位置，必须添加图片验证码验证。图片验证码可以采取字符应答、特定字符图案点击、图片模块拖拽等形式	1、在验证密码、短信验证码等口令信息时，当用户连续验证失败超过X次，则后续验证操作附加图片验证码。 2、在发送短信验证码、短信通知之前，先验证图片验证码。 3、在用户注册、找回密码时，对于用户名是否存在的判	不纳入需求	现有系统已实现	功能性安全需求	产品经理主导 项目经理配合	

安全需求检查示例

序号	安全类别		安全规则	安全要点	参考规范	项目实施关注环节				适用业务处理环节			重要性	适用范围
	大类	子类				需求研制	技术审查	系统开发	系统测试	客户准入	授信定价	开户签约		通用/领域/产品
													高	领域级：
													高	领域级：
													高	领域级：
													高	领域级：
													低	领域级：
													高	领域级：
													高	领域级：
													中	领域级：
													高	领域级：

测试场景及方案示例

2.条目化，打标签，可追溯

按照上述形成的需求清单、测试场景、测试方案，结合系统应用开发的设计，梳理系统功能和接口安全规则的具体实现方式，分析业务流程中关联的各个系统间、系统内交互流转，整理各个输入输出字段项和安全控制要求，形成条目化的安全测试案例设计

思路和安全测试规则，指导具体测试案例设计。

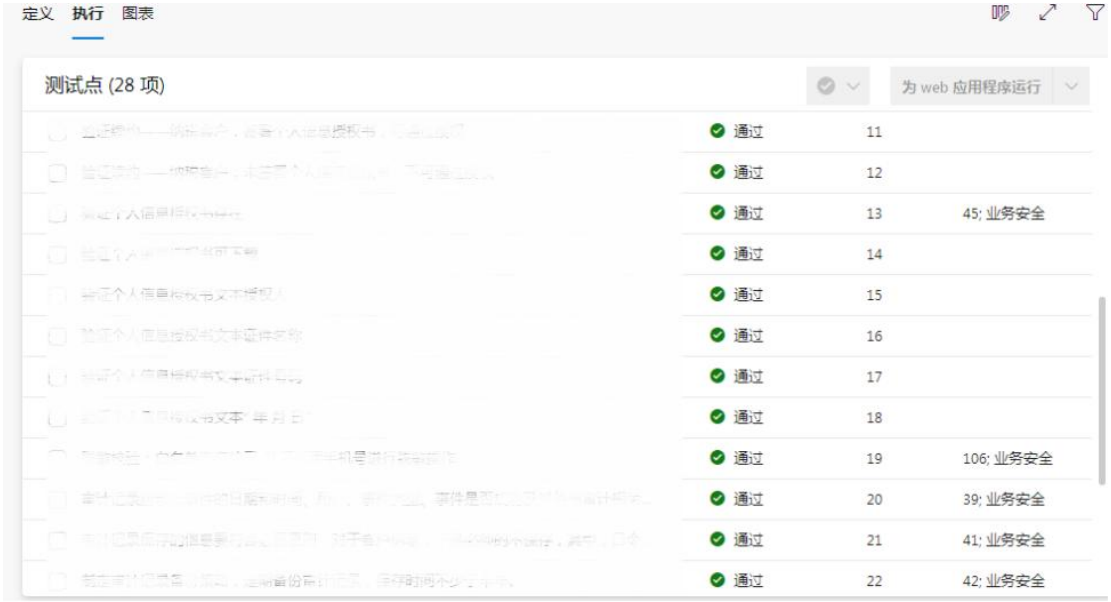
安全类别		测试说明	测试方式/工具	案例概要描述（基于安全类型的角度独立分析的结果，可在编制案例时去除重复部分）
大类	子类			
原则性审查	必要性原则	1、依据接口设计文档，分析判断接口输入、输出要素各字段是否按“最小必要原则”进行设计； 2、检查接口输入、输出报文是否与设计相一致；	查看文档/代码	梳理每个字段的用途，检查接口输入输出报文与设计文档一致
			查看文档/代码	梳理每个字段的用途，检查接口输入输出报文与设计文档一致
			查看文档/代码	梳理每个字段的用途，检查接口输入输出报文与设计文档一致
访问控制	水平越权	1、对输入信息中客户信息进行一致性验证； 冒用客户身份信息 冒用客户关键功能节点信息 冒用客户账户信息 2、是否登录用户与接口返回客户为同一个。	Postman	1、对客户信息（客户id、账号）一致性进行验证 2、渠道端验证返回的客户ID是否与登录用户为同一个
			Postman	无
			Postman	1、对客户信息（客户id、账号）一致性进行验证 2、渠道端验证返回的客户ID是否与登录用户为同一个

条目化的安全测试案例设计思路示例

序号	系统/模块名称	子模块名称	二级子模块名称【可选】	输入输出（I/O）	输入输出取值	系统交易名称	测试规则描述	案例
1								ZYEDXM-001
2								ZYEDXM-003
3								ZYEDXM-008
4								ZYEDXM-009
5								ZYEDXM-012
6								ZYEDXM-016
7								ZYEDXM-018
8								ZYEDXM-025
9								ZYEDXM-026、ZYEDXM-027
10								ZYEDXM-038
11								ZYEDXM-041
12								ZYEDXM-042
13								ZYEDXM-043
14								ZYEDXM-045
15								ZYEDXM-044
16								ZYEDXM-049
17								ZYEDXM-050~ZYEDXM-056
18								ZYEDXM-061
19								ZYEDXM-062
20								ZYEDXM-063
21								ZYEDXM-064
22								ZYEDXM-065
23								ZYEDXM-067
24								ZYEDXM-068
25								ZYEDXM-069
26								ZYEDXM-070
27								ZYEDXM-071
28								ZYEDXM-078
29								ZYEDXM-073~ZYEDXM-077
30								ZYEDXM-079
31								ZYEDXM-0811
32								ZYEDXM-0812
33								ZYEDXM-081、ZYEDXM-082
34								ZYEDXM-085
35								ZYEDXM-086

条目化的安全测试规则示例

将设计好的安全案例统一纳入测试管理工具，与需求中的功能项建立映射关系，对安全测试案例进行单独打标标注，以便于检索和区分，保证需求中每条安全需求均有完整正确的案例覆盖。



安全测试案例 TFS 打标签示例

四、后续工作的思考

总结 DevSecOps 贯标实践、DevOps 评级和以往项目测试经验，在后续的测试中，考虑加强以下几方面建设，保障应用系统安全，提高测试工作质量。

1.做好测试左移。专项针对安全测试，梳理和完善领域级、产品级的安全需求，积累安全需求的测试规则资产。

2.完善安全需求的测试分析和场景设计。按照 DevOps 测试分层理念，分类分析安全规则在系统接口、业务功能流程等不同层次的实现方式，从要素合法性、功能校验逻辑、底线控制规则等不同切面和维度，对应采取适合的测试方法开展测试验证。

3.探索安全测试与自动化测试的结合。参考现有功能测试执行中对于自动化测试工具的使用，建设安全需求和规则对应的自动化测试案例集，将安全案例的执行接入持续集成流水线，提升工作效率，降低验证成本，以便更快、更全面的识别应用安全缺陷。

几个例子带你了解 nonlocal 和 global 的用法区别

◆作者：王小健

题记：
有个朋友问我 nonlocal 的用法，如此，就拿它与 global 做个对比，更能体现出它的用法。
如果你理解了这几个案例，那么你就懂了其中的区别了。

1.global 用法解析

一句话总结：global 可以用于任何地方，声明变量为全局变量（声明时，不能同时赋值）；声明后再修改，则修改了全局变量的值。

用个例子来举证：

```
num = 0
def f():
  def inner():
  global num
  num = 2
  num = 1
  print('num1:: %s' % num)
  inner()
  print('num2:: %s' % num)

if __name__ == "__main__":
  f()
  print('num0:: %s' %num)
```

