Java软件开发中可能出现几个错误观点Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/560/2021_2022_Java_E8_BD_AF_E4_BB_B6_c104_560177.htm Java，但是他们大多数人没有做好足够的思想准备(没有接受OO思想体系相关培训)，以致不能很好驾驭Java项目，甚至 导致开发后的Java系统性能缓慢甚至经常当机。很多人觉得这是Java复杂导致，其实根本原因在于：我们原先掌握的关于软件知识(OO方面)不是太贫乏就是不恰当，存在认识上和方法上的误区。 软件的生命性 软件是有生命的，这可能是老调重弹了，但是因为它事关分层架构的原由，反复强调都不过分。 一个有生命的软件首先必须有一个灵活可扩展的基础架构，其次才是完整的功能。 目前很多人对软件的思想还是焦点落在后者：完整的功能，觉得一个软件功能越完整越好，其实关键还是架构的灵活性，就是前者，基础架构好，功能添加只是时间和工作量问题，但是如果架构不好，功能再完整，也不可能包括未来所有功能，软件是有生命的，在未来成长时，更多功能需要加入，但是因为基础架构不灵活不能方便加入，死路一条。 正因为普通人对软件存在短视误区，对功能追求高于基础架构，很多吃了亏的老程序员就此离开软件行业，带走宝贵的失败经验，新的盲目的年轻程序员还是使用老的思维往前冲。其实很多国外免费开源框架如ofbiz compiere和slide也存在这方面陷阱，貌似非常符合胃口，其实类似国内那些几百元的盗版软件，扩展性以及持续发展性严重不足。 那么选择现在一些流行的框架如Hibernate、Spring/Jdonframework是否就表示基础架构打好了呢?其实还不尽然，关键还是取决于你如何使用这些

框架来搭建你的业务系统。 存储过程和复杂SQL语句的陷阱 首先谈谈存储过程使用的误区，使用存储过程架构的人以为可以解决性能问题，其实它正是导致性能问题的罪魁祸首之一，打个比喻：如果一个人频临死亡，打一针可以让其延长半年，但是打了这针，其他所有医疗方案就全部失效，请问你会使用这种短视方案吗? 为什么这样说呢?如果存储过程都封装了业务过程，那么运行负载都集中在数据库端，要中间J2EE应用服务器干什么?要中间服务器的分布式计算和集群能力做什么?只能回到过去集中式数据库主机时代。现在软件都是面向互联网的，不象过去那样局限在一个小局域网，多用户并发访问量都是无法确定和衡量，依靠一台数据库主机显然是不能够承受这样恶劣的用户访问环境的。(当然搞数据库集群也只是五十步和百步的区别)。 从分层角度来看，现在三层架构：表现层、业务层和持久层，三个层次应该分割明显，职责分明：持久层职责持久化保存业务模型对象，业务层对持久层的调用只是帮助我们激活曾经委托其保管的对象，所以，不能因为持久层是保管者，我们就以其为核心围绕其编程，除了要求其归还模型对象外，还要求其做其做复杂的业务组合。打个比喻：你在火车站将水果和盘子两个对象委托保管处保管，过了两天来取时，你还要求保管处将水果去皮切成块，放在盘子里，做成水果盘给你，合理吗? 上面是谈过分依赖持久层的一个现象，还有一个正好相反现象，持久层散发出来，开始挤占业务层，腐蚀业务层，整个业务层到处看见的是数据表的影子(包括数据表的字段)，而不是业务对象。这样程序员应该多看看OO经典PoEAA。PoEAA 认为除了持久层，不应该在其他地方看到数据表或表字段名。

