**密级： 保密期限：**



**硕士学位论文**



**题目： 基于Peach的 Fuzz策略**

**优化研究与实现**

**学 号： 2012111509**

**姓 名： 赵凯**

**专 业：计算机科学与技术**

**导 师： 温巧燕**

**学 院： 网络技术研究院**

**2014年 10月24日**

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名： 日期：

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名： 日期：

导师签名： 日期：

摘要

中文摘要（不大于500字，格式：正文1）

**关键词**：（3～5个，格式：正文）

Abstrat

Abstrat（格式：正文1）

**Key words:** (3~5 words,格式：正文)

目录

[第一章 基本操作（标题1，标号） 1](#_Toc293948319)

[1.1 该模板使用方法（格式：标题2） 1](#_Toc293948320)

[1.2 多级标题格式的应用 1](#_Toc293948321)

[1.3 参考文献的标注（格式：标题2） 1](#_Toc293948322)

[第二章 高级操作（标题1，标号） 1](#_Toc293948323)

[2.1 图格式及引用 1](#_Toc293948324)

[2.2 公式格式及引用 1](#_Toc293948325)

[2.3 表格式及应用 1](#_Toc293948326)

[2.4 题注中的中文序号替换 1](#_Toc293948327)

[参考文献（不带序号标题1） 1](#_Toc293948328)

[致谢 1](#_Toc293948329)

[附录 1](#_Toc293948330)

[作者简介 1](#_Toc293948331)

图目录

[图 二‑1音圈电机驱动二维运动平台系统 1](#_Toc293948403)

[图 二‑2 四分之一磁路 1](#_Toc293948404)

[图 二‑3 等效电路图 1](#_Toc293948405)

[图 二‑4 替换图 1](#_Toc293948406)

表目录

[表 二‑1音圈电机常见技术参数 1](#_Toc293948396)

# 第一章 绪论

## 课题研究意义

随着计算机和移动设备的普及，互联网发展迅速，尤其是移动互联网。软件数量的不断增长、规模不断扩大、开发过程不规范等造成软件不稳定、不安全、容易崩溃等问题。因此，软件测试在软件周期中越来越重要，测试环节的比重甚至超过了开发。即便这样，软件还是不可避免存在bug。其中有一些bug是致命的，例如，网站系统中，如果在用户输入数据的限制不规范，很可能会存在SQL注入漏洞。对于这类影响严重的bug，我们称之为漏洞[《0day安全：软件漏洞分析技术》]。

当前的软件安全和网络安全的形势非常严峻。根据CNCERT 《2013年中国互联网网络安全报告》，越来越多的黑客利用软件漏洞进行攻击，比如斯诺登棱镜门、OpenSSL心脏流血和iCloud泄密风波等。2013年CNCERT/CC捕获及通过厂商交换获得的移动互联网恶意程序样本数量为702861个，相比2012年大幅增长331.3%。与2012年相比，2013年CNVD收录的漏洞总数增长15.1%，其中高危漏洞和低危漏洞分别增长6.8%和12.2%，低危漏洞大幅增长93.5%。[参考《2013年中国互联网网络安全报告》]。

为了防止不法人员利用漏洞进行攻击，安全研究人员开始使用漏洞挖掘技术进行漏洞挖掘，以保障软件和网络安全。传统的漏洞挖掘技术如手工测试、二进制对比技术、静态分析等，测试周期长，有的还需要源码，已经无法满足现在的需求，因此，Fuzz testing成为了大部分安全研究人员的选择。其核心思想是将不正常的数据输入到一个程序中，如果程序异常或崩溃则发现漏洞，Fuzz testing常常用于检测软件或计算机系统的安全漏洞。目前公布的安全漏洞中，许多是由Fuzz testing技术发现的。

随着Fuzz testing技术的广泛使用，出现了很多Fuzz testing工具，如SPIKE、Peach、General Purpose Fuzzer、Autodafe等。其中Peach因其开源、可扩展性强并且跨平台等优点而得到广泛使用。然而Peach也存在一些局限性，例如测试用例数量庞大导致测试周期长等，本文就对Peach 的Fuzz策略进行优化研究与实现，提出Combination策略和抽样方案，有效地提高了测试效率。在漏洞挖掘领域是非常有意义的。

## 课题来源

通过对Fuzz testing技术的学习，进而使用Peach挖掘漏洞，虽然Peach具有很多优点，以至于得到人们的普遍认可。但是在实际使用中Peach的Random策略需要的无限的测试时间，大大降低了测试效率。

通过对Peach Fuzz策略的研究以及源码的学习，发现可以添加新的Fuzz策略来消除Random策略中重复的测试用例，并且测试用例总数是确定的也即测试时间是有限的。对于复杂的测试情形，测试用例总数会非常大，对此提出了抽样的方案以减少测试用例总数。

本课题基于Peach源码进行修改，提出了Combination策略，以及抽样方案来提高测试效率。

## 国内外研究现状

Fuzz testing技术源于软件测试中的黑盒测试技术。关于Fuzz testing技术，最杰出的的研究成果主要有以下几个方面：

1990年Miller发现，通过简单的Fuzz testing可以使运行于UNIX系统上的多于25％的程序崩溃；2002年Aitel通过自己设计实现的Fuzz testing工具SPIKE成功地发现了多个未知漏洞；2008年Godefroid等人利用Fuzz testing工具SAGE发现大型Windows应用程序中二十多个未知漏洞。

Peach作为比较成熟的Fuzz testing的开源项目,自动化程度高，无须在逆向工程过程中大量的人工参与。因此，Peach在许多领域受到欢迎，许多公司和组织用其来提高软件质量，漏洞分析者使用它发现和报告漏洞，黑客使用它发现并秘密利用漏洞。

著名的浏览器公司Firefox长期使用Peach进行测试，CPU生产商Intel也开始使用Peach，还有专注于金融行业的Square公司，Peach因其扩展性强、跨平台的优点正在被应用到各行各业。

目前，国内对Fuzz testing技术有了较深的研究和应用。特别地，很多安全实验室使用Peach进行漏洞挖掘，一部分开始修改Peach源码满足特定的需求。

## 研究目的和内容

本课题的目的是基于Peach的Fuzz策略进行研究和优化，以提高测试效率。通过修改Peach源码来实现新的设计，在实际测试中检验是否提高了Peach的测试效率。

根据以上的研究目的，本课题主要研究一下几个方面的内容：

1. 研究Peach现有的Fuzz策略，并在此基础上提出Combination策略。
2. 研究Peach中影响测试时间的因素
3. 研究Peach中采取抽样方案，并讨论其影响
4. 利用修改后的Peach在实际环境中进行测试，检验是否提高了测试效率

## 论文结构安排

本论文共分为七章。

第一章介绍了课题的研究意义、课题的来源、国内外研究现状、研究目的和内容以及论文结构安排。

第二章介绍了Fuzz testing技术，

第三章介绍了Peach

第四章介绍了Combination策略

第五章介绍了抽样方案

第六章

第七章

# 第二章 Fuzz testing技术

## 漏洞挖掘技术概述

在计算机安全领域，漏洞是指允许攻击者减少系统信息保护能力的 缺陷[http://en.wikipedia.org/wiki/Vulnerability\_(computing)#cite\_note-1]。也就是说攻击者可以利用它来获得非法的权限，从而进行一些破坏行为。漏洞挖掘技术就是测试人员用来寻找漏洞的一种技术。

根据分析对象的不同，漏洞挖掘技术可以分为基于源码的漏洞挖掘技术和基于目标代码的漏洞挖掘技术两大类[[3]徐良华，孙玉龙，高丰等.基于逆向工程的软件漏洞挖掘技术[J].微计算机信息，2006，22（8-3）]。

从逆向分析的软件测试角度，又可分为白盒测试、黑盒测试和灰盒测试三类。

白盒测试[http://en.wikipedia.org/wiki/White-box\_testing]是一种测试内部结构或者工作原理而不是它的功能软件测试方法。在白盒测试中，经常根据系统内部的观点和编程技巧来设计测试用例。测试人员选择输入来测试不用的编码路径并定义合适的输出。这类似于测试电路中的节点，如在线测试。

白盒测试可以应用于单元测试、集成测试和系统测试。虽然传统的测试人员倾向于把白盒测试理解为单元测试，但是现在它更多地应用于集成测试和系统测试中。它可以测试单元内部的路径，集成测试时不同单元之间的路径以及系统测试时不同子系统之间的路径。即使这种测试设计能发现很多错误，但是它有可能错失规格中未实现的部分。

黑盒测试[http://en.wikipedia.org/wiki/Black-box\_testing]是一种检验程序功能而不用了解它的内部结构和工作原理的软件测试方法。这种测试方法可以应用于软件测试的各个阶段：单元测试、集成测试、系统测试和验收测试。它通常包括大多数更高水平的测试，但也可以控制单元测试。

灰盒测试[http://en.wikipedia.org/wiki/Gray\_box\_testing]是白盒测试和黑盒测试的组合。它的目的是寻找由于不合理的结构或者不合理的使用导致的缺陷。

黑盒测试人员不知道程序的内部结构，而白盒测试人员知道程序的内部结构。灰盒测试人员知道部分程序的内部结构，其中包括内部文档和算法。灰盒测试人员需要描述程序的高级别的和详细的文档，他们通过这些文档来定义测试用例。

## Fuzz testing的概念

Fuzz testing是一种软件测试技术，通过自动或半自动的方法产生非法的、不期待的或随机的数据作为程序的输入。同时监测程序是否出现崩溃、断言错误或潜在的内存泄露等异常现象。Fuzz testing通常被应用于软件安全领域和计算机系统[http://en.wikipedia.org/wiki/Fuzz\_testing]。该技术可以对命令行程序、GUI程序、网络通信、嵌入式设备等进行测试。