代码解析：

1. 首行定义全局变量 num=0；

2. 方法 f 中定义局部变量 num=1，且没有 global 声明，因此与外部的 num 没有任何关系，因此 num1 直接输出为 1；

3. 调用 inner 函数，函数中声明 inner 中的变量 num 绑定为全局变量，即与首行的 num 为同一变量，且变更为 2

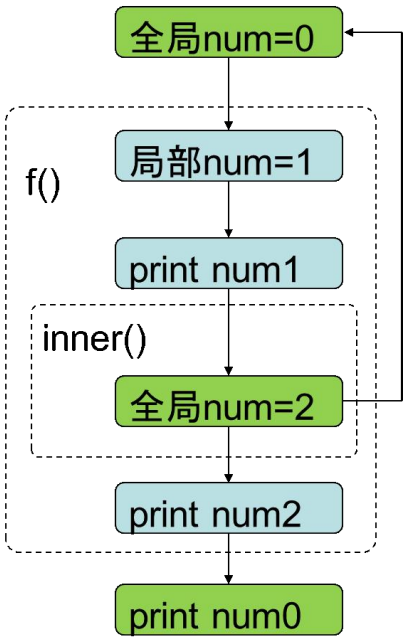
4. 由于 global 声明的变量只在当前方法中生效，因此 inner 中的声明不能作用于 inner 以外函数中的变量，就是说 inner 中的 num 与 f 中的 num 不是同一个变量。而打印 num2 是在 f 方法中打印的，因此还是输出 1；

5. 最后的 num0 打印的全局变量，由于已经被更新为 2，因此输出为 2；

结果：

```
num1:: 1
num2:: 1
num0:: 2
```

下图为变量更新流程图：



再举个例子巩固一下：

```
name = 0
def readonly():
  name = 10
  print("readonly: ", name)

def readwrite():
  global name
  name = 20
  print("readwrite:", name)
  global new_name
new_name = 30

readonly()
print("outer1: ", name)
readwrite()
print("outer2: ", name)
print("new_name: ", new_name)
```

代码解析：

1. 首行定义全局变量 name=0；

2. 调用 readonly 方法，且方法中定义局部变量 name，没有 global 声明，与外部 name 没有任何关系，因此打印 readonly = 10；

3. 同理 outer1 打印的就是全局变量 name，没有受 readonly 影响，输出 0；

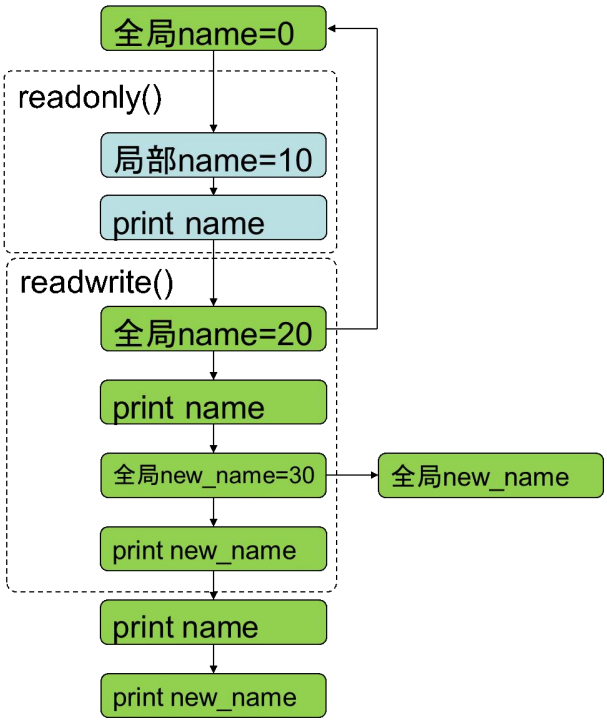
4. 调用 readwrite 方法，方法中声明 name 就是外部的全局变量 name，且更新全局变量的值为 20，因此打印 readwrite 为 20；

5. 方法 readwrite 中直接声明 new_name 为全局变量，但是前面并没有定义，这里注意，global 即使没有提前定义，也可以作为全局变量使用，因为它可以在任何地方使用；

6. 因此 outer2 的输出是全局变量 name，由于被变更为 20，因此输出 20；

7. 而 new_name 本身就是全局变量，输出为 30。

下图为变量更新流程图：



2.nonlocal 解析

一句话概括：nonlocal 的作用范围仅对于所在子函数的上一层函数中拥有的局部变量，必须在上层函数中已经定义过，且非全局变量，否则报错。

用一个例子来说明：

```
count = 0
def outer():
  count = 10
  def inner():
  nonlocal count
  count = 20
  print("count1:",count)
  def inner2():
  nonlocal count
```

```
    count = 30
    print("count2:",count)
    inner2()
    print("count3:",count)
    inner()
print("count4:",count)

outer()
print("count5:", count)
```

代码解析：

1. 首行定义全局 count=0；

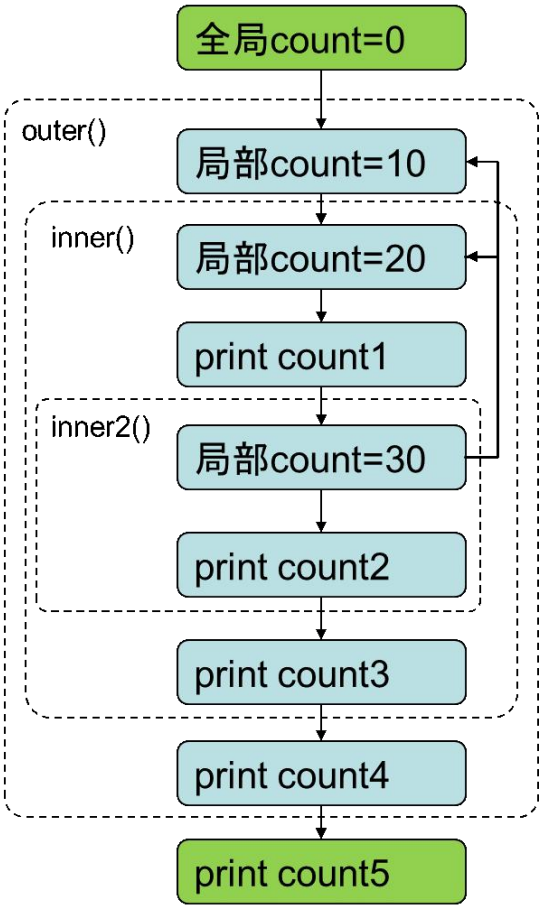
2. outer()方法中没有 global 与 nonlocal，因此为 outer 下的局部变量，值为 10；

3. Inner()方法中定义局部变量 count，且定义为 nonlocal，因此与 outer 下的 count 绑定为同一变量，且更改值为 20；

4. Inner2()方法中定义局部变量 count，且定义为 nonlocal，因此与 inter 下的 count 绑定为同一变量，且更改值为 30，因此 outer 和 inner 和 inner2 中的 count 为同一个变量；

5. 各个方法下的 print 都是打印的本方法下的局部变量，因此 count1 = 20，count2=30，count3=30，count4=30，而 count5=0。（因为全局的 count 没有与局部的 count 产生交集）

下图为变量更新流程图：



再看一个例子：

```
n = 0
def f():
  def inner():
  nonlocal n
  n = 2
  n = 1
  print("n1=",n)
  inner()
  print("n2=",n)
if __name__ == '__main__':
  f()
  print("n3=",n)
```

代码解析:

这个案例乍一看 inner 方法中的 n 之前在这之前没有定义啊，其实代码运行时，是先运行 n=1，再运行 inner 方法的。这边要注意的是，如果把 n=1 这行去掉，程序就会报错，因为 nonlocal 定义的变量在方法外必须要有定义。

1. 首先定义全部变量 n=0；

2. 方法 f()中定义局部变量 n=1，没有 nonlocal 或 global，因此与外面的 n 没有关系，因此 n1=1；

3. 调用方法 inner,方法中定义局部变量 n，且为 nonlocal，即与 f 方法中的 n 绑定为同一变量，因此修改了 f 中的变量 n 值为 2，因此打印 n2=2；

4. 由于 f 中的 n 与外部的 n 没有关系，因此 n3=0，为原来的值。

总结

通过以上几个例子，可以理解 nonlocal 与 global 的区别了吧，总结几点就是：

1.global 可以用于任何地方，声明变量为全局变量；声明后再修改，则修改了全局变量的值。

2. nonlocal 的作用范围仅在于所在子函数的上一层函数中拥有的局部变量，且非全局变量。

3. global 可以不需要提前定义，而 nonlocal 必须要提前定义。

我们还能有怎么样的测试要点

◆ 作者：刘晓佳

今天想跟大家分享一个案例吧，关于一个测试场景引发的测试要点思考。

废话不多说，上菜！

情况介绍~

某天，开发同事提交了一个功能优化单，大概情况就是：为了节省内存使用，开启了一个配置，配置项有自动切换、不切换和固定切换三种模式。当达到一定阈值阀门时，会将在内存临时存储的变量写入 mysql，使用完后即删。我们假设配置模式是 auto、no、yes。

