# Java开发管理与加速

## 被测对象

肯定不是，一切都是相对的；哪些不需要测试呢：

* 你非常熟悉的功能；
* 一些简单的CRUD；
* 你认为不需要测试的；比如你很有把握的东西，就没有必要浪费时间测试了；

哪些需要测试呢：

* 复杂的业务逻辑/系统核心功能，最典型的如订单系统：一定要有足够的单元测试保证，这是一个电商系统的核心；还有如用户系统、积分系统等等；
* 框架级别/工具级别/通用级别的代码需要测试，即提供给第三方使用的代码，因为这些代码可能被很多系统依赖，应该保证其正确性；而且还要保证以后版本升级的向下兼容；
* 你认为需要测试的，比如你没有把握的东西，还是写点测试来缩短如开发web项目的重启系统的时间吧；

# 集成测试 CI

如集成测试时，测试执行起来可能比较慢，跑一遍测试可能需要5分钟，那怎么办呢？

* 每天下班前跑一遍集成测试，然后修复，下班走人；
* CI：持续集成，交给持续集成服务器，自动地测试完成后把测试报告以邮件的形式发到开发人员邮箱；

# Maven test

[**加速Java应用开发速度3——单元/集成测试+CI**](http://jinnianshilongnian.iteye.com/blog/1893135)

**博客分类：**

* [加速Java应用开发](http://jinnianshilongnian.iteye.com/category/283243)

[单元测试](http://www.iteye.com/blogs/tag/%E5%8D%95%E5%85%83%E6%B5%8B%E8%AF%95)[unit test](http://www.iteye.com/blogs/tag/unit%20test)

**《加速Java应用开发速度》系列目录：**

[加速Java应用开发速度1——加速spring/hibernate应用调试时启动速度](http://jinnianshilongnian.iteye.com/blog/1883013)

[加速Java应用开发速度2——加速项目调试启动速度](http://jinnianshilongnian.iteye.com/blog/1887788)

[加速Java应用开发速度3——单元/集成测试+CI](http://jinnianshilongnian.iteye.com/blog/1893135)

[加速Java应用开发速度4——使用模板技术加速项目开发速度](http://jinnianshilongnian.iteye.com/blog/1895481)

[加速Java应用开发速度5——使用脚本自动化日常操作](http://jinnianshilongnian.iteye.com/blog/1896755)

大家可能对如下情景比较熟悉：

* 如果开发过SSH的web项目，启动服务器可能会比较慢，有的项目甚至需要1分多钟，甚至更多，这个启动时间的等待一般就浪费了；
* 在开发项目时，有些功能比较复杂，当时觉得思路特清晰，但是过了一段时间后，自己也忘了，完善功能时频繁出现bug，降低开发速度；
* 在维护项目时，不知道自己修改的对还是不对，是否存在隐患；维护速度降下来了；
* 如果开发一个很多人都使用的接口，典型的如用户系统，要保证比如升级时向下兼容；
* 在团队间协作时，有时候只定义好接口，对方还没有给实现，如何进行同步开发？

如上问题，估计只要是个开发人员，都可能遇到过；如果此时有了单元/集成测试，那我们能很好的解决这些问题。（注：加下来如果没有特殊情况，不刻意强调 单元测试/集成测试，即提到测试是指的是单元/集成测试）

我从以下几个方面介绍测试：

**1、为什么需要测试？**

**2、如何进行测试？**

**3、测试有哪些好处？**

**4、一切都需要测试吗？**

**1、为什么需要测试？**

测试的目的是什么？我的理解是：

* 缩短发现问题到解决问题的速度；
* 给程序一个修改后能验证是否正确的保证；（回归测试）
* 如果是开源软件，我们可以通过单元测试了解其是怎么使用的；比如我之前通过cglib的单元测试学习过cglib的使用；

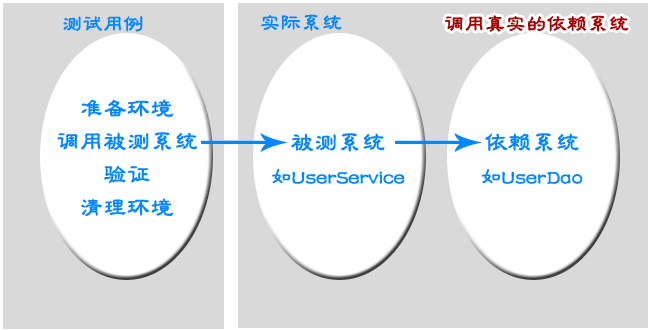
所以如果你遇到如上问题，就需要写测试。写测试可能是为了自己（1、2）；也可能是为了帮助别人（3）。