## Fuzz testing发展历程

* 1988年，Barton P.Miller在一个暴雨雷电交叉的晚上，使用Unix系统中的一个常用工具，结果发现噪音使这个程序崩溃。Barton P.Miller发现不正常的输入很可能会导致程序崩溃，他尝试了一些意见后，将这种技术命名为Fuzz。
* 1990年，Barton P.Miller、Lars Fredriksen和Bryan So三人在不同版本的UNIX操作系统上对内核和常用工具进行测试，结果发现大概25%~33%的程序会崩溃。[An Empirical Study of the Reliability of UNIX Utilities]
* 1995年，Barton P.Miller、David Kosk、Cjin Pheow Lee、Vivekananda Maganty、Ravi Murthy、Ajitkumar Natarajan和Jeff Steidl对UNIX基础的工具、X-Window 应用程序和服务器以及网络服务进行了Fuzz测试。他们的测试工具很大程度上完全自动化，而且测试方法也很简单。测试结果显示，有超过40%的基础程序和超过25%的X-Window应用程序出现崩溃或停止响应，但是没有发现网络服务和X-Window服务器存在问题[Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services]。
* 1999年，芬兰的奥卢大学通过分析网络协议规范开发出了网络协议测试套件，2002年微软向该项目提供资金支持，2003年该项目团队成立Codenomicon公司，致力于商业Fuzz testing测试套件的设计和研发[Fuzzing Brute Force Vulnerability Discovery]。
* 2000年，Justin E.Forrester和Barton P.Miller对Windows NT操作系统上的应用程序进行Fuzz测试，他们采用两种随机输入方法，一种方法是键盘按键和鼠标点击事件的随机组合序列，另一种是随机的Win32消息。测试结果显示，超过21%的程序崩溃，超过24%的程序停止响应[An Empirical Study of the Robustness of Windows NT Applications Using Random Testing]。
* 2002年，Dave Aitel在GNU框架下发布了Fuzz testing工具SPIKE。
* 2006年，Barton P. Miller、Gregory Cooksey和Fredrick Moore对Mac OS X操作系统进行了Fuzz测试。他们对Mac OS X操作系统上的135个命令行工具和30个图形界面工具进行测试，结果发现只有7%的命令行工具崩溃，相对于之前的测试这个结果已经非常低。但是大部分图形界面的工具都会出现崩溃，这个结果比之前的更差[An Empirical Study of the Robustness of MacOS Applications Using Random Testing]。

## Fuzz testing的过程

Fuzz testing对正常的输入进行变异，然后将变异后的数据作为被测对象的输入，如此不断重复，在此同时，监视器一直监视被测对象，并将监测信息记录下来。通过对监视器记录信息的分析，即可知被测对象是否存在漏洞。



Figure 1Fuzz testing的过程

## 变异器的类型

变异器是Fuzz testing技术的核心，其作用就是对输入进行改变，然后传输给被测对象。目前主要有两种变异器，一种是基于突变的，另外一种是基于格式生成的[Analysis of Mutation and Generation-Based Fuzzing]。

### 基于突变

这种变异器先收集正常的输入，例如一个格式正确的XML文件，或者一个符合规范的网络通信包，然后对这些正常的文件进行改变，例如使用长度很小的和长度很大的字符串替换某个字段，将某个整数用非常小的和非常大的整数进行替换，删除一个或者多个字段等方法。

基于突变的优点是测试人员不需要很深的专业背景知识，因为基于突变只需要正常的输入文件和一些预定义的突变方法。缺点是大部分测试用例被识别为无效的输入，导致代码覆盖率低，测试效率较低[王清\_Fuzz技术与软件安全性测试]。

### 基于格式生成

这种变异器根据输入文件的格式说明或者网络协议的RFC文档构造测试用例。产生大量测试用例的关键是是每个测试用例区别于合法的数据，但是测试用例于合法数据的区别不能太多，因为这样的话，被测对象很容易识别出该测试用例非法而抛弃。

基于格式生成的缺点是测试某个对象之前需要进行深入的学习，以掌握输入格式的规范，然后定义生成测试用例的规则。优点是通常人工生成的测试用例与正常的输入很相似，这使得生成的测试用例能够被接受，被测对象中更多的路径被测试到。直观地看，需要很深的背景知识的基于格式生成测试用例的方法产生的测试用例质量更高。[王清\_Fuzz技术与软件安全性测试]

## Fuzz testing工具

从1988年Fuzz testing被提出来，到现在26年中产生了许多Fuzz testing工具，这些工具从刚开始的测试文件、网络通信到现在已经扩展到了移动设备、无线通信等许多方面，随着Fuzz testing研究的不断深入，Fuzz testing工具也层出不穷，一个好的Fuzz testing工具应该抽象并减少乏味的工作。例如，在网络协议测试的第一阶段要对协议建模，如果是单纯的人工建模，需要理解协议的背景知识，而且工作量很大。一些工具就提供了捕捉报文，自动转换成一定格式，再由人工修改确认，这样大大减少了工作量，提高了效率。自动计算报文长度也是必不可少的功能，因为很多场景用到了TLV编码格式。Fuzz testing工具还应该包含计算CRC和校验和的功能，CRC通常在文件和网络协议中验证数据的正确性。

大部分Fuzz testing工具都会提供方法生成伪随机数据。然而其中一些工具更进一步提供了探索式攻击列表，这些列表中很多是已经发现导致程序崩溃的数据，例如格式化字符串（%n%n%n%n），文件路径（../../../../）等常见的攻击方法。使用这些常见的测试用例比毫无目的的随机更加有效[Fuzzing Brute Force Vulnerability Discovery]。

下面简要介绍几个常用的Fuzz testing工具。

### Fuzz

第一个Fuzz testing工具，基于Unix操作系统，可以产生随机字符串，通过选项可以控制产生不同类型的测试用例。字符串包括可打印字符和控制字符，通过选项可以只输出可打印字符、只输出控制字符或者两种字符混合。用户还可以定义不同测试用例之间的时间间隔，这样可以方便定位出现崩溃或停止响应的位置。通过选项可以指定随机种子给随机数生成器，来重现测试用例。同时可以将测试信息保存到文件，并可以控制输出文件的大小防止文件过大。

随后几年，又增加了使用shell脚本进行自动化测试，如果发现崩溃或者停止响应，会将其记录到日志文件中，随后再人工确认每一个问题。

之后，该程序扩展到了windows NT操作系统，通过模拟键盘和鼠标等外设的操作测试GUI程序，还可以随机产生Win32消息来测试程序。最后该程序扩展到了Mac OS X操作系统。

该程序功能简单，容易使用，可以用来测试命令行程序、GUI程序以及网络协议，缺点是自动化不完善，需要很多的人工参与。

### antiparser

使用Python语言开发，具有良好的跨平台性。该工具提供了Fuzz test和缺陷注入的API，可用来测试网络协议和文件。用户只需要编写模型，antiparser具有很多种方法产生随机数据来触发软件中的bug或者安全漏洞。[http://antiparser.sourceforge.net/]

但是antipaser有一些限制，代码的可重用性低，导致在antiparser上开发新的功能工作量较大。它不支持已有的漏洞库，不能计算TLV格式的长度，也不能计算CRC和校验和。而且它的文档很少，自2005年发布以后很久没有更新了。结合以上分析的特点，antiparser适用于简单的测试，不能满足复杂测试的需要。

### Dfuz

Dfuz由Diego Baucge使用C开发的，该项目的维护更新很快，已经被用来挖掘多个公司的漏洞。该项目是开源的，但是有很严格的拷贝、修改限制，用户很难根据自己的需求随意修改源代码。该项目功能不是很强大，但是简单容易上手，而且适合初学者学习，用户可以学习其源码，更深刻地了解Fuzz testing技术。[https://code.google.com/p/dfuz/]

### SPIKE

SPIKE由Dave Aitel使用C开发，得到广泛使用和认可。SPIKE是开源的项目并且基于GPL，为网络协议的测试人员提供了快速和高效的API。大部分网络协议的格式很相似，而且很多在SPIKE中都已经支持。该项目能够很快重复一个未知的复杂的网络协议，而且它内置了很多已知的漏洞，对网络协议测试的丰富支持使得测试人员很容易对网络协议进行测试。

SPIKE支持自动计算TLV格式的长度字段以及CRC和校验和，对二进制数据也有很丰富的方法支持测试，与其它基于GPL的开源项目可以很好的结合使用。由于该项目在开源社区很受欢迎，文档和示例比较丰富，通过修改部分源码能够很快满足用户的特殊需求。

### Autodafe

Autodafe是由Martin Vuagnoux使用C开发，能够对网络协议和文件格式进行Fuzz testing。该项目基于GNU，并且有详细的安装和使用文档。该项目提出了新的技术，定义用户输入的字符串或者其他值为标记者，定义debugger列举出的每一个动态调用的方法为跟踪者，该技术对每一个标记者赋予不同的权重，权重越高，测试的越多，还可以抛弃一些权重很低的标记者，这样可以减少很多测试用例。权重的大小是根据跟踪者的次数决定的，跟踪者遇到标记者的次数越多，该标记者的权重越大。这个新技术很大程度上提高了Fuzz testing的效率，使得Autodafe在挖掘缓冲区溢出漏洞处于领先地位，使用该工具已经发现了多达865个软件的缓冲区溢出漏洞。[http://autodafe.sourceforge.net/tutorial/]

### General Purpose Fuzzer

General Purpose Fuzzer是由Jared DeMott使用python开发的，该项目基于GPL，版本维护非常快。不同于其它的Fuzz testing工具，它可以产生无限多的测试用例，最大的优点就是只需要python的解释器，不需要进行任何的编译和安装，功能简单，用户可以直接运行。缺点就是测试时间无法控制，无法满足复杂需求。[http://www.gnucitizen.org/blog/general-purpose-fuzzer\_py/]

### Sulley

Sulley是由Pedram Amini和Aaron Portnoy使用python开发的，该项目是开源的，并且在社区非常活跃。Sulley包含了很多扩展组件，功能很完善。该工具的目标不仅是简化数据产生的方式而且简化了数据传输的方式以及对被测对象的监视。

之前提到的大部分Fuzz testing工具都仅仅专注于任何生成数据，Sulley不仅具有大量的生成数据，而且包含其它特色，比如，观察网络信息并系统地记录详细的日志，它使用多种方法对被测对象进行监测，可以并行测试从而大大加速的测试速度。Sulley最大的特色是对网络协议测试的支持很强大，它对网络协议的测试很深入，可以对网络协议的各个过程进行测试，这是很多Fuzz testing工具无法做到的。虽然Sulley的功能很强大，但是它用户界面很友好，甚至在复杂的情形下使用起来也比较容易。[http://www.fuzzing.org/wp-content/SulleyManual.pdf]

### Peach

Peach是使用python开发的跨平台的开源项目，v2版本的Peach都是使用python开发的，v3版本的Peach转由C#进行开发，该项目是跨平台的，支持Windows、Linux和Max OS X操作系。Peach具有非常好的架构设计，所以具有良好的可移植性和扩展性。它的功能非常完善，而且进行了很多创新，添加了很多新的功能，例如，支持对嵌入式设备进行测试，同时支持本地监视和远程监视，智能地生成测试用例，提供不同的测试方式等。