这个功能优化的原因在于：存储的变量在大数据的情况下会占用很大的临时内存，造成内存浪费。而存入 mysql 后，内存使用会大大下降。

那么，针对这样的一个优化功能，我们可以怎么设计测试要点呢？或者说需要测试哪些呢？

过来一起看看？

暂不说性能测试，我们单纯讨论下功能测试要点。

我想大家肯定会说：三个不同配置一定要测！这是必然的，我们得至少证明配置生效了不是？！那么我们先来看一看。

1)不同配置项测试

存在三个配置项 auto、no、yes，那么我们依次测试。

• 首先测试 no，不切换

该模式下，临时变量（假设为 monitor）会存入内存。因为无法检测区分 monitor 变量占用的内存变化（除非从代码层加入临时打印），所以对于 no 配置的测试我们重点在于监控日志打印（有该变量的输出），以及已有功能回归（确保不会因为新配置的引入导致故障的产生）。

总结：本次测试要点为已有功能的回归测试。

• 测试 yes，切换

该模式下，临时变量 monitor 会存入 mysql 的某个表，且在使用完成后，该变量会清除。对于此，我们能够很方便的从 mysql 观测到。因此我们的重点除了已有功能的回归测试外，还有变量存数据库，存入变量值的正确性，以及及时清除。

那么如何确定变量值的正确性？可以通过构造特定数据，产生期望的 monitor 值，然后观察是否存入了 mysql。

那么如何确保及时清楚？这里可以分为正常情况下的清除（比如调用程序使用完成后清除）和非正常情况下的清除（比如调用程序瘫痪，无法消费该 monitor 变量，那么是否能在一定时间内清除）。

除此之外，我们还容易遗漏的一个测试要点！在并发的情况下，写入 mysql 的变量值有没有可能错乱？由此导致调用程序消费出错以及不能及时清除。

总结：本次测试要点为已有功能的回归测试，存入变量值的正确性，变量存数据库和及时清除，以及在并发情况下的测试。

• 测试 auto，自动切换

在上两个配置项测试的基础上，我们主要需要测试阈值阀门有效性。也就是当某个变量（比如本例中的 num）到达一定值时，是否能打开自动切换功能。

但在此，值得主意的一点是：往往，我们的阈值会设置的比较大，而测试环境可能达不到这样的阈值，那么怎么测试？可以采取调下阈值。比如原有配置 num>1000 开启自动切换，我们可以配置阈值 num>100 时开启自动切换，以此验证该功能正确性。

除此之外呢，我们还需要把 no 和 yes 的测试要点回归一下。

总结：本次测试要点为已有功能的回归测试，存入变量值的正确性，变量存数据库和及时清除，以及在并发情况下的测试，还有阈值上下边界测试。

除此之外呢？还有没有？！

当然有！默认配置呢？！无配置测试呢？！

• 默认配置测试

在本例中，默认配置是 auto，那么在测 auto 的时候也就把默认配置测了。那对于有的配置，假如默认配置为空呢？！要知道对于空的处理很容易产生空指针之类的故障。所以，我们一定不要忘记测试默认配置！

上面所说的还都算是中规中矩比较正常的测试，那么，我们来当个“反叛者”？做做破坏性测试？

2)探索性破坏测试

回忆一下，这个案例中涉及了哪些程序或组件？mysql，产生 monitor 的程序 A，以及调用写入 mysql 的 monitor 变量的程序 B。

发出邪恶的笑声吧~

• 停止或重启 mysql

在 A 程序正常产生 monitor 的过程中，使用 auto 或 yes 配置，将 mysql 停止或重启，试试能不能正常写入，或者看看 A 程序是否检测到了异常，并正确处理了异常。

• 停止或重启调用写入 mysql 的 monitor 变量的程序 B

monitor 变量写入 mysql 后，将 B 程序停止或重启，看看变量能否正常消费或及时清除。

• 产生 monitor 的程序 A

有人可能会问：我听它干嘛？你不想看看要是 monitor 值有很多个时，写入 mysql 写了一半，A 程序停止或重启了是个什么样的情况？就比如 A 的上游程序是否会有异常打印或者重试策略？

那么还有吗？

别忘了我们为什么优化。

3)优化指标对比

因为程序运行过程中，很难区分具体变量的内存占用，但如果我们想要测试，可以试试求助开发人员，帮助我们打印或监控下内存的消耗。

因为要是有人声称：我的改动可以节省 10 倍内存，你会不会怀疑？

先总结一下！

我们把上面分析出的测试要点先总结下吧。

测试要点	要点细分	建议
不同配置项测试	首先测试 no，不切换	已有功能的回归测试
	测试 yes，切换	已有功能的回归测试 存入变量值的正确性，变量存数据库和及时清除 并发情况下的测试 阈值上下边界测试
	测试 auto，自动切换	已有功能的回归测试 存入变量值的正确性，变量存数据库和及时清除 并发情况下的测试 阈值上下边界测试
	默认配置测试	测试默认配置，尤其是空配置
探索性破坏测试	停止或重启 mysql	注意观察日志异常打印和程序面对异常情况下的正确处理
	停止或重启调用写入 mysql 的 monitor 变量的程序 B	
	产生 monitor 的程序 A	
优化指标对比	内存消耗对比	没有合适的监控方法，可以求助开发人员

说了这些，你觉得还有什么吗？别害羞，一起过来讨论讨论。

聊聊网络性能测试二三事

◆作者：Explorer

1、背景

当下云计算、大数据盛行的背景下，大并发和大吞吐量的需求已经是摆在企业面前的问题了，其中网络的性能要求尤为关键，除了软件本身需要考虑到性能方面的要求，一些硬件上面的优化也是必不可少的。

作为一名测试工作者，对于性能测试的问题肯定不会陌生，但是测试不仅仅是执行和收集数据，更多的应该是分析问题、找到性能瓶颈以及一些优化工作。毕竟在客户现场测试性能的时候，能够通过一些系统层面的调优，提升软件的性能，那对项目无疑是一件锦上添花的事。

2、指标

不管你做何种性能测试，指标是绕不过去的事，指标是量化性能测试的重要依据。衡量某个性能的指标有很多，比如衡量数据库性能通常用 TPS、QPS 和延时，衡量 io 性能通常用 iops、吞吐量和延时，衡量网络的性能指标也有很多，在业界有一个 RFC2544 的标准，参考链接：https://www.rfc-editor.org/rfc/rfc2544

