

《测试驱动数据库开发》

图书基本信息

书名：《测试驱动数据库开发》

13位ISBN编号：9787115346283

出版时间：2014-6

作者：[美] Max Guernsey, III

页数：270

译者：伍斌

版权说明：本站所提供下载的PDF图书仅提供预览和简介以及在线试读，请支持正版图书。

更多资源请访问：www.tushu111.com

《测试驱动数据库开发》

内容概要

测试驱动开发(TDD)的实践已经帮助众多软件开发人员提高了软件开发的质量、敏捷性、生产力和速度,《测试驱动数据库开发》将展示如何对TDD进行调整,以便在数据库设计与开发工作中获得同样强大的优势。《测试驱动数据库开发》共4个部分,全面介绍测试驱动数据库开发(TDDD)技术。第1章至~第4章重点讨论数据库的类的基本概念,第5章至第9章讨论如何用面向对象的方式来精益地做数据库的类的设计以及修复设计的错误,第10章至第13章讨论使用mocking和重构来应对由传统方法开发出来的遗留数据库的两种方法,第14章和第15章讨论如何使一个数据库应用系统能够满足不同客户的不同需求,以及如何将本书的技术运用到其他数据持久化方案之中。

《测试驱动数据库开发》适合没有接触过测试驱动开发且正在开发规模较大、需求多变的数据库应用系统的开发人员和架构师阅读,同时也适合尚未在持久化层运用测试先行开发技术的测试驱动开发爱好者阅读。

《测试驱动数据库开发》

作者简介

Max Guernsey, III

Hexagon软件公司的管理成员，他以该公司作为一个平台，将真正的数据库和用户界面的敏捷性引入到那些已经在中间层采用了敏捷软件开发方法的组织之中。他有十余年的软件开发经验，其中大部分经验是在敏捷软件开发的环境中获得的，这期间的几乎一半时间，他一直就敏捷和测试驱动数据库开发主题，写博客、写作和发表演讲。通过Net Objectives，他开设“数据库敏捷培训”这样一门开创性的课程，该课程专注于频繁、安全和毫无痛苦地变更数据库设计所需的关键的技术技能。他的博客maxg3prog.blogspot.com。

伍斌

独立匠艺程序员。专注于测试驱动开发、驯服烂代码及编程操练。除翻译本书外,还在撰写《驯服烂代码》和《会运行的文档》两本书。自从1993年大学毕业以来,先后做过程序员、测试工程师、项目经理和软件开发咨询师。2013年4月创办公益编程操练社区“bjdp.org北京设计模式学习组”。个人网站wubinben.com，微信公众号bjdp_org。

书籍目录

第1章 为何改变书的内容、谁是目标读者和什么是障碍	1
1.1 为何改变书的内容	1
1.1.1 每天敏捷都在逐步地入侵我们的领域	2
1.1.2 若没有TDD敏捷就没有成效	2
1.1.3 在数据库领域运用TDD是个挑战	3
1.2 谁是目标读者	3
1.2.1 TDD和OOP	3
1.2.2 应用程序和数据库	4
1.3 什么是障碍	4
1.3.1 数据库就是对象	4
1.3.2 TDD适用于类，不适用于对象	4
1.3.3 我们需要数据库的类	5
1.4 小结	6
第2章 建立数据库的类	7
2.1 TDD中类的角色	7
2.1.1 可靠的实例化过程	7
2.1.2 测试检查对象	8
2.2 面向对象编程语言中的类	8
2.2.1 类的构建很容易：构建新对象即可	8
2.2.2 一条途径：必要时析构	9
2.3 数据库的类	9
2.3.1 两条途径：创建或改变	10
2.3.2 难点：统一两条途径	10
2.3.3 真实的数据库的生长情况	11
2.3.4 将每个数据库构建成生产数据库会怎么样	11
2.3.5 所有数据库都遵循完全相同的途径	12
2.4 增量构建	12
2.4.1 用文档记录每一次数据库的变更	12
2.4.2 标识当前版本	13
2.4.3 根据需要依次实施变更	13
2.5 实现	13
2.5.1 需求	13
2.5.2 数据库实例化机制的伪代码	14
2.5.3 输入的伪代码	14
2.6 小结	14
第3章 讲一点TDD	16
3.1 测试先行的技术	16
3.1.1 编写测试代码	17
3.1.2 让测试失败得有一些有价值的启示	19
3.1.3 看到测试运行通过	19
3.1.4 重复	20
3.2 测试即规格	21
3.2.1 “测试不是测试，而是规格”	21
3.2.2 “测试不是规格，而是测试”	22
3.2.3 测试是可运行的规格	22
3.2.4 增量设计	24
3.3 构建良好的规格	24