相对其它Fuzz testing工具，Peach具有丰富的文档，而且Peach社区非常活跃，不仅可以在邮件列表中讨论问题，而且有官方论坛进行讨论交流。而且Peach的商业版取得了很大成功，已经应用在Firefox、Square和Intel公司进行漏洞检测。很多国内外的安全研究人员都使用Peach进行测试，虽然功能很强大，但是Peach的测试用例太多，测试效率较低，本文接下来就对Peach进行介绍，提出Combination策略和抽样方案，以及实验验证。

# 第三章 Peach Fuzz testing框架

## Peach概述

Peach开始是使用python开发的跨平台的功能强大且扩展性强的Fuzz testing平台，支持基于格式生成和基于突变的两种变异器，之后v2版本都是使用python开发，v3版本开始使用C#开发。Peach项目的开源社区非常活跃，目前在国内外有很多安全研究人员在使用，Peach已经被安全咨询公司、产品测试人员和商业质量保证团队用来挖掘漏洞[http://peachfuzzer.com/]。

Peach具有以下特点：

* 智能测试：Peach支持基于格式生成的变异和基于突变的变异，支持有限状态机测试，更高级的变异方法。
* 黑盒测试：用户不需要登录到目标查看是否发生异常，Peach通过使用正常的输入能够检测目标是否正常。
* 扩展性：Peach具有良好的架构，围绕核心Peach.Core，用户很容易就可以创建自己的接口。
* 平衡性：Peach自身可以控制自己的Fuzz进程以到达平衡的状态。
* 用户界面：Peach具有良好的用户操作界面，丰富的配置文件，报表模板等，使得操作简便，用户工作量减少。
* 回报率：Peach的测试是自动化的，无需大量的人工参与，Peach可以提高代码覆盖并且对软件产品进行深入的测试。
* 测试范围：Peach不仅支持对文件和网络协议进行测试，还支持对GUI程序、内核、嵌入式设备、Android平台的测试。

## Peach工作原理

如图三-1所示，用户首先编写Pit文件，在该文件中指定文件格式，选择变异策略，选择监视器类型。然后通过Peach命令行使用该Pit文件进行测试，执行测试之后控制器控制测试引擎读入Pit文件，测试引擎是Peach的核心部分，它控制对哪个字段进行变异，以及如何变异。测试引擎将Pit文件和变异信息传递给变异器，变异器根据变异信息选择基于格式生成变异还是基于突变变异。之后变异器将变异之后的文件传递给被测对象，被测对象处理完之后回复给测试引擎，测试引擎再将测试信息返回给控制器，控制器可以控制测试引擎的执行、暂停等状态。在测试的同时监视器每隔一段时间就对被测对象进行正常通信，查看被测对象是否正常，并将此信息反馈给控制器，如果被测对象出现异常，控制器可以根据用户需求暂停测试或重启设备继续测试。Peach具有强大的漏洞管理功能，凡是在测试用出现异常时的状态都会被保存下来，供以后重现、分析使用。

图 三‑1Peach工作原理

## Pit文件

Peach中需要用户大量参与的就是Pit文件的编写，该文件可以控制输入文件和网络通信的格式，控制使用哪个变异策略，控制使用哪种监视器，控制被测对象出现异常时的行为等。用户编写完该文件交给Peach执行就可以了，只需等待Peach测试完成之后分析测试结果。

Pit文件主要分为模型、变异策略、变异方法、代理、I/O模块、日志等模块，图三-2展示了Pit文件的主要结构。[http://old.peachfuzzer.com/Introduction.html]。



图三-2 Pit文件的主要组成

### 模型

Pit文件的模型包括数据模型和状态模型。

#### 数据模型

数据模型使用DataModel表示，Pit文件至少包含一个DataModel，该模型描述了数据信息，这些信息包括类型信息，关系信息（大小，数量，偏移），还有其它影响变异的信息。

对于文件，DataModel就是按照文件正确的格式进行编写，对于网络协议，DataModel就是按照网络协议的RFC规范进行编写，所以说该模型的建立需要了解测试环境相关的知识，这样的测试效果较好。在很多测试场景中需要整数、字符串、计算校验和。因此DataModel内部还包括以下元素：

* DataModel：DataModel内部可以嵌套DataModel。
* Blob：Blob只能是DataModel和Block的子元素，主要用来表示缺乏类型定义或者格式的字段。
* Block：Block只能是DataModel和Block的子元素，主要用来将Number和String组合成一个逻辑结构。
* Choice：Choice只能是DataModel和Block的子元素，它的子元素都是合法的，但是只能选择一个。很想变成语言中的switch语句。
* Flag：Flag表示具体的位字段，该元素常用语网络协议中，例如TCP协议的头部标志位就使用位作为单位。
* Number：Number可以是DataModel、Block和Choice的子元素。Number常用来表示长度字段。在网络协议测试中，需要注意大小端之分。
* Padding：Padding用来在DataModel和Block元素中补齐。
* String：String只能是DataModel和Block的子元素，用来表示字符串。

DataModel中还包含计算某个字段的长度，计算校验和转换编码等功能。

#### 状态模型

状态模型使用StateModel表示，StateModel使用状态机的方式控制测试行为，这在网络协议测试中是非常重要的。StateModel中定义了如何发送数据给目标和如何从目标接收数据，StateModel从简单到复杂之间变化，用户最好将StateModel定义简单，然后根据需求不断扩展。

StateModel中包含两个元素State和Action。

##### State

表示一个测试状态，简单情形下可能只需要一个State，复杂情形下需要许多State来表示不同的状态，State内部是由Action组成的。每一个StateModel中都必须指定一个初始的State，不同的State之间可以根据测试情况进行跳转，这就能满足复杂网络协议的测试。

##### Action

Action元素可以产生多种操作，它是发送命令给I/O模块的主要方式。Action可以输出数据到I/O模块，从I/O模块读取数据，也可以使I/O模块与目标建立连接。Action除了与I/O模块交互，还有改变当前状态，在DataModel之间转移数据，调用代理方法等动作。

下面列举几种不同的Action：

* start：启动I/O模块，这是开始后默认执行的动作，一般不用显示调用。
* stop：暂停I/O模块，这是结束前默认执行的动作，一般不用显示调用。
* open/connect：对于文件，是打开文件，对于网络通信则是建立连接。这个动作是默认执行的，只有当特殊需求的时候，用户才需要显示使用该动作。
* close：关闭文件或断开连接，
* accept：接收外来连接。不是所有类型的I/O模块都支持该动作，它会阻塞直到外部请求连接的到来。
* input：从I/O模块接收或读取数据，需要DataModel定义特定的解析格式。
* output：发送数据到I/O模块，需要指定DataModel。
* call：调用I/O模块定义的方法，并不是所有的I/O模块都支持该动作。
* setProperty：设置属性值，部分I/O模块支持该动作。
* getProperty：获得属性值，部分I/O模块支持该动作。
* changeState：从当前State跳转到指定状态，只有当满足一定条件的时候才会跳转。

### 补丁

补丁在Peach中表示为Fixup，它是DataModel中的重要应用。Fixup有很多功能，比如动态地计算文件的CRC，计算TCP报文的校验和等。Fixup具有很好的扩展性，用户可以根据自己的需要定义自己的Fixup。

下面简单介绍下Peach中常用的Fixup。

#### 实用补丁

* 拷贝补丁：将指定字段的补丁拷贝到当前字段。
* 顺序递增补丁：每个测试用例，该值都会增加1。
* 随机补丁：每个测试用例，该字段都会随机生成一个整数。对于某些网络协议经常使用该补丁。

#### 校验和补丁

包括CRC补丁，Ethernet校验和补丁，ICMP校验和补丁，TCP和UDP校验和补丁，以及表达式补丁。

#### 哈希补丁

包括MD5哈希补丁，SHA系列补丁。

### 代理

Pit文件中的代理使用Agent表示，它是Peach中的一个特殊进程，它可以本地运行也可以运行在远程。每一个进程都可以有若干个监视器，这些监视器可以连接到调试器，观察内存使用情况，检测是否出现异常等。

Agent是跨平台的，可以运行在Windows、Linux和OS X上，下面简单介绍下常用的监视器。

#### Windows监视器

* Windows 调试监视器：负责监视Windows下的进程、服务、内核的调试信息。
* 清理注册表监视器：负责清除注册表中的数据。
* Windows服务监视器：负责监视Windows服务。

#### OS X监视器

* 崩溃监视器：负责监视Mac OS X进程信息。
* 崩溃报表监视器：负责报告Mac OS X系统崩溃信息。

#### Linux监视器

* Linux崩溃监视器：通过在内核注册脚本捕捉进程崩溃信息。

#### 跨平台监视器

* 内存监视器：负责监视内存的使用情况，当超过设置的上限时会抛出异常，在检查非崩溃内存使用情况非常有用。
* Pcap监视器：负责捕捉网络通信报文，如果出现异常，那么Pcap监视器就会将当时的通信过程保存成pcap文件，该文件可以使用Wireshark和tcpdump打开。
* Ping监视器：常用的监视器，可以检测与目标之间的连接是否正常。
* Vmware监视器：该监视器可以控制虚拟机开机、关机、返回到某个快照。当目标崩溃时，可以使虚拟机返回到特定的快照。

### 变异策略

DataModel中定义了需要测试的字段，每种类型的字段都有对应的变异方法，每个变异方法中都有几个到很多个值。变异策略就是决定测试的顺序，即哪些字段先变异，先取这些字段中的哪个变异方法，先取这些变异方法中的那个值。

传统上Peach支持自上而下的顺序变异，这种策略可以保证每个字段都会被测试到，但是对这种策略不适用于能够产生数以亿计测试用例的大型系统。而且对于不同的情形相应的改变变异的顺序可能会产生更好的效果，因此Peach的架构设计使得用户很容易定制自己的变异策略，除此之外Peach默认支持三种变异策略。

#### Sequential

该策略对DataModel中的每个字段逐一测试，即每次只对一个字段进行测试，比如当前字段可以产生10个测试用例，那么接下来的10个测试用例都是对该字段变异得到，之后再对下一个字段进行变异。这样一直到最后一个字段测试完成，所以该策略的执行过程是固定的，产生的测试用例的总数也就是确定的。该策略的随机种子一直都是31337，用户不能修改。

#### Random

该策略生成的测试用例数是无限的，一旦测试开始执行就永远进行测试，除非人为终止或目标出现异常不能重启才会停止。每个测试用例都随机选择1到N个字段进行变异，其中N是由用户指定的，每个字段随机选取一个对应的变异方法，该变异方法再随机选取一个取值。Peach通过随机种子进行随机的选择，通过传入相同的随机种子就可以重复某个测试用例，这样如果目标出现异常，就可以使用特定的测试用例重现。