这里稍微解释一下几个指标

• 背靠背测试：在一段时间内，以合法的最小帧间隔在传输介质上连续发送固定长度的包而不引起丢包时的数据量。

• 丢包率测试：在路由器稳定负载状态下，由于缺乏资源而不能被转发的帧占所有该被转发的帧的百分比。

• 时延测试：输入帧的最后一位到达输入端口到输出帧的第一位出现在输出端看的时间间隔。

• 吞吐量测试：没有丢包情况下能够转发的最大速率。

通常客户对于吞吐量和时延的指标是比较关心的，吞吐量反应了系统的并发的处理能力，时延反应了整体业务的响应时间。

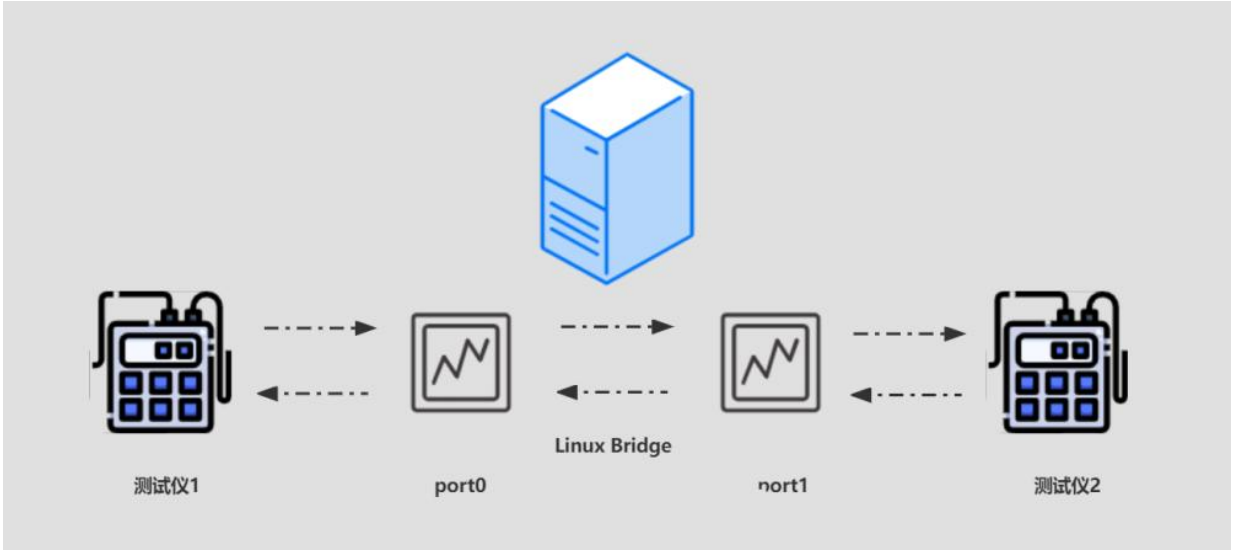
3、测试准备

本文以网卡中的战斗机 intel X710 为测试对象，进行小包的吞吐量测试。

Tips：通常大包的测试可以达到线速，小包则很难线速，相同 mtu 下，包越小单位时间内 cpu 处理的包数越多，cpu 的压力越大。

3.1 测试拓扑

在被测设备上插入 X710 网卡，使用网卡的 2 个网口建立 linux bridge，通过测试仪发送接收数据包




```
[root@localhost ~]# brctl addbr test
[root@localhost ~]# brctl addif test enp11s0f2 enp11s0f3
[root@localhost ~]# ip link set test up
[root@localhost ~]# brctl show
bridge name     bridge id               STP enabled     interfaces
```

test	8000.6cb311618c30	no	enp11s0f2 enp11s0f3
virbr0	8000.5254004c4831	yes	virbr0-nic

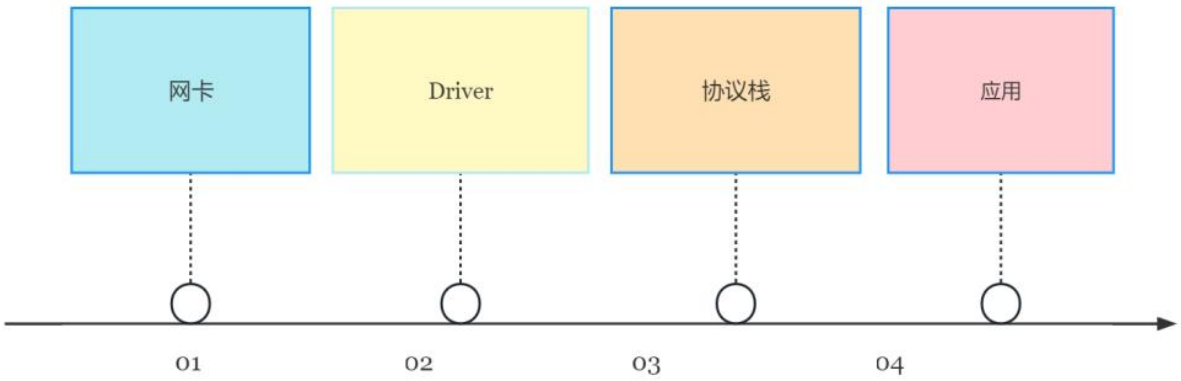
3.2 硬件信息

有个伟人曾说过，“如果性能测试不谈硬件，那么就和恋爱不谈结婚一个道理，都是耍流氓”。

鲁迅:"我没说过！"

题外话，这里先简单说一下数据包进入网卡后的流程：

数据包进入到网卡后，将数据缓存到服务器内存后通知内核进行处理，接着协议栈进行处理，通常 netfilter 也是在协议栈去处理，最后应用程序从 socker buff 读取数据。



言归真正，本文测试采用的是国产之光飞腾 S2500 服务器。

硬件配置	型号
CPU	Phytium,S2500/64
OS	Kylin Linux Advanced Server release V10 (Tercel)
kernel	4.19.90-23.27.v2101.ky10.aarch64
NIC	intel X710

4、测试调优

针对以上拓扑，发送 128 字节万兆双向流量，用 RFC2544 标准进行测试。

帧长(字节)	目标负载 (%)	实际负载 (%)	吞吐量 (%)	吞吐量 (fps)	吞吐量 (Mbps)
128	4.094	4.094	4.094	691554	818.8

RFC2544 的标准是 0 丢包测试吞吐量，结果是 818Mbps

以上都是前菜，接下来是本文的重头戏，根据已有资源做一些优化，提升“测试”性能，让客户满意。

4.1 调整队列数

调整网卡的队列数，利用网卡的多队列特性，不同的队列通过中断绑定到不同的 cpu，提升 cpu 处理性能，提升网络带宽。

调整队列数，这里也不是越大越好，因为服务器的 cpu 个数是有上限的，队列多的话会出现多个中断绑定在同一个 cpu 上，这里我服务器单个 numa 有 8 个 cpu，修改队列数为 8。

```
[root@localhost ~]# ethtool -l enp11s0f2
Channel parameters for enp11s0f2:
Pre-set maximums:
RX:             0
TX:             0
Other:          1
Combined:       128
Current hardware settings:
RX:             0
TX:             0
Other:          1
Combined:       8
```

因为数据包会根据 hash 来进入到多个收包队列，因此发送数据包的时候，可以选择源 ip 变化的流来确保使用了网卡的多队列，可以查看网卡的队列计数。

```
[root@localhost ~]# ethtool -S enp11s0f3
NIC statistics:
     rx_packets: 4111490
     tx_packets: 4111490
     rx_bytes: 509824504
     tx_bytes: 509824504
     rx_errors: 0
     tx_errors: 0
     rx_dropped: 0
     tx_dropped: 0
     collisions: 0
     rx_length_errors: 0
     rx_crc_errors: 0
     rx_unicast: 4111486
     tx_unicast: 4111486
   ...

     rx-2.rx_bytes: 63718020
     tx-3.tx_packets: 513838
     tx-3.tx_bytes: 63715912
     rx-3.rx_packets: 514847
     rx-3.rx_bytes: 63841028
     tx-4.tx_packets: 1027488
     tx-4.tx_bytes: 127408512
     rx-4.rx_packets: 513648
     rx-4.rx_bytes: 63692352
     tx-5.tx_packets: 52670
     tx-5.tx_bytes: 6530824
     rx-5.rx_packets: 514388
     rx-5.rx_bytes: 63784112
     tx-6.tx_packets: 1029202
     tx-6.tx_bytes: 127621048
     rx-6.rx_packets: 514742
```

rx-6.rx_bytes: 63828008

tx-7.tx_packets: 460901

tx-7.tx_bytes: 57151724

rx-7.rx_packets: 512859

修改完队列后再次使用 RFC2544 进行测试，性能比单队列提升了 57%。

帧长(字节)	目标负载 (%)	实际负载 (%)	吞吐量 (%)	吞吐量 (fps)	吞吐量 (Mbps)
128	6.438	6.438	6.438	1087500	1287.6

4.2 调整队列深度

除了调整网卡的队列数外，也可以修改网卡的队列深度，但是队列深度不是越大越好，具体还是需要看实际的应用场景。对于延时要求高的场景并不适合修改太大的队列深度。

查看当前队列深度为 512，修改为 1024 再次测试：

```
[root@localhost ~]# ethtool -g enp11s0f2
Ring parameters for enp11s0f2:
Pre-set maximums:
RX:             4096
RX Mini:        0
RX Jumbo:       0
TX:             4096
Current hardware settings:
RX:             512
RX Mini:        0
RX Jumbo:       0
TX:             512
```

修改 tx 和 rx 队列深度为 1024：

```
[root@localhost ~]# ethtool -G enp11s0f3 tx   1024
[root@localhost ~]# ethtool -G enp11s0f3 rx   1024
```

```
[root@localhost ~]# ethtool -g enp11s0f2
Ring parameters for enp11s0f2:
Pre-set maximums:
RX:             4096
RX Mini:        0
RX Jumbo:       0
TX:             4096
Current hardware settings:
RX:             1024
RX Mini:        0
RX Jumbo:       0
TX:             1024
```