- 3.3.1 规定行为，而不是结构 24
- 3.3.2 从一无所有开始驱动设计，而不是从其他方式开始 25
- 3.3.3 从内向外地定义设计 25
- 3.3.4 从外向内地定义设计 27
- 3.4 小结 29
- 第4章 安全地改变设计 31
 - 4.1 什么是安全 31
 - 4.1.1 违约有点糟 32
 - 4.1.2 丢失数据可能会让你被炒鱿鱼 33
 - 4.1.3 不改设计也同样危险 34
 - 4.2 解决方案：过渡测试 37
 - 4.2.1 测试驱动的实例化 37
 - 4.2.2 建立过渡测试 38
 - 4.2.3 累加变化的过渡测试 40
 - 4.2.4 过渡测试的变形 44
 - 4.2.5 为什么不使用公共接口 49
 - 4.3 过渡保障 49
 - 4.3.1 Read/Read过渡测试 49
 - 4.3.2 每次升级时通过数据库的类来运行 52
 - 4.3.3 备份和失败时回滚 53
 - 4.3.4 让过渡测试充分利用过渡保障 53
 - 4.4 小结 54
- 第5章 遵循接口 55
 - 5.1 接口的优势 55
 - 5.1.1 更强的耦合语言 56
 - 5.1.2 弱耦合的语言 57
 - 5.1.3 共识 57
 - 5.1.4 耦合到数据库的类 58
 - 5.1.5 问题是发生了重复 58
 - 5.2 像客户对象般的遵循 58
 - 5.2.1 创建DatabaseDesign类的需求 58
 - 5.2.2 规定DatabaseDesign类 60
 - 5.2.3 摆脱使用多个客户端平台时出现的重复 61
 - 5.2.4 当耦合出问题时会发生什么 62
 - 5.2.5 消除数据库构建和客户端代码之间的重复 62
 - 5.2.6 解除实现与设计之间的耦合 63
 - 5.3 症结：变更 64
 - 5.3.1 随时间而变化的设计 64
 - 5.3.2 记录所有版本的设计 65
 - 5.3.3 耦合到设计的正确版本 68
 - 5.4 症结：耦合 69
 - 5.4.1 不同的客户端耦合到不同的版本 69
 - 5.4.2 总是不得不修改所有东西也是重复 69
 - 5.4.3 透镜概念介绍 73
 - 5.4.4 虚拟透镜 76
 - 5.4.5 “当前”透镜 78
 - 5.4.6 “新”透镜 79
 - 5.5 小结 81
- 第6章 定义行为 82