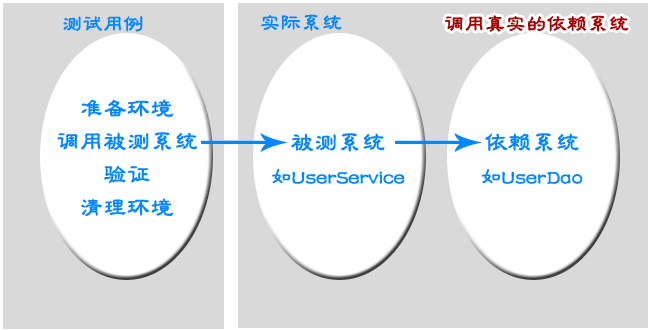
**2、如何进行测试？**

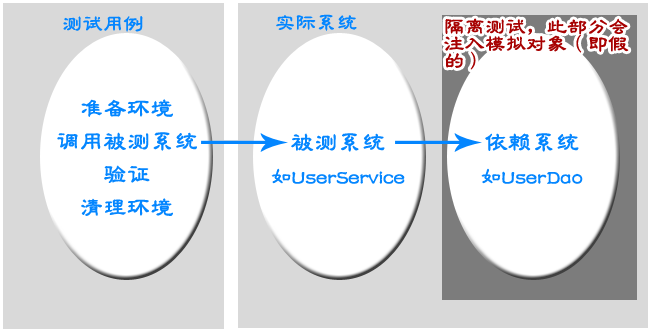
很多朋友不知道如何进行测试，其实测试很简单，别把它想复杂了，按照自己的想法测试每个功能点是否正确即可。

**2.1、测试流程**

单元测试流程



集成测试流程  


集成测试流程   


可以看出，单元测试与集成测试唯一不同点是一个调用依赖系统而一个不调用；因为单元测试是最小粒度的测试，如在Java中是测试一个类，不会测试依赖系统；而集成测试是会测试依赖系统的。

**测试的步骤：**

1. 准备环境
2. 调用被测系统
3. 验证
4. 清理环境

**环境**：也叫做夹具（fixture）或者固件，表示调用被测系统时需要准备/清理的数据等等；

**被测系统**：在Java中就是要测试的类，如UserService；

**依赖系统**：测试被测系统时，其依赖的部分，如UserDao；

**测试用例**：包含测试方法的类，里边有很多测试方法来测试被测系统。

接下来仔细看看各部分都做了哪些工作。

**2.2、环境**

 环境，也叫做夹具（fixture），表示调用被测系统时需要准备/清理的数据等等；保证测试时环境是干净的，如不被之前的数据库数据影响；保证每次测试都是在干净/新鲜的环境中执行的。所谓干净的环境表示如当前测试不被之前测试插入/删除/修改的数据造成影响。在junit中可以使用：

* @Before（setUp） 安装夹具或准备环境：在测试用例的每个测试方法之前执行；比如创建新鲜的被测系统，单元测试时安装Mock的依赖系统；
* @After（tearDown）卸载夹具或清理环境：在测试用例的每个测试方法之后执行；比如数据库测试时回滚事务，删除数据；关闭文件；
* @BeforeClass：在整个测试用例之前执行；
* @AfterClass：在整个测试用例之后执行；

使用如上方法，而不是直接在测试方法中安装/卸载；是因为不管有没有异常，@After/@AfterClass都会执行，这样防止出现异常可能造成环境是不新鲜的问题。

如果大家使用spring test来测试数据库相关的系统，可以考虑使用@TransactionConfiguration来支持默认事务回滚，这样不会对现有系统造成影响。具体可参考《[【第十三章】 测试 之 13.1 概述 13.2 单元测试 ——跟我学spring3](http://jinnianshilongnian.iteye.com/blog/1465582" \o "【第十三章】 测试 之 13.1 概述 13.2 单元测试 ——跟我学spring3" \t "_blank)》和《[【第十三章】 测试 之 13.3 集成测试 ——跟我学spring3](http://jinnianshilongnian.iteye.com/blog/1469524" \o "【第十三章】 测试 之 13.3 集成测试 ——跟我学spring3" \t "_blank)》