再次进行 RFC2544 测试，这次也是有相应的提升。

帧长(字节)	目标负载 (%)	实际负载 (%)	吞吐量 (%)	吞吐量 (fps)	吞吐量 (Mbps)
128	7.344	7.344	7.344	1240540	1468.8

4.3 中断绑定

网卡收包后，内核会触发软中断程序，如果此时运行中断的 cpu 和网卡不在一个 numa 上，则性能会降低。

查看网卡 pci 插槽对应 numa node：

```
[root@localhost ~]# cat /sys/bus/pci/devices/0000\:0b\:00.3/numa_node
0
```

这里对应的 numa node 为 0

飞腾 S2500 有 16 个 numa node，node8-15 和 node0 所在的物理 cpu 不同，如果中断跑在上面性能会更加低

```
[root@localhost ~]# numactl -H
available: 16 nodes (0-15)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 65009 MB
node 0 free: 46721 MB
```

node 1 cpus: 8 9 10 11 12 13 14 15

node 1 size: 0 MB

node 1 free: 0 MB

node 2 cpus: 16 17 18 19 20 21 22 23

node 2 size: 0 MB

node 2 free: 0 MB

node 3 cpus: 24 25 26 27 28 29 30 31

node 3 size: 0 MB

node 3 free: 0 MB

node 4 cpus: 32 33 34 35 36 37 38 39

node 4 size: 0 MB

node 4 free: 0 MB

node 5 cpus: 40 41 42 43 44 45 46 47

node 5 size: 0 MB

node 5 free: 0 MB

node 6 cpus: 48 49 50 51 52 53 54 55

node 6 size: 65466 MB

node 6 free: 64663 MB

node 7 cpus: 56 57 58 59 60 61 62 63

node 7 size: 0 MB

node 7 free: 0 MB

node 8 cpus: 64 65 66 67 68 69 70 71

node 8 size: 65402 MB

node 8 free: 63691 MB

node 9 cpus: 72 73 74 75 76 77 78 79

node 9 size: 0 MB

node 9 free: 0 MB

node 10 cpus: 80 81 82 83 84 85 86 87

node 10 size: 0 MB

node 10 free: 0 MB

node 11 cpus: 88 89 90 91 92 93 94 95

node 11 size: 0 MB

node 11 free: 0 MB

node 12 cpus: 96 97 98 99 100 101 102 103

node 12 size: 0 MB

node 12 free: 0 MB

node 13 cpus: 104 105 106 107 108 109 110 111

node 13 size: 0 MB

node 13 free: 0 MB

node 14 cpus: 112 113 114 115 116 117 118 119

node 14 size: 64358 MB

node 14 free: 63455 MB

node 15 cpus: 120 121 122 123 124 125 126 127

node 15 size: 0 MB

node 15 free: 0 MB

node distances:

node	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0:	10	20	40	30	20	30	50	40	100	100	100	100	100	100	100	100
1:	20	10	30	40	50	20	40	50	100	100	100	100	100	100	100	100
2:	40	30	10	20	40	50	20	30	100	100	100	100	100	100	100	100
3:	30	40	20	10	30	20	40	50	100	100	100	100	100	100	100	100
4:	20	50	40	30	10	50	30	20	100	100	100	100	100	100	100	100
5:	30	20	50	20	50	10	50	40	100	100	100	100	100	100	100	100
6:	50	40	20	40	30	50	10	30	100	100	100	100	100	100	100	100
7:	40	50	30	50	20	40	30	10	100	100	100	100	100	100	100	100
8:	100	100	100	100	100	100	100	100	10	20	40	30	20	30	50	40
9:	100	100	100	100	100	100	100	100	20	10	30	40	50	20	40	50
10:	100	100	100	100	100	100	100	100	40	30	10	20	40	50	20	30
11:	100	100	100	100	100	100	100	100	30	40	20	10	30	20	40	50
12:	100	100	100	100	100	100	100	100	20	50	40	30	10	50	30	20
13:	100	100	100	100	100	100	100	100	30	20	50	20	50	10	50	40
14:	100	100	100	100	100	100	100	100	50	40	20	40	30	50	10	30
15:	100	100	100	100	100	100	100	100	40	50	30	50	20	40	30	10

查看网卡对应的中断绑定。

tips：系统会有一个中断平衡服务，系统会根据环境负载情况自行分配 cpu 到各个中

断，所以这里为了强行把中断平均的绑定到各个 cpu，需要先停止该服务：

```
[root@localhost ~]# cat /proc/interrupts | grep enp11s0f3 | cut -d: -f1 | while read i; do echo -ne irq":$i\t bind_cpu: "; cat /proc/irq/$i/smp_affinity_list; done | sort -n -t' ' -k3
irq:654    bind_cpu: 0
irq:653    bind_cpu: 1
irq:656    bind_cpu: 2
irq:657    bind_cpu: 2
irq:659    bind_cpu: 3
irq:655    bind_cpu: 4
irq:652    bind_cpu: 5
irq:658    bind_cpu: 6

[root@localhost ~]# systemctl    stop irqbalance.service
```

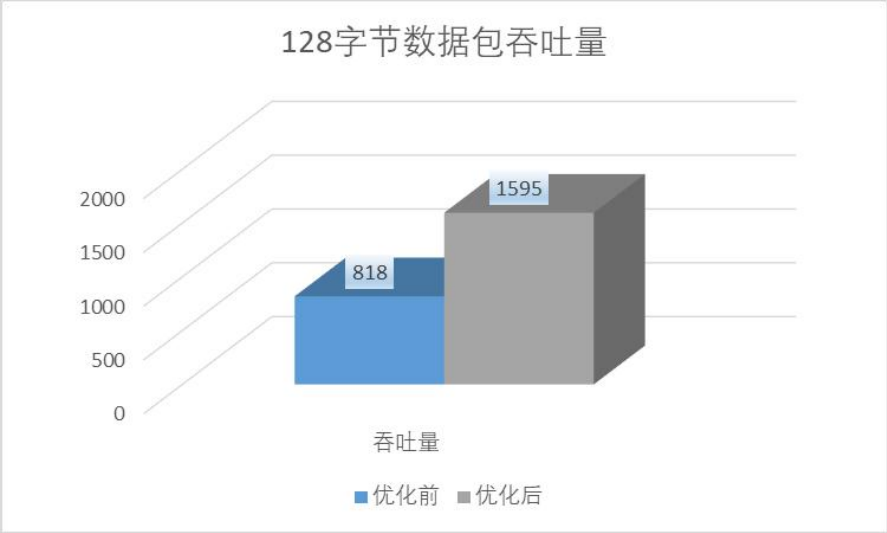
手动修改网卡对应的 cpu 亲和性：

```
echo 0 > /proc/irq/654/smp_affinity_list
echo 1 > /proc/irq/653/smp_affinity_list
echo 2 > /proc/irq/656/smp_affinity_list
echo 3 > /proc/irq/657/smp_affinity_list
echo 4 > /proc/irq/659/smp_affinity_list
echo 5 > /proc/irq/655/smp_affinity_list
echo 6 > /proc/irq/652/smp_affinity_list
echo 7 > /proc/irq/658/smp_affinity_list
```

调整后继续测试 RFC2544，又有了一定的性能提升。

帧长(字节)	目标负载 (%)	实际负载 (%)	吞吐量 (%)	吞吐量 (fps)	吞吐量 (Mbps)
128	7.977	7.977	7.977	1347466	1595.4

通过以上三种优化方式后，性能提升了 95%，很显然，如果发生在客户现场，那绝对是值得高兴的一件事。



4.4 其他优化项

除了以上几种方式外，还有一些日常的调优手段，大家可以试一下针对不同的场景，选择不同的方式。

4.4.1 打开 tso 和 gso

利用网卡硬件进行分片或者推迟协议栈分片，以此来降低 cpu 负载，提升整体性能。

4.4.2 调整 TLP

调整 pcie 总线每次数据传输的最大值，根据实际情况调整，bios 中可以修改。

4.4.3 调整聚合中断

合并中断，减少中断处理次数，根据实际情况调整。

4.4.4 应用程序 cpu 绑定

如果测试是使用类似 netperf，qperf 的工具，可以使用 taskset 命令绑定该测试进程到指定 cpu。

5、总结

随着性能测试的发展以及对测试工程师的要求提高，优化性能已经不再是单纯开发同学所要做的事情，使用合适的测试方法和测试工具进行测试，收集数据找到性能瓶颈，并能进行一系列的调优，这才是性能测试团队做的真正有意义以及有价值的事情。