该策略适用于DataModel比较复杂的情况，也可以在顺序测试完成之后进行测试。

#### RandomDeterministic

如果不指定变异策略的话，那么就会默认使用该策略。该策略与Sequential策略相似，只不过测试用例的顺序是随机的。生成的测试用例的数量是确定的，并且与Sequential策略生成的测试用例的数量是相同的。通过传入相同的随机种子就可以重复某个测试用例。

### 变异方法

选定一个字段，并不是所有的变异方法都适应它，每种类型的字段都对应一些变异方法，除此之外的变异方法是不适用的。用户可以在Pit文件中指定使用哪些变异方法，指定不使用哪些变异方法，用户还可以定义自己的变异方法。

下面对Peach中默认的变异方法进行简要介绍。

#### ArrayNumericalEdgeCasesMutator

对数组进行变异的方法，将数组的大小改为非常大。用户可以指定生成的测试用例数，默认生成50个测试用例。

#### ArrayRandomizeOrderMutator

对数组进行变异的方法，随机改变数组中的顺序。用户可以指定生成的测试用例数，默认生成50个测试用例。

#### ArrayReverseOrderMutator

对数组进行变异的方法，翻转数组中的顺序，只能生成一个测试用例。

#### ArrayVarianceMutator

对数组进行变异的方法，将数组的大小在[-N, +N]的范围内进行改变。用户可以指定生成的测试用例数，默认生成50个测试用例。

#### BlobBitFlipperMutator

对Blob类型的字段进行变异，翻转二进制中的某些位。用户可以指定生成的测试用例数，默认生成50个测试用例。

#### BlobDWORDSliderMutator

对Blob类型的字段进行变异，生成的测试用例数为当前测试字段的字节数。

#### BlobMutator

对Blob类型的字段进行变异，可以扩大字段的二进制位数，或者减少字段的二进制位数。生成的测试用例数为改变二进制位数的方法数目。

#### DataElementDuplicateMutator

适用于大部分类型的字段，将当前字段拷贝1遍至50遍，生成50个测试用例。

#### DataElementRemoveMutator

适用于大部分类型的字段，删除当前字段，生成1个测试用例。

#### DataElementSwapNearNodesMutator

适用于大部分类型的字段，需要选择两个字段，然后交换这两个字段的位置，生成1个测试用例。

#### FiniteRandomNumbersMutator

对Number类型的字段进行变异，随机生成一个数替换当前字段。用户可以指定生成的测试用例数，默认生成5000个测试用例。

#### NumericalEdgeCaseMutator

对Number、String、Flag类型的字段进行变异，使用边界值替换。用户可以指定生成的测试用例数，默认生成50个测试用例。

#### NumericalVarianceMutator

对Number类型的字段进行变异，默认值加上[-N,+N]区间内的一个值。用户可以指定生成的测试用例数，默认生成50个测试用例。

#### SizedDataNumericalEdgeCasesMutator

适用于具有大小的data字段，用户可以指定生成的测试用例数，默认生成50个测试用例。

#### SizedDataVaranceMutator

适用于具有大小的data字段，默认的大小加上[-N,+N]区间内的一个值。用户可以指定生成的测试用例数，默认生成50个测试用例。

#### SizedNumericalEdgeCasesMutator

适用于具有数值的字段，使用边界值替换。用户可以指定生成的测试用例数，默认生成50个测试用例。

#### SizedVaranceMutator

适用于具有数值的字段，默认的数值加上[-N,+N]区间内的一个值。用户可以指定生成的测试用例数，默认生成50个测试用例。

#### StringCaseMutator

对String类型的字段进行变异，改变字符串的大小写，生成3个测试用例。

#### StringMutator

对String类型的字段进行变异，使用常用的字符串进行替换，该方法中存储了2379个常用的字符串用于替换，所以可生成2379个测试用例。

#### UnicodeBadUtf8Mutator

对String类型的字段进行变异，使用UTF-8格式的字符串进行替换，该方法中存储了150个UTF-8格式的字符串用于替换，所以可生成150个测试用例。

#### UnicodeBomMutator

对String类型的字段进行变异，使用UTF BOM格式的字符串进行替换，该方法中存储了1413个UTF BOM格式的字符串用于替换，所以可生成1413个测试用例。

#### UnicodeStringsMutator

对String类型的字段进行变异，使用UTF格式的字符串进行替换，该方法中存储了202个UTF格式的字符串用于替换，所以可生成202个测试用例。

#### UnicodeUtf8ThreeCharMutator

对String类型的字段进行变异，使用长的UTF-8三字节的字符串进行替换，该方法中存储了301个UTF-8三字节的字符串用于替换，所以可生成301个测试用例。

### I/O模块

I/O模块使用Publisher表示，它负责与目标建立连接，发送数据到目标，从目标接收数据。Publisher支持文件、流和调用操作。

Publisher与Action是相关的，不同的Publisher支持的Action不同，例如，文件传输支持input和output动作，但是不支持accept和connect。COM传输支持调用call动作，但是不支持input、output和accept。

Pit文件中至少要指定一个Publisher，一般情况下也只有一个。对于特殊情况下，可以指定多个，但是需要仔细考虑之后再定义，这种情况下需要为每个action指定其使用的Publisher，如果不指定的话，默认使用第一个Publisher。

Peach中的Publisher种类很丰富，主要分为文件、网络通信、控制台和COM。

#### 文件

Publisher可以指定两种具体的文件，一种是单一文件，只保存当前的测试用例信息，即新的测试用例信息会覆盖掉旧的测试用例信息；另一种是多文件，每个测试用例都会保存成独立的文件，不同的测试用例之间不会影响，测试结束之后可以很方便地查看每个测试用例的信息。

#### 网络通信

Publisher对网络协议的支持非常丰富，不仅支持基于TCP和UDP 的应用层协议，还支持基于IPv4和IPv6的传输层协议，以及基于Ethernet的IP层协议。Publisher支持大部分网络协议的测试，还支持WebService的远程调用，而且Publisher支持TCP服务器。

#### 控制台

Publisher支持将测试信息和测试数据直接输出到控制台，方便用户查看测试进展。Publisher支持两种类型的控制台输出，一种是测试数据正常输出，另一种是输出测试数据的十六进制形式。

对于控制台简便易用，不用配置地址、端口等信息，但是它不适合接收数据，所以一般是用来测试、调试。

#### COM

COM是一种跨语言的二进制代码共享方法，它是基于Windows操作系统的，所以这种类型的Publisher只能运行在Windows环境。

使用COM可以很方便的对Windows环境下的链接库、执行程序等二进制模块和远程调用服务进行测试。

### 日志

Peach的日志功能非常强大，而且具有很好的扩展性，用户可以根据需要进行扩展。Peach默认提供一个单一日志文件，该文件保存在指定路径下，文件夹的名字是由run的名字和时间组成的。只有当发现异常时，才会将一部分测试日志信息保存到硬盘上，这样可以防止日志文件过大占用硬盘空间。

# 第四章 Combination策略

## 变异策略分析

下面详细介绍Peach中支持的三种变异策略。在介绍变异策略之前我们先讨论下一个变异操作的产生。为了简便起见我们只讨论具有一个DataModel和一个State的情形，如图四-1所示。

对DataModel进行变异，分为单字段变异和多字段组合变异。单字段变异首先选择一个需要变异的字段（如果在字段中设置属性mutable为false，就不对当前字段进行变异），然后根据该字段的类型选择其对应的变异方法，变异方法根据当前字段的不同产生的实际变异操作也不相同，按照一定顺序执行变异操作之后，一个测试用例就生成了。多字段组合变异与单字段变异相似，不同的是多字段组合变异是多个字段按照先后顺序进行变异，例如3字段组合变异会有3次实际的变异操作。

设Fi表示第i个需要变异的字段，设DataModel中共有n个需要变异的字段，则Fn表示最后一个字段。设M（Fi，j）表示Fi对应的变异方法中的第j个，设Fi对应的变异方法共有C（Fi）个，则M（Fi，C（Fi））表示Fi对应的最后一个变异方法。设V（M（Fi，j），k）表示M（Fi，j）中的第k个变异操作，设M（Fi，j）共有C（Fi，j）个变异操作，则V（M（Fi，j），C（Fi，j））表示M（Fi，j）的最后一个变异操作，则V（M（Fi，j），C（Fi，j））表示Fi的最后一个变异操作，则V（M（n，C（n）），C（n，C（n）））表示该DataModel的最后一个变异操作。使用V可以表示一个唯一的变异操作，对于单字段变异也就是一个测试用例，而多字段组合变异的一个测试用例就是多个V执行后生成的。



图四-1简单的Pit文件

### Sequential

对DataModel中的每个需要变异的字段逐一测试，即从F1开始测试，直到F1测试完才开始测试下一个字段直到最后一个字段测试完。该策略的执行过程是固定的，产生的测试用例的总数也就是确定的。该策略的随机种子一直都是31337，用户不能修改，这样可以保证每一个的测试过程产生的测试用例是相同的，目标出现异常后可以重复当时的测试用例。

图四-2描述了Sequential策略下第i个字段生成的所有测试用例，对Fi字段变异时，第1个测试用例为V（M（Fi，1），1），第2个测试用例为V（M（Fi，1）， 2），一直到最后一个测试用例V（M（Fi，C（Fi）， C（Fi，C（Fi））））执行完后即完成对该字段的测试。Sequential策略就按照这种方式依次对F1~Fn进行测试，当Fn测试完成之后就该策略就完成了所有测试。



图四-2 Fi生成的测试用例

Sequential策略的执行步骤：

1. **初始化变异序列**
2. 设当前DataModel中共有n个字段需要测试，使用iterations容器存储每个测试用例。设i = 1。
3. 如果i大于n，则跳转到第11步，否则继续。
4. 当前字段为Fi，设Fi对应的变异方法数为C（Fi）。设j=1。
5. 如果j大于C（Fi），则跳转到第10步，否则继续。
6. 当前变异方法为M（Fi， j），设该方法对应的变异操作共有为C（Fi，j）个。设k=1。
7. 如果k大于C（Fi，j），则跳转到第9步，否则继续。
8. 当前的测试用例为第i个字段，该字段对应的第j个变异方法，该变异方法的第k个变异操作。将V（i，j，k）存储到iterations中。
9. k++，跳转到第6步。
10. j++，跳转到第4步。
11. i++，跳转到第2步。
12. 初始化变异序列完成。
13. **执行测试**
14. 如果用户终止测试或目标异常无法重启，跳转到第6步，否则继续。
15. 设i=1，N=iterations.size()。
16. 如果i大于N，则跳转到第6步，否则继续。
17. 执行当前测试用例。
18. i++，跳转到第3步。
19. 测试结束。

