架构设计原则(核心价值)

架构设计的原则（核心价值）

#一、简单原则：

我们应该尽量降低架构的复杂度，只为现实的（和合理的可预见的）需求提供支持，不要试图预先把所有的问题都考虑进去。但是，在力求简单的同时，有必要留意架构的设计质量，以保证未来能够对其进行重构，使其能够应对更加复杂的设计质量，以保证未来能够对其进行重构，使其能够应对更加复杂的需求。对架构的重构不像重构代码那么简单，但既然我们不希望面对新的需求时被迫修改大量代码，就必须重视架构的重构。即拥有一个具有伸缩性的简单架构是至关重要的。

在某些领域，J2EE架构师们常常过高地估计了需求的复杂程度，例如：数据库的分布和多种客户端。按照J2EE的正统思想，我们根本不允许客户有如此简单的需求。我们这些J2EE架构师凭自己的学识就知道：客户的业务早晚会变得复杂起来，那时我们提前叫客户掏钱购买的复杂架构就能派上用场了。

这种想法有两个问题：首先，是否让系统变得如此复杂不应该有作为架构师和开发者的我们来决定，因为买单的人不是我们：其次，即便系统最终变得如此复杂，我们又怎么知道一开始将它们考虑进来就能节约成本呢？说不定，等到有需求的时候再修改架构还会更节约呢。实际上，很可能永远不会出现这样的需求；即便他们真的出现了，我们也可以逢山开路遇水架桥。XP(eXtreme Programming,极限编程）的核心教义之一就是：很多时候，越是节约成本，就越能开发出高质量的软件；不要试图预先解决所有能想到的问题。

#二、生产率：

采用好的方案。复用原则，代码和方案复用，快速迭代，快速响应，持续优化。

#三、OO设计：

OO的设计比具体的技术（例如J2EE）要更加重要。我们应该尽量避免让技术上的选择（例如J2EE）妨碍我们使用真正的OO设计。

在J2EE中例如：

- 1、使用带有远程接口的EJB，以便分布业务对象。因为出于性能的考虑，带有远程接口的组件必须提供粗粒度的接口，以避免频繁的调用。另外，它们还需要以传输对象或者值对象的形式传递输入和输出参数。
- 2、认为持久对象不应该包含任何行为。因为entity bean在技术上的严重缺陷，而不是因为什么所谓的设计原则。仅仅暴露getter和setter（例如，通过getter和setter暴露持久化数据）的

> 对象不是真正的对象，一个真正的对象理应把针对自己状态的行为动作封装起来。即一旦你发现自己正在编写一个“不是真正对象”-----也就是说，只有一些用于暴露数据的方法-------的对象，你就应该想想自己为什么要这样做，是否有更好的选择。

#四、需求至上：

应用架构应该有业务需求来驱动，而不是以技术为目标，J2EE开发者常常凭空想出一些需求，如：

- 1、同时支持多个数据库
- 2、要求能够毫无成本地移植到别的应用服务器：
- 3、要求能够轻松地移植到别的数据库；
- 4、支持多种客户端。 这些都是潜在的业务需求，但它们不一定是真实的需求。需要具体情况具体分析。

#五、经验过程:

在设计、开发、测试、部署、演进 等方面要多和经验丰富的同事沟通讨论，同时设计和开发方案要经过团队的核心成员评审，保证效率和质量。尽量少采坑，节约时间。

#参考文章

- https://www.cnblogs.com/yelinpalace/archive/2009/05/31/1492576.html