Atitit 项目与产品模式架构与技术方案的区别与不同

项目模式和产品模式有很多不同，一般来说 项目进度为主，一般产品扩展性为主，大型产品可能要以性能为主。。不同的模式对应不同的模型

其次，使用场景也要考虑 ，一些内部项目，内部人员使用，根本无需考虑安全性。就好像我们只是外部安装一个防盗门，家里内部门都没有什么安全性。。如果考虑安全性，全部替换成防盗门，那就成本很高了。就算防盗门，也要考虑防盗级别，而不是全盘使用成本较高的金库安全门。

团队成员规模考虑。。大团队可以适当分层分块，小团队过度设计过度分层不可取，只会增加多余工作量。。

使用人数考虑，一般最多几万人使用的项目或产品，无需考虑性能问题。默认即可满足。。

这就要求我们进行架构与技术方案选型的时候，选择一个合适的模型。。

其次这里所说的项目和产品都是99%的普通项目。。大型互联网模式的的要根据实际情况调整。

|  |  |  |  |
| --- | --- | --- | --- |
| 常用的技术点 | 项目模式适用度 | 产品模式适应度 | 混合模式 |
| 前后台分离 | 适用 | 适用 | 适用 |
| H5 es6 | 推荐 | 推荐 |  |
| 客户端ui | 客户端ui | 客户端ui | 客户端ui |
| 简化中间架构 | 推荐 | 不适用 | 适当调节 |
| Sql | 大力推荐 | 适当使用 |  |
| sp存储过程 | 推荐 | 不适用 | 适当使用，会降低数据库移植性 |
| tigger触发器 | 推荐 | 不适用 | 适当使用 |
| Spring以及ioc | 不推荐 | 适用 | 适当使用 |
| Jdbc | 不推荐 | 不推荐 | 不推荐 |
| dbutil | 推荐 | 推荐 |  |
| Mybatis | 推荐 | 可能不适用 | 适当使用 |
| hibernate | 不推荐 | 可能不适用 | 适当使用 |
| view视图 | 大力使用 | 可能不适用 |  |
| 中文命名提升可读性 | 推荐 | 推荐 | 适当使用 |
|  |  |  |  |
|  |  |  |  |

## 前后台分离，通过接口对接ui和后端

这个值得全面推荐，大力提升几乎所有方面。。

## 界面设计静态html化

这个值得全面推荐，大力提升几乎所有方面。。

## H5 es6

值得全面推荐 ,提升进度 质量 安全等 chrome ff 360浏览器都全面支持

缺点：牺牲了一点界面兼容性。主要是落后和不安全的ie。

## Vue.js jq界面绑定数据框架

全面推荐

缺点：对爬虫友好性较差，不适合与那些需要seo的页面，那种可以使用服务端界面技术绑定数据。。

## 简化中间三层架构到俩层

全面推荐，主要看业务复杂度，简单的模块大力简化到双层架构 单层，复杂的模块就不简化。

提升整洁度 可读性 性能 。对其他不影响。

需要稍作分析模块复杂度，一般增删改查属于简单模块。有复杂流程的属于复杂模块。

## Sql和view视图

大力推荐..如有数据库移植要求的，要适当降低使用范围

## sp存储过程 和 trigger触发器

大力推荐...如有数据库移植要求的，则最好不使用，如有使用，限制使用范围。。

大力提升可读性 安全 性能 进度 ，牺牲数据库替换性。。

## Spring以及ioc

这个主要提升扩展性，对性能 进度，复杂度，都是负面影响

普通项目不推荐， 如确实要使用，也要限制使用范围从

## Jdbc 全面的不推荐 ，api繁琐

只有一个性能优势。。

## Dbutil 推荐

## Domain模型 vs Map动态模型

大型项目推荐Domain模型，普通项目推荐Map动态模型。。

## Mybatis

比较推荐。。有sql和映射俩种api。。

项目优先使用sql api

## Hibernate

最大优势，就是跨数据库，对一些产品有用。。项目基本一开始就明确数据库了，所以没啥作用。。

适当使用，注意使用范围，对性能和开发进度 复杂度都负面影响

提供sql hql javaapi三种api。。

数据库确定下，优先使用sql api。其次hql，最后3gl javaapi。如果确定要使用，限制使用范围度， 映射javaapi模式下最好不超过20%的范围使用，使用sql api 模式下可以不限制使用范围。。