Sequential策略是基本的变异策略，目前大部分Fuzz testing工具都使用这种方式进行测试，很多测试情形都是先使用Sequential策略进行测试。这种策略比较简单，但是测试模拟的情形也是简单的，与正常输入只有一个字段的区别。还有一些漏洞是跟上下文相关的，所以不同的变异顺序也可能效果不一样。因此除了按照字段顺序依次变异外还有打乱正常测试顺序的需求，RandomDeterministic策略。

### RandomDeterministic

如图四-3 和图四-4所示，RandomDeterministic策略与Sequential策略相似，生成的测试用例是一样的，只不过顺序不同。该策略是根据输入的随机种子进行随机变换顺序的，所以只要随机种子相同，测试用例就想听，因此该策略变异时会记录使用的随机种子，如果想重复某个测试用例只需传入该测试用例使用的随机种子即可。

图四-3 Sequential策略生成的测试用例 图四-4 RandomDeterministic策略生成的测试用例

RandomDeterministic策略的执行步骤：

1. **初始化变异序列**
2. 设当前DataModel中共有n个字段需要测试，使用iterations容器存储每个测试用例。设i = 1。
3. 如果i大于n，则跳转到第11步，否则继续。
4. 当前字段为Fi，设Fi对应的变异方法数为C（Fi）。设j=1。
5. 如果j大于C（Fi），则跳转到第10步，否则继续。
6. 当前变异方法为M（Fi， j），设该方法对应的变异操作共有为C（Fi，j）个。设k=1。
7. 如果k大于C（Fi，j），则跳转到第9步，否则继续。
8. 当前的测试用例为第i个字段，该字段对应的第j个变异方法，该变异方法的第k个变异操作。将V（i，j，k）存储到iterations中。
9. k++，跳转到第6步。
10. j++，跳转到第4步。
11. i++，跳转到第2步。
12. 初始化变异序列完成。
13. 随机改变iterations中的序列，打乱原来的顺序。
14. **执行测试**
15. 如果用户终止测试或目标异常无法重启，跳转到第6步，否则继续。
16. 设i=1，N=iterations.size()。
17. 如果i大于N，则跳转到第6步，否则继续。
18. 执行当前测试用例。
19. i++，跳转到第3步。
20. 测试结束。

目前的Fuzz testing工具支持这种策略的较少，该策略随机变换变异顺序，能够满足上下文相关的测试情形的需求。一般是使用Sequential策略测试一遍之后，根据情况有选择地使用该策略测试。

### Random

Random策略是多个字段组合测试，一个测试用例包括1~N（最大组合字段数）字段的变异。该策略生成的测试用例是无限的，测试进行中不会自动停止除非人为终止或目标出现异常导致无法重启。

Random策略的执行步骤：

1. 如果用户终止测试或目标异常无法重启，跳转到第9步，否则继续。
2. 使用随机种子S，随机生成一个整数i，则本次测试j =（i%N + 1）个字段。
3. 使用随机种子S，随机选择j个字段。设k =1。
4. 如果k大于j，则跳转到第8步，否则继续。
5. 使用随机种子S，从该字段对应的变异方法中随机选择一个变异方法M。
6. 使用随机种子S，从该变异方法的变异操作中随机选择一个变异操作V，并执行该变异操作。
7. K++，返回第4步。
8. （如图四-5所示）生成一个测试用例，返回第1步。
9. 测试结束。



图四-5 Random策略中的一个测试用例

目前的Fuzz testing工具支持这种策略的较少，该策略既多个字段组合测试，又随机选择字段，能够满足复杂苛刻的测试需求，缺点是测试用例数无限大，测试时间无限长，只能人为终止。

### 局限性分析

Sequential策略和RandomDeterministic策略适用于大部分测试场景，但这两中策略都是单字段测试，Random策略的出现弥补了这两种策略的不足，该策略随机选择多个字段进行测试，但是该策略生成的测试用例数是无限的，测试时间也是无限长的。

实际测试中常遇到这种测试场景，DataModel中的字段较少，Sequential策略和RandomDeterministic策略生成的测试用例较少，并且在短时间内能够完成测试。如果使用Random策略的话就会一直测试，不会停止，在这个过程中就会由很多重复的测试用例。对于这种场景Peach支持的三种策略都无法满足需求。

## Combination策略

由于当DataModel中的字段较少时，仅使用Sequential策略和RandomDeterministic策略测试覆盖范围较小，使用Random策略虽然覆盖范围较大，但是测试时间无限长而且存在重复测试用例，无法满足需求。因此，提出了Combination策略，该策略借鉴了Peach支持的以上三种策略，遍历所有组合数小于指定值的字段组合情况，然后既可以像Sequential策略顺序依次测试，又可以像RandomDeterministic策略打乱顺序进行测试，而且该策略生成的测试用例是确定的，测试时间是有限的，测试过程中不会产生重复的测试用例。由此，可以推论Combination策略很适合DataModel中字段较少的场景，是Sequential策略、RandomDeterministic策略和Random策略的很好的补充。

### Combination策略的设计

借鉴Sequential策略和RandomDeterministic策略，可以在Combination策略分为SequentialCombination策略和RandomCombination策略，SequentialCombination策略按照顺序依次进行测试，RandomCombination策略则打乱顺序进行测试。

将Combination策略的执行过程分为两部分，第一部分完成初始化变异序列，即遍历所有的合法的字段组合，并记录存储到容器iterations中，如果是SequentialCombination策略则第一部分完成，如果是RandomCombination策略则需要将iterations中的顺序打乱。第二部分按照iterations的顺序执行变异。

#### 辅助算法

设DataModel中共有n个字段，最大组合数为MaxGroup，最小组合数为MinGroup。定义两个方法以简化Combination策略的执行步骤。

##### 初始化组合字段

该方法的参数分别为组合字段的第一个字段Fi，组合数j。该方法分别对Fi，F（i+1），…，F（i+j-1）这些字段进行初始化，即将这些字段的当前的变异方法设置成对应的变异方法中的第一个，以及将该变异方法中的变异操作设置成第一个。

下面是该方法的伪代码描述：

初始化组合字段（i，j）

1. k = i
2. while (k++ < i + j)
3. Fk.CurrentMutator = M(Fk, 1)
4. Fk.CurrentMutator.CurrentValue = V(M(Fk, 1), 1)

##### 遍历组合字段

该方法的参数分别为组合字段的第一个字段Fi，组合数j。如图四-6所示，分别遍历F（i+j-1），…，F（i+1），F（i）。字段F（i+j-1）是最内层的循环，只要其它字段没有遍历到最后一个测试用例，就遍历F（i+j-1）字段，直到这些字段都遍历到最后一个测试用例为止。同时需要记录每个组合字段的测试用例的信息到iterations中。

初始化组合字段（i，j）

1. while (1)

// 将当前的测试用例存储到iterations中

1. list = {};
2. for (k = i; k < i + j; k++)
3. list.add(Fk.CurrentMutator.CurrentValue);
4. iterations.add(list);

// 判断组合字段是否到达最后一个测试用例

1. is\_finish = true
2. for (k = i; k < i + j; k++)
3. if (Fk.CurrentMutator != M(Fk, C(Fk)))
4. is\_finish = false
5. break
6. if (Fk.CurrentMutator.CurrentValue != V(M(Fk, C(Fk)), C(Fk, C(Fk))))
7. is\_finish = false
8. break;
9. if (is\_finish)

// 当前测试用例是该组合字段的最后一个测试用例

// 则该组合字段遍历完成，退出循环

1. break;

// 寻找下一个测试用例

1. if (! F(i+j-1).CurrentMutator.CurrentValue.IsLastValue())

// 如果F（i+j-1）当前的变异方法中当前的变异操作不是最后一个

// 则对F（i+j-1）当前变异方法的中的变异操作下移一个

1. F(i+j-1).CurrentMutator.CurrentValue.MoveNext();
2. else if (! F(i+j-1).CurrentMutator. IsLastMutator())

// 如果F（i+j-1）当前的变异方法不是最后一个

// 则对F（i+j-1）当前变异方法下移一个

F(i+j-1).CurrentMutator.MoveNext();

F(i+j-1).CurrentMutator.CurrentValue = F(i+j-1).CurrentMutator.FirstValue;

1. else

// F（i+j-1）当前的测试用例已经是最后一个

// 设置F（i+j-2）的下一个测试用例

// 如果F（i+j-2）当前的测试用例是最后一个，那么继续设置F（i+j-3）// 一直到Fi

1. k = i+j-2
2. for (; k >= i; k--)
3. if (!Fk.CurrentMutator.CurrentValue.IsLastValue())
4. Fk. CurrentMutator.CurrentValue.MoveNext()
5. break
6. else if (!Fk.CurrentMutator. IsLastMutator())
7. Fk. CurrentMutator. MoveNext()
8. break



图四-6 组合字段的测试用例。其中字段F（j）对应的数字1表示该字段的第一个变异，End表示该字段的最后一个变异。

##### 改进的随机算法

本部分参考《算法导论》

#### 执行步骤

1. **初始化变异序列**
2. 如果用户终止测试或目标异常无法重启，则跳转到第12步，否则继续。
3. 如果n小于MinGroup，跳转到第12步，否则继续。
4. 设i = MinGroup，如果MaxGroup大于n，则令MaxGroup=n。
5. 如果i大于MaxGroup，则跳转到第10步，否则继续。设j=1。
6. 如果j大于n-i+1，则跳转到第9步，否则继续。

// 从Fj开始，依次对Fj，F（j+1），F（j+2），F（j+i-1）进行变异。

1. 调用辅助算法：初始化组合字段（j，i）
2. 调用辅助算法：遍历组合字段（j，i）
3. j++，跳转到第5步。
4. i++， 跳转到第4步。
5. 如果是SequentialCombination策略则跳转到第12步，否则继续。
6. 随机打乱iterations中的顺序。
7. 测试结束。
8. **执行测试**
9. 如果用户终止测试或目标异常无法重启，跳转到第6步，否则继续。
10. 设i=1，N=iterations.size()。
11. 如果i大于N，则跳转到第6步，否则继续。
12. 执行当前组合测试用例。
13. i++，跳转到第3步。
14. 测试结束。