拓展学习

[1] nmon 下的性能测试监控报告优化

https://www.atstudy.com/course/1012156

[2] 全方位多终端高级性能测试实战学习

咨询：微信 atstudy-js 备注：性能

通过钉钉机器人发送禅道缺陷标题

◆ 作者：测试安静

前言：目前大多数公司都是使用禅道，jira 这些来管理缺陷，研发和测试每天站会或者周会都想知道昨天或者这周一共解决了多少个缺陷，如果每天都通过禅道上去查看可能有点麻烦且不方便，今天小编介绍一种方法，我们可以通过办公软件钉钉或者企业微信通过项目群中进行添加机器人，每天自动发送到群里，供大家参考查看。

钉钉机器人推送消息

小编目前公司用的是钉钉，就先拿钉钉来介绍了，方法都是类似的，首先我们先创建一个群聊，然后在群里添加对应的机器人，可以通过“群设置”--->“机器人”--->“添加机器人”，这里选择添加自定义的机器人服务

添加完成后，我们值需要设置机器人的名称，安全设置中，选择自定义关键词，然后设置我们的关键词，这里小编用的是缺陷名称

添加完成后，一定要记得将 webhook 复制，关于推送消息，主要通过这里。

python 发送消息

其中钉钉机器人发送消息有很多类型，文本，link 链接，Markdown，图片等供我们进行使用，这里小编通过简单的文本形式进行操作。通过查看官方文档，可以看到我们如何进行发送请求。

```
curl 'https://oapi.dingtalk.com/robot/send?access_token=xxxxxxxx' \
-H 'Content-Type: application/json' \
-d '{"msgtype": "text","text": {"content":"我就是我，是不一样的烟火"}}'
```

其中这里的 url 地址怎么好熟悉，其实这个 url 地址就是我们复制的 webhook 的地址，这里我们只需要请求这个地址，然后将对应的参数填写上去就可以了。

```
# coding:utf-8
import requests
import json
# 这里的地址是机器人的地址 webhook 地址，自行申请使用
url = 'https://oapi.dingtalk.com/robot/send?access_token=xxxxxxxx'
headers = {'Content-Type': 'application/json'}
# 参数内容
data = {"msgtype":
            "text","text":
    {"content":"缺陷名称：今天你有学习吗？"},
        "at": {
            # 要@的人
            "atMobiles": " ",
            # 是否@所有人
            "isAtAll": False
}
        }
r = requests.post(url,headers=headers,data=json.dumps(data))
print(r.status_code)
print(r.text)
```

执行后，这个时候就会看到钉钉群里，已经自动发送了消息。到这里，我们的本次的内容已经完成一部分了，下面就要通过读取禅道的数据库了。

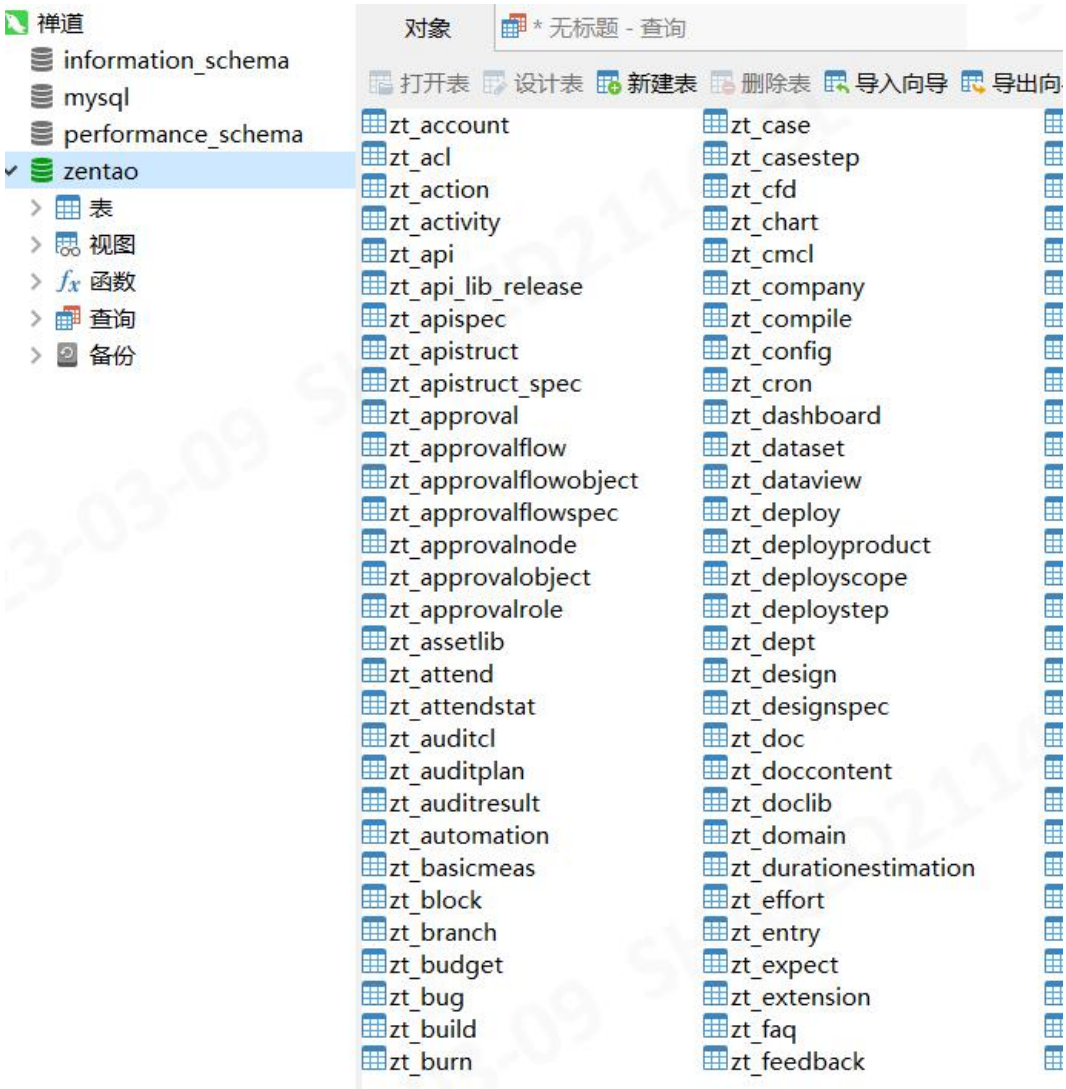
python 读取禅道数据库

首先安装禅道，这里小编就不介绍了，大家可以自行百度下载禅道后并安装，安装完成后，我们需要进入到禅道中，进行创建项目和缺陷。

Bug标题	级别	P	状态	创建者
这是一个轻微的缺陷		3	激活	admin
这是一个致命的缺陷		3	激活	admin
这是一个严重的缺陷		3	激活	admin

连接数据库

先通过 navicat 添加禅道数据库，因为是在本地的直接添加的，如果是服务器上，请连接具体服务器的主机，需要先添加禅道的数据库 zentao，然后进行查看。



通过禅道官方文档查看每个表中具体表示什么，这里我们只需要找到 bug 表就行。

四、测试相关的表

- zt_bug，bug表，也是大家非常熟悉的一张表了。
- zt_case，用例表。记录了所有的测试用例。
- zt_casestep，则是记录了用例相关的步骤，包括历史。
- zt_testtask，测试版本表，记录了历次的测试任务。
- zt_testrun，则记录了每个测试任务所对应的用例执行情况。
- zt_testresult，记录了每个用例历次执行的结果。
- zt_testsuite，测试套件表。
- zt_suitecase，套件用例表。
- zt_testreport，测试报告表。