当然适量使用存储过程，使用数据库优点也是允许的。按照Evans DDD理论，可以将SQL语句和存储过程作为规则Specification一部分。 Hibernate等ORM问题 现在使用Hibernate人也不少，但是他们发现Hibernate性能缓慢，所以寻求解决方案，其实并不是 Hibernate性能缓慢，而是我们使用方式发生错误：“ 最近本人正搞一个项目，项目中我们用到了struts1.2 hibernate3, 由于关系复杂表和表之间的关系很多，在很多地方把lazy都设置false，所以导致数据一加载很慢，而且查询一条数据更是非常的慢。 ”Hibernate是一个基于对象模型持久化的技术，因此，关键是我们需要设计出高质量的对象模型，遵循DDD领域建模原则，减少降低关联，通过分层等有效办法处理关联。如果采取围绕数据表进行设计编程，加上表之间关系复杂(没有科学方法处理、侦察或减少这些关系)，必然导致 系统运行缓慢，其实同样问题也适用于当初对EJB的实体Bean的CMP抱怨上，实体Bean是Domain Model持久化，如果不首先设计Domain Model，而是设计数据表，和持久化工具设计目标背道而驰，能不出问题吗?关于这个问题N多年就在Jdon争论过。 这里同样延伸出另外一个问题：数据库设计问题，数据库是否需要在项目开始设计? 如果我们进行数据库设计，那么就产生了一系列问题：当我们使用Hibernate实现持久保存时，必须考虑事先设计好的数据库表结构以及他们的关系如何和业务对象实现映射，这实际上是非常难实现的，这也是很多人觉得使用ORM框架棘手根本原因所在。 当然，也有脑力相当发达的人可以 实现，但是这种围绕数据库实现映射的结果必然扭曲业务对象，这类似于两个板块(数据表和业务对象)相撞，必然产生地震，地震的

结果是两败俱伤， 软的一方吃亏，业务对象是代码，相当于数据表结构，属于软的一方，最后导致业务对象变成数据传输对象DTO, DTO满天飞，性能和维护问题随之而来。 领域建模解决了上述众多不协调问题，特别是ORM痛苦使用问题，关于ORM/Hibernate使用还是那句老话：如果你不掌握领域建模方法，那么就不要用Hibernate，对于这个层次的你：也许No ORM 更是一个简单之道。 Spring分层矛盾问题 Spring是以挑战EJB面貌出现，其本身拥有的强大组件定制功能是优点，但是存在实战的一些问题，Spring作为业务层框架，不支持业务层Session 功能。 具体举例如下：当我们实现购物车之类业务功能时，需要将购物场合保存到Session中，由于业务层没有方便的Session支持，我们只得将购物车保存到HttpSession，而HttpSession只有通过HttpRequest才能获得，再因为在Spring业务层容器中是无法访问到HttpRequest这个对象的，所以， 最后我们只能将“ 购物车保存到HttpSession ”这个功能放在表现层中实现，而这个功能明显应该属于业务层功能，这就导致我们的Java项目层次混乱，维护性差。 违背了使用Spring和分层架构最初目的。 领域驱动设计DDD 现在回到我们讨论的重点上来，分层架构是我们使用Java的根本原因之一，域建模专家Eric Evans在他的“ Domain Model Design ”一书中开篇首先强调的是分层架构，整个DDD理论实际是告诉我们如何使用模型对象oo技术和分层架构来设计实现一个Java项目。 我们现在很多人知道Java项目基本有三层：表现层 业务层和持久层，当我们执著于讨论各层框架如何选择之时，实际上我们真正的项目开发工作还没有开始， 就是我们选定了某种框架的组合(如Struts Spring Hibernate或Struts EJB

或Struts JdonFramework)，我们还没有意识到业务层工作还需要大量工作，DDD提供了在业务层中再划分新的层次思想，如领域层和服务层，甚至再细分为作业层、能力层、策略层等等。通过层次细化方式达到复杂软件的松耦合。DDD提供了如何细分层次的方式 当我们将精力花费在架构技术层面的讨论和研究上时，我们可能忘记以何种依据选择这些架构技术?选择标准是什么?领域驱动设计DDD 回答了这样的问题，DDD会告诉你如果一个框架不能协助你实现分层架构，那就抛弃它，同时，DDD也指出选择框架的考虑目的，使得你不会 人云亦云，陷入复杂的技术细节迷雾中，迷失了架构选择的根本方向。 现在也有些人误以为DDD是一种新的理论，其实DDD和设计模式一样，不是一种新的理论，而是实战经验的总结，它将前人 使用面向模型设计的方法经验提炼出来，供后来者学习，以便迅速找到驾驭我们软件项目的根本之道。 