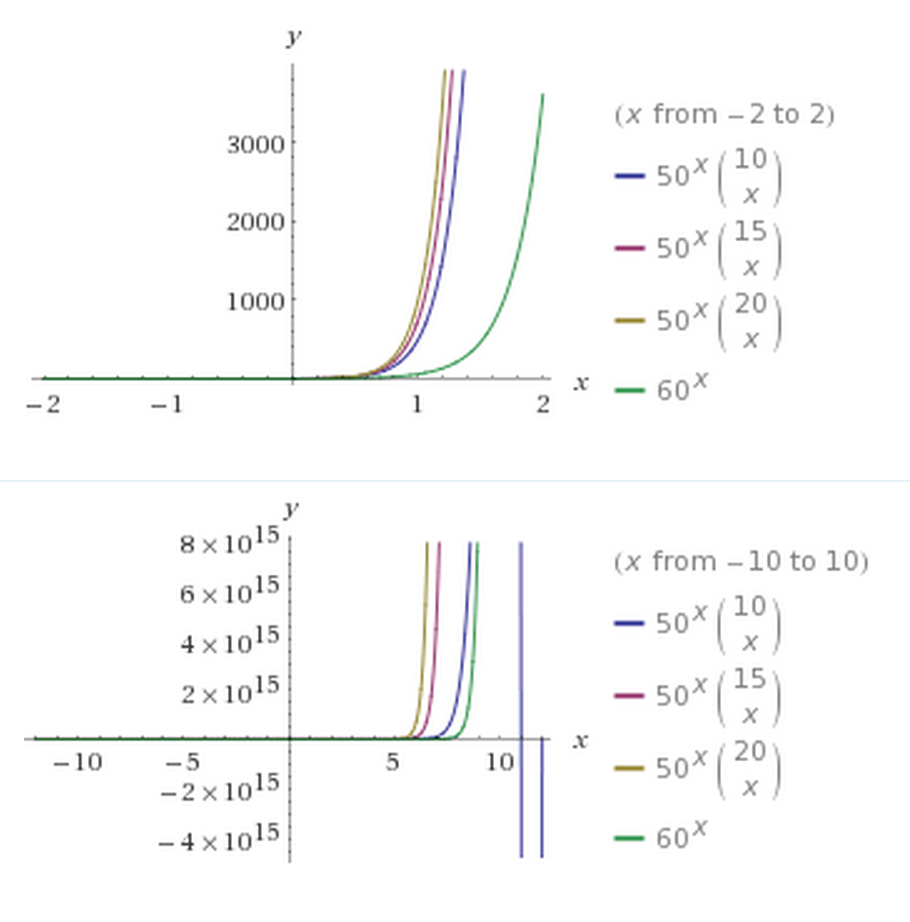
### 组合数与测试用例数总数

我们需要知道Combination策略生成的测试用例总数来估算测试时间， 为简便计算我们认为这里讨论的字段属于同一种类型，所以他们对应的变异方法是一样的，也近似认为每个变异方法的变异操作是一样的。

设共有N个字段，x个字段组合时生成的测试用例总数为f(x)，字段对应的变异方法有a个，变异方法包含的变异操作有b个。

现在讨论f(x)的特性，因为a、b为常数，根据实际经验我们假设a = 5, b = 10。如图四-7所示，图中分别描绘了N=10、15、20三种情况，可以直接得出结论：

1. 当x不变时，f(x)的值随N的增大缓慢增大。
2. 当N不变时，f(x)的值随x的增大迅速增大，近似于指数函数。



图四-7 f(x)=的函数绘图，绿色表示指数函数。

如表四-1所示，当a=5, b=10的情况下，不同字段数和组合数的情况下需要的测试时间，当组合数为1的时候该策略与Sequential策略和RandomDeterministic策略相同。当字段数为10时，组合数为2时，需要3个小时进行测试；当组合数为3时，需要17天进行测试，此时就得视情况而定，如果添加允许的话，则可以进行测试，否则就不能设置组合数为3；当组合数大于3时，需要的时间分别是4年、249年、10404年等，大大超出了我们能接受的范围。当字段数为15、20、30时，组合数为2的测试周期分别为7小时、13小时、30小时都是可以接受的；组合数为3的测试周期分别为66天、164天、587天，这就得看实际条件了；组合数大于3时，测试周期都需要1年以上，大大超出了我们能接受的范围。

因此，对于大部分情况使用组合数为2的组合策略进行测试是可行的，对于字段数较少时可以看实际情况决定是否使用组合数为3的组合策略进行测试。

由以上分析可知，Combination策略是可用的，且是Peach中变异策略的很好的补充，对于大部分情况，使用组合数为2的组合策略，可以快速的进行测试，其测试范围比Sequential策略和RandomDeterministic策略广很多，很有可能发现很多2字段组合情形的漏洞。

|  |  |  |  |
| --- | --- | --- | --- |
| **字段数** | **组合数** | **测试用例总数** | **测试时间** |
| 10 | 1 | 500个 | 50s |
| 2 | 112,500个 | 3h |
| 3 | 15, 000,000个 | 17天 |
| 4 | 1,312,500,000个 | 4年 |
| 5 | 78,750,000,000个 | 249年 |
| 6 | 3,281,250,000,000个 | 10404年 |
|  |  |  |  |
| 15 | 1 | 750个 | 75s |
| 2 | 262,500个 | 7h |
| 3 | 56,875,000个 | 66天 |
| 4 | 8,531,250,000个 | 27年 |
| 5 | 938,437,500,000个 | 2976年 |
| 6 | 78,203,125,000,000个 | 247980年 |
|  |  |  |  |
| 20 | 1 | 1, 000个 | 100s |
| 2 | 475, 000个 | 13h |
| 3 | 142,500,000个 | 164天 |
| 4 | 30,281,250,000个 | 96年 |
| 5 | 4,845,000,000,000个 | 15363年 |
| 6 | 605,625,000,000,000个 | 1920424年 |
|  |  |  |  |
| 30 | 1 | 1, 500个 | 150s |
| 2 | 1,087,500个 | 30h |
| 3 | 507,500,000个 | 587天 |
| 4 | 171,281,250,000个 | 543年 |
| 5 | 44,533,125,000,000个 | 141213年 |
| 6 | 9,277,734,375,000,000个 | 29419502年 |

表四-1 不同情形下的测试周期

## 组合爆炸问题

如表四-1所示，一般情况下当组合数大于3时，组合策略生成的测试用例总数大于10亿，相应的测试时间也需要若干年甚至更多。当组合数增加时，测试用例总数呈指数递增，测试时间也呈指数递增，这就出现了组合爆炸。

### 解决方案

解决组合爆炸问题的就是大幅减少测试用例总数，而测试用例总数与字段数、变异方法数、变异方法具有的变异操作数相关，下面就分几方面提出改进方案。

#### 减少字段数

由公式四-1和表四-1可知，减少字段数可以减少测试用例总数。实际测试场景中，有的字段之间联系密切，测试这些字段的组合就比较有价值；有的字段之间没有联系，测试这些字段的组合价值较低，为了减少测试用例总数，我们可以忽略这些价值较低的组合。

如图四-8所示，在Strategy标签下增加Groups标签，其内部可以有多个Group标签，Group标签内部包含有关联的若干字段。初始化变异序列时，只考虑Group标签内部的字段的组合情况。图四-8中表示字段num\_0、num\_1、num\_2、num\_3和num\_4之间具有关联，字段num\_7、num\_8和num\_9之间也具有关联。组合数为3时，先遍历字段num\_0、num\_1、num\_2、num\_3和num\_4选取3个字段的组合，之后再遍历字段num\_7、num\_8和num\_9选取3个字段的组合。这样就不会出现num\_0与num\_5组合的情况，也不会出现num\_0与num\_8组合的情况，测试用例数也就会显著减少。

图四-8表示的正常情况下总的测试用例数为1,327,500,000个，使用关联之后总的测试用例数为32,625,000个，大约是之前的1/40。

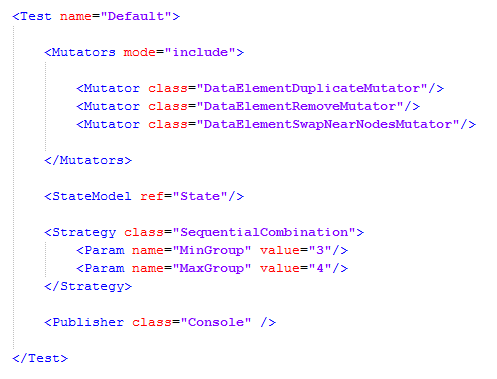


图四-8 对有关联的字段进行组合测试的Pit文件

#### 减少变异方法数

由公式四-1可知，当N固定时，测试用例总数与ab的积成指数关系，所以减少a和b的值可以大幅度减少测试用例总数。

实际测试场景中，有时候只需要测试几种变异方法，这种情况下我们就可以显示指定使用哪几个变异方法，如图四-10所示，指定了3个变异方法进行测试，即只在这3个变异方法中选择。假如该Pit文件中有10个字段，且都对应选择的这三个变异方法，正常情况下生成的测试用例总数为1,327,500,000个，减少变异方法数之后为173,340,000个，大约是之前的1/8。



图四-10 选择特定变异方法的Pit文件

除了减少变异方法数外，还可以减少变异方法中包含的变异操作数。不同的变异方法减少变异操作数的方法是不一样的，例如，对于DataElementDuplicateMutator方法，默认包含50个变异操作，就是将变异字段拷贝1~50次，可以将其改为1~50中随机选取5个进行拷贝，则修改之后该方法包含5个变异操作。对于UnicodeBadUtf8Mutator方法，该方法包含150个字符串，该方法的变异操作分别使用这150个字符串进行替换，这些字符串具有一些共性和特性，可以随机选取15个字符串进行替换，这样该方法的变异操作数就减少到15。

#### 限制测试用例数

有时候用户想多测试几个字段的组合情况，如果组合数大于5的话，使用上面两种方案的效果有限。当字段数为10是，默认情况下，组合数为3时包含15, 000,000个测试用例，组合数为4时包含1,312,500,000个测试用例，组合数越大测试用例数更多，为了达到测试组合数大于5的目的，可以设置每个组合数生成的测试用例数的上限。

如图四-11所示，测试组合数3~6的情形，设置组合数为3时的测试用例总数最多为1000个，但实际中组合数为3时具有的测试用例总数为15,000,000个，采用随机的方式从中选择1000个进行测试。组合数为4、5、6时与组合数为3时的策略是一样的，实际的测试用例总数为3,361,327,500,000个，设置上限之后的测试用例总数为1,111,000个，是正常情况的1/3025497。可见这种方式减少了很多的测试用例，能够满足用户测试多个组合数的需求。但是应该注意到该方法的局限性，扩大组合数的范围是在牺牲很多测试用例为代价的。这也就导致测试的不完整性，即可能有若干的潜在漏洞没有发现。