查询数据

数据库也成功连接了，对应的数据表也找到了，先查看下数据是否正确，通过 navicat 编写查询语句进行查询，这里小编只查询到缺陷标题。

python 连接数据库

这里小编通过 pymysql 这个 python 的第三方库进行连接数据库

安装：pip install pymysql

这里安静就直接上代码了，思路都是一样的，通过连接数据库，然后进行查询数据。

```
import pymysql
# 连接数据库
count = pymysql.connect(
                host = '127.0.0.1',       # 数据库地址
                port = 3306,              # 数据库端口号
                user='root',              # 数据库账号
                password='821006052',     # 数据库密码
                db= 'zentao',   # 数据库表名
                charset = 'gbk'       # 中文乱码
)
# 完成 mysql 数据库实例化
db = count.cursor()
# sql 语句
sql = 'select title from zt_bug'
# 执行 sql
a = db.execute(sql)
# 查找所以内容
result = db.fetchall()
print(result)
```

执行代码缺陷标题已经全部都查询出来，接下来就是将我们钉钉机器人推送消息和数据库查询到的数据进行结合，就达到了我们的需求了。

```
(('这是一个严重的缺陷',), ('这是一个致命的缺陷',), ('这是一个轻微的缺陷',))

Process finished with exit code 0
```

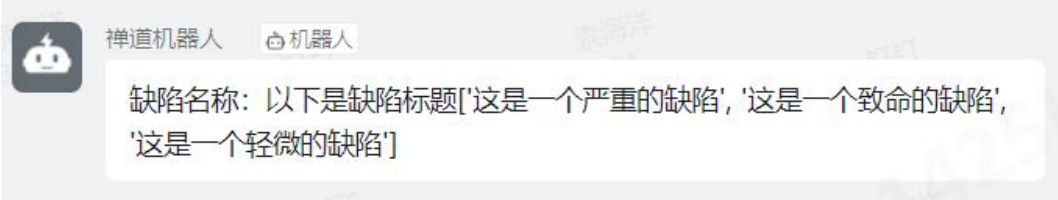
python 实现钉钉机器人推送缺陷内容

上面每个模块都已经完成了，后面只需要我们一点点的调式了，小编这边直接将调试完成的代码进行运行起来。

```
# coding:utf-8
import pymysql
import requests
import json
# 连接数据库
def MYsql(sql):
    a = []
    count = pymysql.connect(
                host = '127.0.0.1', # 数据库地址
                port = 3306,      # 数据库端口号
                user='root',      # 数据库账号
                password='821006052',      # 数据库密码
                db = 'zentao')      # 数据库表名
    # 创建数据库对象
    db = count.cursor()
    # 写入 SQL 语句
    sql_ = sql
    # 执行 sql 命令
    db.execute(sql_)
    # 获取全部的查询内容
    restul = db.fetchall()
    # 将查询的结果添加到列表中
    for i in restul:
        a.append(i[0])
    db.close()
    return a
def DingDing(text):
    # 钉钉机器人的 webhook 的地址
    url = 'https://oapi.dingtalk.com/robot/send?access_token=xxxxxxxxx'
```

```
        headers = {'Content-Type': 'application/json'}
        data = {"msgtype":
                    "text", "text":
                    {"content": "缺陷名称：以下是缺陷标题%s" %text},
                "at": {
                    # 要@的人
                    "atMobiles": "",
                    # 是否@所有人
                    "isAtAll": False
                }
                }
        r = requests.post(url, headers=headers, data=json.dumps(data))
        print(r.status_code)
        print(text)
result = MYsql("select title from zt_bug")
DingDing(result)
```

点击执行程序，并且观察钉钉群中的变化，发现钉钉机器人已经成功的将我们的数据发送到了群里。到这里小编已经实现了该功能，只不过小编这里是通过获取缺陷标题，大家可以根据未解决的缺陷和解决的缺陷，将一些缺陷标号，严重程度，一步步实现起来。方法都是类似的。



禅道连接钉钉机器人

上述方法介绍了通过 python 将禅道的信息和钉钉机器人连接起来了，但是有的朋友可能感觉会不会太麻烦了，那么小编在介绍一种方法，直接通过禅道将我们的信息通过链接的形式发送给钉钉微信群中。

添加钉钉群机器人

我们在刚才的群里在新建一个钉钉机器人，这次我们通过安全设置中的加签的形式进行创建，创建完成后，分别复制 webhook，和加签产生的数据。

禅道添加钉钉机器人

打开禅道，进入到"后台"---> “通知设置” ---> “Webhook”，将上述步骤中所复制的 webhook 和加签的数据放入到禅道中的 webhook 中，其中关联产品和关联执行，可以自行获取想要接收到那些信息，如果填空，则说明均可以接收到。

创建成功后，我们直接在禅道对应的产品下，进行添加一个缺陷，缺陷成功添加后，就会发现钉钉群中已经将我们刚创建的缺陷发出来了。

总结

小编这里通过两种方法进行实现了如果将禅道上的缺陷内容和钉钉机器人联调起来，实现每天或者每周都可以很清楚的通过钉钉群了解到项目的缺陷列表。非常感谢您的阅读，希望对您有所帮助。

自动化中如何增加 log 日志功能

◆作者：测试安静

前言：
在自动化操作过程中，我们可以通过增加 log 日志的情况进行更加直观的了解我们测试用例的执行情况，包括执行状态，方便排查问题和分析问题，通常在自动化中增加日志大家最常用的可能就是直接通过 print，但是这个调试方面，对于真正跑自动化的过程中可能不太方面，小编今天简单介绍两种，一种通过 pytest 方法的来添加日志，一种通过装饰器的形式添加日志情况。两种情况可能各有不同，大家一起来看下吧。

pytest

这里小编通过 pytest.ini 这个功能来实现添加日志，pytest.ini 文件是 pytest 的主配置文件，可以改变 pytest 的运行方式，且是一个固定的文件 pytest.ini 文件，pytest.ini 一般存放在项目的根目录中。其中 pytest.ini 有很多参数配置，小编今天主要介绍 log_cli 这个方法来实现增加日志功能。

log_cli

log_cli：表示在执行过程中是否启动实时监测日志，默认为 Flase，我们可以通过 pytest -h?，查看下基本介绍。

```
doctest_encoding (string):
                      Encoding used for doctest files
cache_dir (string):   Cache directory path
log_level (string):   Default value for --log-level
log_format (string):  Default value for --log-format
log_date_format (string):
                      Default value for --log-date-format
log_cli (bool):       Enable log display during test run (also known as "live logging")
log_cli_level (string):
                      Default value for --log-cli-level
log_cli_format (string):
                      Default value for --log-cli-format
log_cli_date_format (string):
                      Default value for --log-cli-date-format
log_file (string):    Default value for --log-file
log_file_level (string):
                      Default value for --log-file-level
log_file_format (string):
                      Default value for --log-file-format
log_file_date_format (string):
                      Default value for --log-file-date-format
log_auto_indent (string):
                      Default value for --log-auto-indent
pythonpath (paths):   Add paths to sys.path
faulthandler_timeout (string):
                      Dump the traceback of all threads if a test takes more than TIMEOUT seconds to finish
addopts (args):       Extra command line options
```

接下来安静通过实例来介绍下如何通过 pytest 进行添加 log 信息

pytest.ini

首先我们需要创建 pytest.ini 文件，在文件中添加对应的配置信息：

log_cli = True：表示是否实时打开 log 监测，默认为 False。

log_cli_level： 表示监测 log 日志等级显示。

log_cli_format： 表示输出 log 日志显示格式。

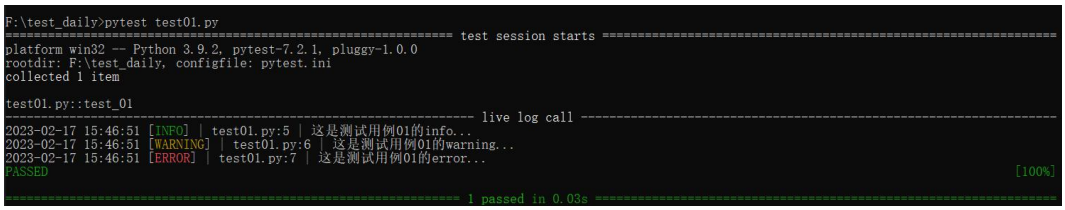
log_cli_date_format： 表示显示 log 时间。

```
[pytest]
log_cli = True
log_cli_level = INFO
log_cli_format = %(asctime)s [%(levelname)s] | %(filename)s:%(lineno)s | %(message)s
log_cli_date_format = %Y-%m-%d %H:%M:%S
```