- 6.1 一组新问题 83
 - 6.1.1 无封装 83
 - 6.1.2 隐藏一切 84
 - 6.1.3 数据库中的业务逻辑 84
- 6.2 知识、信息与行为 85
 - 6.2.1 通告 86
 - 6.2.2 知识 88
 - 6.2.3 行为 90
- 6.3 由外而内地开发 92
 - 6.3.1 定义测试 93
 - 6.3.2 生长出接口 94
 - 6.3.3 生长出行为和结构 95
- 6.4 用规格来实现合理的设计 97
 - 6.4.1 开发当下的需求，而不是将来的需求 97
 - 6.4.2 用增量的方式构建 98
 - 6.4.3 将访问限定在规定的內容上 98
 - 6.4.4 小结 99
- 第7章 为可维护性而构建 100
 - 7.1 再也不要担心未来 100
 - 7.1.1 在当下寻找机会 101
 - 7.1.2 针对通告进行设计 102
 - 7.1.3 使用行为来翻译通告和知识 106
 - 7.2 用激情和热忱来保护知识 108
 - 7.2.1 不做改变是最危险的选择 108
 - 7.2.2 让设计保持自然 110
 - 7.3 当事情在未来发生时再处理 111
 - 7.3.1 定义新的设计 111
 - 7.3.2 引入最小的变化 113
 - 7.3.3 让测试运行通过 115
 - 7.3.4 停下来，思考，重构 117
 - 7.3.5 小结 119
- 第8章 错误与修复 121
 - 8.1 各种错误 121
 - 8.1.1 轴：好的错误还是坏的错误 122
 - 8.1.2 轴：错误发布了没有 124
 - 8.2 处理好的错误 125
 - 8.2.1 修复它就好了 125
 - 8.2.2 现在就记录行为 126
 - 8.2.3 回溯功能的根源 128
 - 8.3 处理坏的错误 129
 - 8.3.1 未发布的错误 129
 - 8.3.2 已发布的错误 133
 - 8.3.3 灾难性的错误 139
 - 8.4 小结 140
- 第9章 设计 141
 - 9.1 结构与设计 142
 - 9.1.1 结构：执行细节 142
 - 9.1.2 测试和类信息 144
 - 9.2 什么是设计 144

- 9.2.1 概念之桶 145
- 9.2.2 真正的TDD中强制性的部分 147
- 9.3 组合与聚合 148
 - 9.3.1 组合：一件事有多个组成部分 148
 - 9.3.2 聚合：连接截然不同的东西 151
- 9.4 复用 154
 - 9.4.1 避免将同样的内容开发两遍 154
 - 9.4.2 通过组合或聚合来实现复用 156
- 9.5 抽象 157
 - 9.5.1 发现运用抽象的机会 157
 - 9.5.2 封装行为 159
 - 9.5.3 寻找各种方式来允许变化发生在依赖关系中 164
 - 9.5.4 处理时间问题 165
- 9.6 小结 169
- 第10章 mocking 171
 - 10.1 测试单个的行为 171
 - 10.1.1 为什么封装 172
 - 10.1.2 测试就是对那些在其控制之外的一切进行测试 172
 - 10.1.3 从测试那里来控制不相关的行为 173
 - 10.1.4 mocking控制了行为 174
 - 10.2 在面向对象编程中的mocking 174
 - 10.2.1 设置 175
 - 10.2.2 解耦 179
 - 10.2.3 隔离 181
 - 10.2.4 集成 182
 - 10.3 在数据库设计中使用mocking 182
 - 10.3.1 示例问题 183
 - 10.3.2 示例解决方案 184
 - 10.3.3 组合 187
 - 10.3.4 聚合 188
 - 10.3.5 为可测试性而设计 188
 - 10.4 小结 189
- 第11章 重构 190
 - 11.1 什么是重构 190
 - 11.1.1 改变设计但不改变行为 191
 - 11.1.2 在测试运行通过的背景下 192
 - 11.2 较低和较高风险的设计变更 199
 - 11.2.1 较低风险：改变类一级的设计 199
 - 11.2.2 中等风险：重新安排行为的逻辑 200
 - 11.2.3 较高风险：改变知识的容器 202
 - 11.2.4 这不是一个跳过测试的邀请 202
 - 11.3 小结 202
- 第12章 遗留数据库 203
 - 12.1 提升到一个类 203
 - 12.1.1 推导初始版本 204
 - 12.1.2 用测试来钉牢过渡行为 206
 - 12.2 控制耦合 207
 - 12.2.1 识别和锁定现有的使用数据库的情况 207
 - 12.2.2 按需封装 209