图四-11 设置每个组合数上限的Pit文件

## 实验

为了验证组合策略的是实用性、有效性，现对Peach源码进行修改以支持组合策略。并且使用同一Pit文件，分别使用Sequential策略和SequentialCombination策略进行测试，分析数据验证组合策略的实用性、有效性。

### 实验环境

Peach ：peach-3.0.202-source

Vulserver ：Vulnserver 1.00

操作系统：Windows 7 Ultimate Service Pack 1

集成开发环境：Microsoft Visual Studio Ultimate 2013 版本 12.0.30723.00 Update 3

.Net 框架：Microfost .Net Framework 版本 4.5.50938

### Peach源码编译

1. 解压peach-3.0.202-source.zip，转到目录peach-3.0.202-source/，下面的路径都是以该目录为跟路径。
2. 双击打开Peach.net文件（peach-3.0.202-source/Peach.net）
3. 出现提示“复查项目和解决方案更改”，选择“确定”。
4. 右键点击“解决方案’Peach.net’”，选择“属性”。
5. 左侧边栏选择“通用属性”中的“启动项目”，设置单启动项目为“Peach”。
6. 左侧边栏选择“配置属性”中的“配置”，上栏“配置”选择“Debug - Windows”，“平台”选择“x64”。
7. 左侧边栏选择“配置属性”中的“配置”，单击右上角“配置管理器”。“活动解决方案配置”选择“Debug - Windows”，“活动解决方案平台”选择“x64”，然后点击“关闭”。
8. 点击“ok”，关闭“解决方案‘Peach.net’”属性页。
9. 选择“生成”，单击“重新生成解决方案”。
10. 编译成功。
11. 转到目录： Peach/bin/x64/Debug，启动cmd，并转到该目录。
12. 执行命令：Peach ..\..\..\samples\HelloWorld.xml
13. 控制台中显示出Peach测试的过程。

### 源码修改

修改后的源码位于这里[github地址]，下面简要介绍下对源码的改动。

#### 编辑文件MutationStrategies.cs

编辑文件Peach.Core/MutationStrategy.cs，MutationStrategy是变异策略的父类，其子类有Sequential、RandomDeterministic和Random。因为新添加的SequentialCombination类中用到了MaxGroup和MinGroup属性，所以需要在MutationStrategy中加入MaxGroup和MinGroup属性。

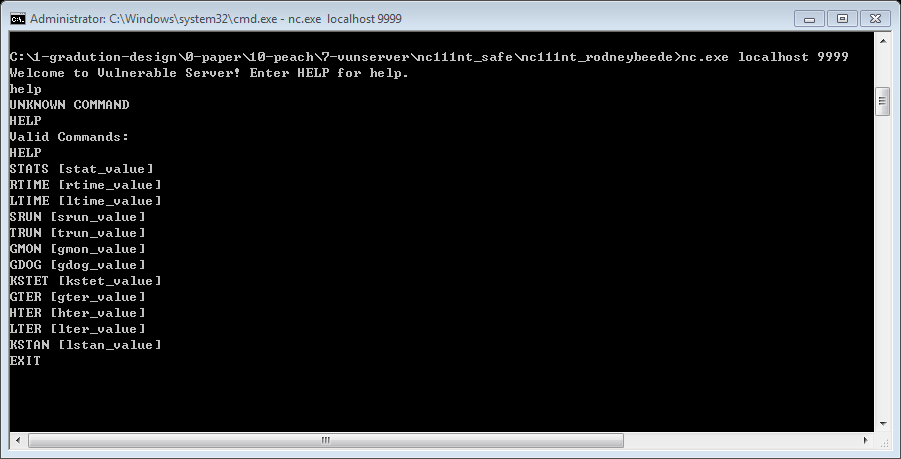
#### 添加文件SequentialCombination.cs

在Peach.Core/MutationStrategies目录下添加文件SequentialCombination.cs文件，该文件参考Sequential.cs文件，包含组合策略的主要实现：1) 初始化变异序列 2）控制测试过程。

### Pit文件

我们测试的目标就是Vulnserver，Vulnserver是由Stephan Bradshaw使用C语言编写的，它是专门用来练习漏洞挖掘的，提供了TCP服务器，用于接收消息和返回消息，Vulnserver中包含缓冲区溢出和SHE重写漏洞。

如图四-12所示，Vulnserver提供了多种消息通信格式，我们实验选择KSTET进行测试。下面描述Pit文件中的主要部分。



图四-12 Vulnserver的格式

#### DataModel

#### StateModel

#### Agent

#### Test

### 实验步骤

### 对比实验

### 分析实验结果

# 第五章 抽样[使用vulnserver进行测试][期望、方差]

## 为什么要抽样

当字段数量较多时，产生的测试用例数量就会非常大，需要的测试时间就会增多，但很多测试用例回报率很低，就会降低测试效率。

如果对字段进行抽样测试，而不是所有的字段都进行测试，就会显著的减少测试用例数量。同理，对变异方法和变异方法的取值也进行抽样，就会极大减少测试用例数量，从而减少测试周期，但是这是以降低测试用例覆盖率为代价的，合理的测试应该在测试周期与测试覆盖率之间寻找一个较好的方案。

## 抽样方法

### 简单随机抽样

从总体N个单位中随机地抽取n个单位作为样本，使得每一个单位都有相同的概率被抽中。特点是：每个样本单位被抽中的概率相等，样本的每个单位完全独立，彼此间无一定的关联性和排斥性。简单随机抽样是其它各种抽样形式的基础。通常只是在总体单位之间差异程度较小和数目较少时，才采用这种方法。

### 系统抽样

将总体中的所有单位按一定顺序排列，在规定的范围内随机地抽取一个单位作为初始单位，然后按事先规定好的规则确定其他样本单位。先从数字1到k之间随机抽取一个数字r作为初始单位，以后依次取r+k、r+2k……等单位。这种方法操作简便，可提高估计的精度。

### 分层抽样

将抽样单位按某种特征或某种规则划分为不同的层，然后从不同的层中独立、随机地抽取样本。从而保证样本的结构与总体的结构比较相近，从而提高估计的精度。

### 整群抽样

将总体中若干个单位合并为组，抽样时直接抽取群，然后对中选群中的所有单位全部实施调查。抽样时只需群的抽样框，可简化工作量，缺点是估计的精度较差。

结合Peach中测试用例的特点，选用简单随机抽样和系统抽样两种方法，其中默认为简单随机抽样，如果用户指定使用系统抽样则使用系统抽样。

## 抽样描述

### 抽样设计

#### 覆盖率的定义

字段覆盖率（Element Rate of Coverage）：指抽样选取的字段数在总的字段数中的比例。

变异方法覆盖率（Mutator Rate of Coverage）：指抽样选取的变异方法数在总的变异方法数中的比例。

变异方法取值覆盖率（Value Rate of Coverage）：指抽样选取的变异方法取值数在总的变异方法取值数中的比例。

用户可以自定义ERC、MRC和VRC的值，抽样普遍适用于已有的策略以及新添加的Combination策略。如果三个值都为1，则相当于没有进行抽样，用户可以根据实际需要，自行决定ERC、MRC和VRC的值。

#### 定义抽样方法

根据5.2 中抽样方法的描述，在Peach中可以使用简单随机抽样和系统抽样。

对于简单随机抽样方法。第一步，根据字段覆盖率，在所有待测试的字段中随机选取一些字段，只对这些字段进行测试；第二步，根据变异方法覆盖率，在所有变异方法中随机选取一些变异方法，只对这些变异方法进行测试；第三部，根据变异方法取值覆盖率，在待测试的变异方法中随机选取一些变异方法取值，该变异方法中只取这些值。该方案的优点是简单易实现，缺点是随机性太强，不同的测试周期之间总的测试用例相差较大。

下面把这种方法表示成伪代码，E表示字段、变异方法或变异方法取值的容器，R表示相应的覆盖率，Random(i, j)表示在i到j中随机一个整数，包括i和j。

SIMPLE-RANDOM-SAMPLE(E, R)

n = E.size()

m = E \* R

for i = 1 to m

j = Random(i, m)

swap(E[i], E[j])

对于系统随机抽样方法。第一步，根据字段覆盖率，在所有待测试的字段中使用系统随机抽样方法选取一些字段，第二步和第三步，分别对变异方法和变异方法取值使用系统随机抽样。该方案的优点是简单易实现，且分布均匀，缺点是随机性太强，不同的测试周期之间总的测试用例相差较大。

下面把这种方法表示成伪代码，E表示字段、变异方法或变异方法取值的容器，R表示相应的覆盖率，Random(i, j)表示在i到j中随机一个整数，包括i和j。

SYSTEM-RANDOM-SAMPLE(E, R)

n = E.size()

m = E \* R

T = n / m

r = Random(1, T)

let NE[1...m] be a new array

for i = 1 to m

NE[i] = E[r + T\*(i-1)];

return NE

## 测试用例总数

设共有N个字段，每个字段都有M个变异方法，每个变异方法都有P个取值；字段的覆盖率ERC为A，变异方法的覆盖率MRC为B，变异方法的取值的覆盖率VRC为C。

下面表格1和表格2中分别列出了不使用抽样和使用抽样时测试用例总数。

|  |  |
| --- | --- |
| 策略 | 测试用例总数 |
| RandomDeterministic | NMP个 |
| Sequential | NMP个 |
| Random | 个 |
| Combination | （i为组合数） |

表格 2不使用抽样

|  |  |
| --- | --- |
| 策略 | 测试用例总数 |
| RandomDeterministic | ANBMCP个 |
| Sequential | ANBMCP个 |
| Random | 个 |
| Combination | （i为组合数） |

表格 3 使用抽样

可知，除Random策略外，对于它策略使用抽样都可以有效地减少测试用例数。

## 覆盖率的取值

采用抽样选取测试用例的方案可以减少总的测试用例数目，加速了测试速度。为了在速度与测试覆盖率之间寻找一个较好的方案，需要选择合适的抽样值。

如何选取合适的字段覆盖率、变异方法覆盖率以及变异方法取值覆盖率呢？

Peach中的变异方法有24种，即M=24，P的取值则普遍比较大，因此如果字段较多，即N比较大，则令A取比较小的值，如果字段较少，则令A取比较大的值；B取值适中即可；C可以取比较小的值。

测试人员需要根据实际情况，对A、B、C取合适的值，已达到测试目的。

## 具体实现

这里是Peach代码

## 实验结果

这里是实验数据

# 标题

（格式：正文1）图、公式以及表格的格式对于长文档编辑来说是一个很麻烦的内容，修改起来往往非常麻烦和繁琐，往往让人望而生畏，在本章将介绍图、公式以及表格的格式以及引用的方法。