在 pytest.ini 文件创建完成后，我们还需要在编写测试用例前需要导入 loging 库，这个库书写详细的 log 信息：

```
# test01.py
# coding:utf-8
import logging
def test_01():
    logging.info('这是测试用例 01 的 info...')
    logging.warning('这是测试用例 01 的 warning...')
    logging.error('这是测试用例 01 的 error...')
    assert 1 == 1
```

通过命令行进行执行：



通过上面的执行结果可以看到，我们已经将对应的 log 信息显示出来了。

log 日志实时写入文件中

上面介绍的是讲 log 信息显示在控制台上，现在我不想让他显示在控制台中，想要显示在对应的文件中，这里 pytest.ini 也可以进行配置，详细信息也是可以通过 pytest -h 中查看：

log_file：表示存放文件的路径。

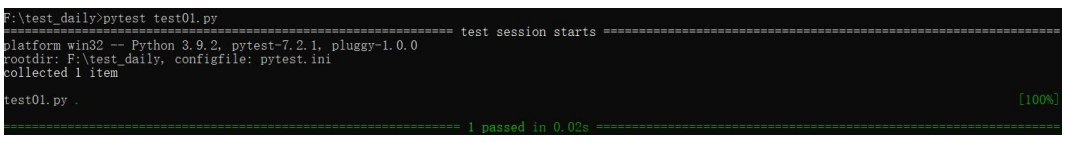
log_file_level ：表示文件中显示的日志等级。

log_file_date_format：表示文件中显示的 log 时间。

log_file_format：表示文件中显示 log 格式。

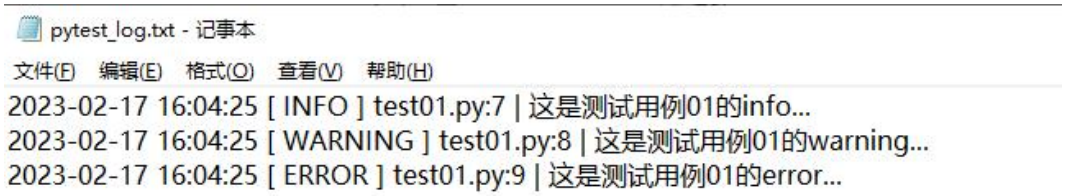
```
[pytest]
log_file =pytest_log.txt
log_file_level = INFO
log_file_date_format = %Y-%m-%d %H:%M:%S
log_file_format = %(asctime)s [ %(levelname)s ] %(filename)s:%(lineno)s | %(message)s
```

pytest.ini 配置文件编写完成后，直接通过命令行进行执行：



发现 log 信息已经不再控制台显示了，当前目录下生成了一个 log 文件，记录了当前

的信息：



装饰器

大家首先会问什么是装饰器，装饰器其实就是装饰对象的器件。可以在不修改原有代码的情况下，为被装饰的对象增加新的功能或者附加限制条件或者帮助输出装饰对象的器件。可以在不修改原有代码的情况下，为被装饰的对象增加新的功能或者附加限制条件或者帮助输出。

使用方法：直接在将要被装饰的对象上输入@装饰器名称

```
# 定义一个装饰器
def fun(foo):
    def add():
        print('日志打印')
        foo()
        print('日志上传成功')
    return add
@fun
def test01():
    print('这是自动化用例 01')

@fun
def test02():
    print('这是自动化用例 02')
test01()
test02()
```

通过执行代码会发现，会将我们增加的日志信息打印出来:

```
D:\python\python.exe D:/test_daily/test02.py
```

日志打印

这是自动化用例 01

日志上传成功

日志打印

这是自动化用例 02

日志上传成功

上面主要介绍下什么是装饰器，以及通过装饰器简单的实现了为测试用例增加 log 的方法。

logging

logging 属于 python 的一个库，它主要可以帮助我们实现为增加日志的目的，其中 logging 大概分为 4 个组件。

组件名称	对应类名	功能描述
日志器	loger	提供了应用程序可一直使用的接口
处理器	Handler	将 logger 创建的日志记录发送到合适的目的输出
过滤器	Filter	提供了更细粒度的控制工具来决定输出哪条日志记录，丢弃哪条日志记录
格式器	Formatter	决定日志记录的最终输出格式

日志等级

日志级别分为 NOTSET < DEBUG < INFO < WARNING < ERROR < CRITICAL

通过代码进行举例说明:

```
# coding:utf-8
import logging
```

```
# 设置打印日志级别
logging.basicConfig(level=logging.DEBUG)
logging.debug('调试模式')        # 调试模式
logging.info('基础信息')          # 基础信息
logging.warning('警告信息')       # 警告
logging.error('错误信息')         # 错误
logging.critical('严重错误信息')     # 严重错误
```

通过运行代码得到结果，其中我们可以通过设置打印显示的日志级别：

DEBUG:root:调试模式

INFO:root:基础信息

WARNING:root:警告信息

ERROR:root:错误信息

CRITICAL:root:严重错误信息

自动化增加日志

好了，到了最终实践的时候了，简单的了解装饰器和 python 的日志库，我们可以通过这里进行实现自动化增加日志的功能。

```
# coding:utf-8
import logging

# 获取日志记录器、配置日志等级
log = logging.getLogger(__name__)
log.setLevel("INFO")
# 默认日志格式
formatter = logging.Formatter("%(asctime)s - [%(levelname)s] - %(message)s")
# 输出到控制台的 handler
shl = logging.StreamHandler()
# 配置默认日志格式
```

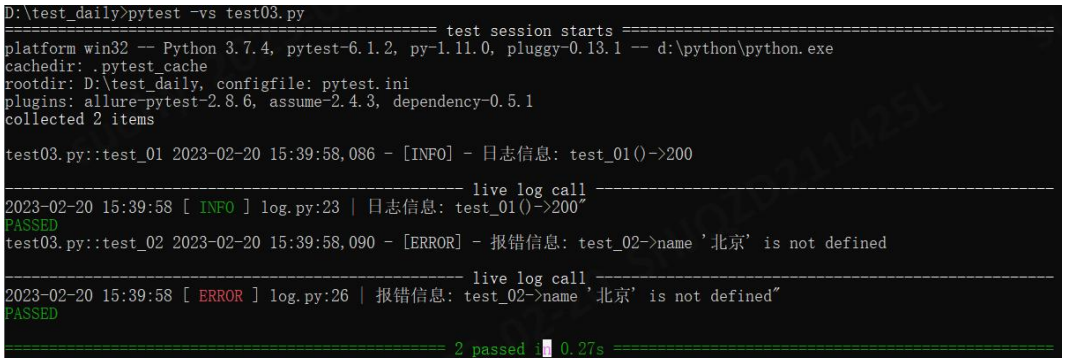
```
    shl.setFormatter(formatter)
    # 日志记录器增加此 handler
    log.addHandler(shl)


    # 装饰器
    def loger(func):
        def inner(*args, **kwargs):
            try:
                result = func(*args, **kwargs)
                log.info(f"日志信息: {func.__name__}{args}->{result}")
                return result
            except Exception as e:
                log.error(f"报错信息: {func.__name__}->{str(e)}")
        return inner
```

上面的内容是小编简单编写的生成 log 的装饰器，下面我们继续边界自动化脚本，然后加上装饰器。

```
    # coding:utf-8
    from log import loger
    import requests
    @loger
    def test_01():
        url = 'http://apis.juhe.cn/simpleWeather/query'
        data = {
            "city": '上海',
            'key': 'xxxxxxxxxx'
        }
        r = requests.post(url, data=data)
        return (r.status_code)
    @loger
    def test_02():
        url = 'http://apis.juhe.cn/simpleWeather/query'
```

```
data = {
    "city": 北京,
    'key': 'xxxxxxxxxxx'
}
r = requests.post(url, data=data)
return (r.status_code)
```

上述代码是简单的接口代码，其中 test_01 为成功的案例，test_02 为我们异常的案例，我们这里可以通过 pytest 来进行操作实现（其中 XXX 部分为聚合数据 key 大家自行申请）通过 cmd 进行运动后可以发现，我们已经简单的实现了关于自动化的日志功能。



总结

小编通过 pytest.ini 中的功能和装饰器的方法进行对 pytest 中增加 log 信息，执行自动化的过程中，更加的清楚报错信息，也方面帮助我们进行调试。顺便让大家更加理解 pytest.ini 的使用方法和装饰器的应用，感谢您的阅读，希望能给您带来帮助。