- 12.3 控制变更 210
 - 12.3.1 用测试驱动新的行为 210
 - 12.3.2 按需钉牢构造行为 212
 - 12.3.3 按需钉牢行为 213
 - 12.3.4 实现新的行为 214
- 12.4 查找接缝和组件 215
 - 12.4.1 查找接缝 215
 - 12.4.2 封装组件 218
- 12.5 小结 222
- 第13章 Fa?ade模式 224
 - 13.1 使用Fa?ade的封装 224
 - 13.1.1 Fa?ade模式的说明 225
 - 13.1.2 测试驱动开发出来的新的Fa?ade数据库 229
 - 13.1.3 使用组合方法的替代方案 235
 - 13.1.4 封装还是不封装 235
 - 13.2 扼杀旧接口 236
 - 13.2.1 将正在改变的行为转移到Fa?ade 236
 - 13.2.2 当不再需要时删除访问权限和功能 237
 - 13.3 在Fa?ade数据库中对行为进行测试驱动开发 238
 - 13.3.1 暴露遗留的行为 238
 - 13.3.2 做事情的另一方法 239
 - 13.3.3 新的行为 239
 - 13.4 小结 241
- 第14章 变奏曲 242
 - 14.1 重要的是拥有一个类，而不是实现 243
 - 14.2 场景：跳过那些步骤 243
 - 14.2.1 问题 243
 - 14.2.2 解决方案 244
 - 14.2.3 正确的工作量 246
 - 14.3 偏离 246
 - 14.3.1 问题 246
 - 14.3.2 解决方案 247
 - 14.3.3 应用解决方案 248
 - 14.4 通用的解决方案 252
 - 14.5 小结 252
- 第15章 其他应用 254
 - 15.1 XML 255
 - 15.1.1 封装 255
 - 15.1.2 XSD Schema 255
 - 15.1.3 XSLT过渡 257
 - 15.1.4 对XSLT的变更进行过渡测试 258
 - 15.2 文件系统和其他的对象目录 259
 - 15.2.1 对文件系统的操作进行过渡测试 259
 - 15.2.2 Shell脚本过渡 261
 - 15.3 数据对象 262
 - 15.3.1 类的定义就是Schema 262
 - 15.3.2 对Ugrader类进行过渡测试 263
 - 15.3.3 编写过渡 266
 - 15.4 小结与寄语 270

精彩短评

1、翻译让人看不懂，也许也是我理解不了吧。

1、裸奔的数据库 读<<测试驱动数据库开发>>序中的这一段有感而发：“一旦客户的需求到了程序员手中,先找出需求中的实体,在分析这些实体之间的关系,并画出E-R图,然后确定各个实体之间的属性,并找出主键,最后根据这份E-R图在数据库中生成数据库表,之后就可以用象Java这样的编程语言来进行应用系统的开发了”多么熟悉的声音啊,感觉昨天就是这么干的,不排除这种做法完全没有用武之地,但是极其少见。这么干过的朋友们,不知道遇到下面这些场景没有?数据库建好了,写应用的时候,发现缺少一个关键概念,只好加一个表。应用写完了,发现当初建的数据库里有相当一部分表根本没用上。。。。。。这是一种面向过程的设计思维,只有在概念已经非常稳定,在可预见的将来不会发生变化的情况下比较适用,比如我们是对数学定理建模,对分子式进行建模等,这些模型的特点就是稳定,除非科学上有质的突破,都不会发生变化。但是现实中这类需求很少很少,大部分都是和人财物打交道,而牵扯到这三者的概念,都具有很大的不稳定性,这种需求下,面向过程的设计就显得吃力。《测试驱动数据库开发》提出了一个新的视角,以面向对象的方式去进行数据库的设计,那么作者的场景(绿色文字)就变成下面这样:“一旦客户的需求到了程序员手中,可以用象Java这样的编程语言来进行应用系统的开发,这时候需要存储的数据就先用文本文件来存储吧,一个应用模块开发完成后,从这个模块存储需求中找出实体,分析这些实体间的关系,并画出E-R图,然后确定实体的属性,并找出主键,最后根据这份E-R图在数据库中生成数据库表”和前面场景的最大区别就是,这里在开发的最后才去考虑数据库的存储设计,这样的设计下来,数据库倚赖应用模块,随着应用模块的变化而变化,正如面向对象设计里的倚赖倒置原则。最后有一点提示:能用文本文件组织起来的存储需求,转到数据库存储是很简单的事情,但是反过来,可不一定

《测试驱动数据库开发》

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:www.tushu111.com