## 图格式及引用

（格式：正文1）图格式通常是一个让人头疼的问题，我们往往会遇到这样的问题：当移动图片位置时，文档也会随着发生变化，有时甚至找不到图片在什么位置去了。这是由于我们在插入图片时引用了一些不可靠的位置。为了保证文档的稳定性，建议在插入图片时，选择图片的文字环绕样式为“嵌入式”。

图片的插入、格式及引用的步骤如下：

1. 在要插入图片的位置处另起一行，选择样式“图片样式”后，点击插入图片。插入后将图片文字环绕样式设置为“嵌入式”。
2. 选中插入的图片，选择引用下的插入题注，在“标签”处选择“图”后，点击确定。
3. 在要引用图题注的位置，点击插入交叉引用，引用类型选择“图”，引用内容为“只有标签和编号”。
4. 对于一排需要放置多张不同序号的图的情况，可以通过应用分栏来实现。
5. 我们可以看到图题注中有含有中文序号，这是我们不希望看到的，那么如何更改，在此暂且不急，在后面将介绍。

示例如下：

如**错误！未找到引用源。**所示为一音圈电机驱动的并联式二维运动平台系统，系统由音圈电机二维平台、音圈电机驱动箱和运动控制箱构成。

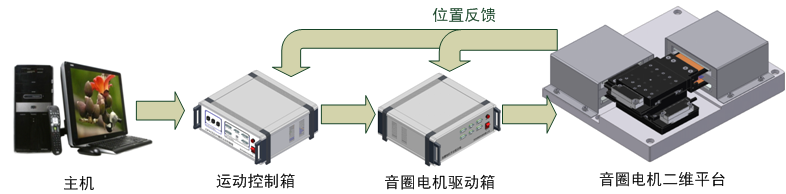


图 二‑1音圈电机驱动二维运动平台系统

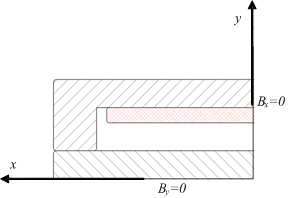


图 二‑2 四分之一磁路

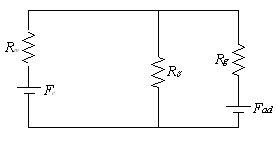


图 二‑3 等效电路图

## 公式格式及引用

公式的格式及引用通常也是非常的麻烦，特别是编号，我是花了大量的时间才摸索出来。因为我们没有安装正版的公式编辑器，编号就不是那么容易。

在编辑公式时，首先需要安装公式编辑器MathType，最好是5.0以上的版本。有了这个工具之后，编辑公式就会让我们轻松的多，下面介绍公式的格式及引用。

示例如下：

//////

力的大小与磁场强度*B*、电流大小*i*以及导体有效长度*l*成正比。对处于单恒定磁场中的N匝线圈，安培力表达式为：

 (二–1)

由于音圈电机常采用永磁材料作为励磁，其中磁场可当着恒定磁场，那么式(二–1)亦即为音圈电机的力输出表达式。

/////

例子中的公式格式及编号格式已经做好，在添加新公式时，只需将该行复制后，修改公式中的内容。然后选中编号内容，按F9进行更新。在引用该公式编号时，在要引用公式编号的位置，点击插入交叉引用，引用类型选择“式”，引用内容为“只有题注文字”。

## 表格式及应用

表格通常要求采用三线表，选择插入表格，后选择表格样式为三线表。

选中表格，选中插入的图片，选择引用下的插入题注，在“标签”处选择“表”后，点击确定。

在要引用图题注的位置，点击插入交叉引用，引用类型选择“表”，引用内容为“只有标签和编号”。

示例如下：

在设计电机时通常需要考虑电机的技术参数，音圈电机常用技术参数见表 二‑1。在此将对其中几个参数作进一步说明。

表 二‑1音圈电机常见技术参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 说明 | 参数 | 说明 |
| *Fc* | 连续推力（Continuous Force）N | *Fp* | 峰值推力（Peak Force）N |
| *Kf* | 推力常数（Force Constant）N/A | *Km* | 电机常数（Motor Constant）N/sqrt(W) |
| *τe* | 电气时间常数（Elec. Time Constant）s | *τm* | 机械时间参数（Mech. Time Constant）s |
| *Re* | 阻抗（Resistance）Ohm | *Le* | 感抗（Inductance）H |
| *S* | 平动行程（Stroke）m | *Kv* | 反电动势常数（Back EMF constant）V/m/s |

## 题注中的中文序号替换

我们可以看到在图的题注、公式编号以及表题注中都存在中文编号，而论文中的要求通常是西文序号，因此我们要对中文序号进行替换。步骤如下：

1. 本步骤必须在全文结束时，确定不再有图、表及公式的插入时进行。为了防止出现以后需要再插入时的麻烦，可以先将带有中文编号的文档储存一个副本。
2. 选择全部文档“Ctrl+A”后，按“Alt+F9”切换域代码。
3. 按下“Ctrl+F9”，出现大括号{ }，在{ }中输入SEQ seqp \c，注意不要漏掉空格。将大括号的内容（连同大括号）选中后复制“Ctrl+C”
4. 按下“Ctrl+H”进行代码替换，填入如图 二‑4所示。
5. 替换完成后，全选文档“Ctrl+A”后，按“Alt+F9”切换域代码。然后按F9进行代码更新即可。

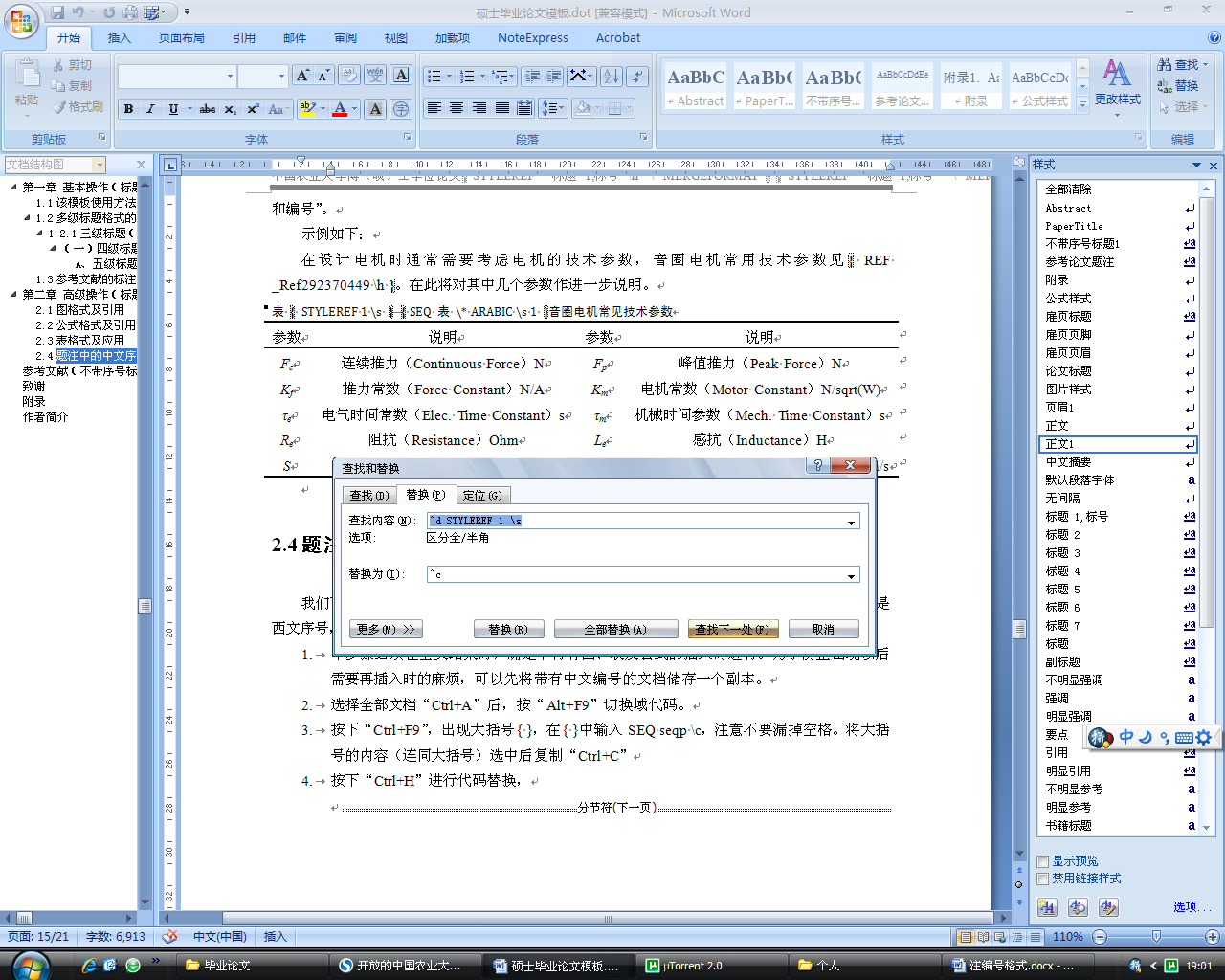


图 二‑4 替换图

# 结论与展望

## 本文总结

## 展望

### 三级标题（格式：标题3）

#### 四级标题（格式：标题4）

##### 五级标题

参考文献

* 1. Rune Hammersland, EinarSnekkenes. Fuzz testing of web applications. Faculty of Computer Science and Media Technology ,Gjøvik University College, Norway.
  2. Hao Wang, Qiaoyan Wen, Zhao Zhang. Improvement of Peach Platform to support GUI-based Protocol State Modeling. State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications.
  3. Patrice Godefroid. Automated Whitebox Fuzz Testing. Microsoft (Research).
  4. 赵丽娟，温巧燕，张华. 基于Peach的协议测试设计与实现. 北京邮电大学网络与交换技术国家重点实验室, 2011.
  5. 李伟明. 网络协议的自动化模糊测试漏洞挖掘方法. 华中科技大学计算机学院, 2011.
  6. 吴志勇. Fuzzing技术综述 [A]. 解放军电子工程学院网络工程系, 2010.
  7. 郭瑞杰. 软件安全性测试技术研究. 北京邮电大学网络与技术研究院, 2009.
  8. 郑叔芳. 随机模糊测试的确定性. 南京航空航天大学5 系, 1993.
  9. 迟强, 罗红, 乔向东. 漏洞挖掘分析技术综述[J]. 计算机与信息技术, 2009.

致谢

所有的老师都得感谢一遍

最后，我要感谢张钊博士和王浩师兄，理解与支持，谢谢！

附录

1. 格式：附录

作者简介

格式：正文1