**测试时一定要保证环境是干净/新鲜的，才能保证每次测试的结果是一样的。**

**2.3、被测系统与依赖系统**

被测系统：在Java中就是被测试的Java类。

依赖系统：就是被测试Java类依赖的其他类。

如果是单元测试，一般情况下，会对依赖系统进行模拟（Mock），即给它一个假的实现；典型的如测试服务层时注入一个Mock的DAO层，这样的好处：

* 加快测试速度；因为不会调用真实的被测系统，所以速度特别快；
* 测试还没有完成的功能；尤其在多团队协作时，可以只在定义好接口的情况下开发系统；

如果是集成测试时，直接注入真实的依赖系统即可，好处：

* 完成联调；
* 发现自己的问题；
* 还可能发现自己使用上问题及使用的API的问题；

单元测试虽然好，但是是隔离测试，即不会调用被测系统来完成测试，因为不是真实的联调，所以很可能会潜在有一些问题，因此还是需要集成测试。（所以不是很刻意分单元或集成测试，且有些系统可能只有集成测试）

但是集成测试速度是比较慢的，一般提交给CI执行，不影响当前开发进度。

**2.4、验证**

验证的目的：是保证实际结果和我们预期的结果是否一致，说白了就是是否是我们想的那样。

一般使用断言来验证，如：

Assert.assertEquals(expectedResult, actualResult); //验证预期结果和实际结果是否相等

验证主要有两种：

* 结果验证
* 行为验证

结果验证：即验证被测系统返回的结果是否正确，如：

**Java代码  [收藏代码](javascript:void())**

1. @Test
2. **public** **void** testCount() {
3. String ql = "select count(o) from User o";
4. **long** expectedCount = repositoryHelper.count(ql) + 1;
6. User user = createUser();
7. repositoryHelper.getEntityManager().persist(user);
9. **long** acutalCount = repositoryHelper.count(ql);
10. Assert.assertEquals(expectedCount, acutalCount);
12. }

验证返回的数据总数 = 插入之前的总数 + 1； 即结果验证。此处我们使用了一种叫做相对（delta）测试；即不关心数据库里到底多少条，只关心实际的和预期的差。

行为验证：即验证被测系统是否调用了依赖系统的某个API ，这个只有当我们使用Mock时测试时比较简单，如当用户注册时：

1、加积分

2、发系统消息

3、……

此时我们并不能通过结果验证是否调用了这些方法；那么我们可以使用Mock技术来完成验证是否调用了这些API，比如使用jmock测试框架就支持行为验证。集成测试是很难进行行为验证的，如果测试需要预留间谍接口。

**3、测试有哪些好处？**

我们写代码的目的是正确的完成某个功能，如何保证正确呢？测试！所以在不使用如单元测试技术时，我们也是需要测试，但是这个测试是我们人工验证的。缺点很明显：

* 不是自动的，每次需要对比预期结果与实际结果，尤其数据量/逻辑复杂时更痛苦；
* 不是回归的，上次测试完成后，下次还得重复自己一遍；

为了解决这个问题，我们使用如单元测试技术来解决这个问题：

* 测试自动化；即验证预期结果与实际结果交给计算机吧；
* 测试回归性，可以重复执行测试，验证修改后逻辑是否还是正确的；

即测试的好处，从如上已经提炼出来了：

* 缩短发现问题到解决问题的时间；
* 重复使用测试，保证修改后的代码还是正确的；
* 如果做开源项目，可以提供给使用人员参考如何使用；
* 因为单元测试都非常快，所以提升了开发速度；

**4、一切都需要测试吗？**

肯定不是，一切都是相对的；哪些不需要测试呢：

* 你非常熟悉的功能；
* 一些简单的CRUD；
* 你认为不需要测试的；比如你很有把握的东西，就没有必要浪费时间测试了；

哪些需要测试呢：

* 复杂的业务逻辑/系统核心功能，最典型的如订单系统：一定要有足够的单元测试保证，这是一个电商系统的核心；还有如用户系统、积分系统等等；
* 框架级别/工具级别/通用级别的代码需要测试，即提供给第三方使用的代码，因为这些代码可能被很多系统依赖，应该保证其正确性；而且还要保证以后版本升级的向下兼容；
* 你认为需要测试的，比如你没有把握的东西，还是写点测试来缩短如开发web项目的重启系统的时间吧；

 测试不是不耗时间的，没意义的测试就是浪费时间，最典型是一些书上的对一个增删改查进行测试，实际项目没有任何意义。所以你应该只对自己很难驾驭的觉得有必要的代码进行测试。不要成为一个测试狂，什么都测试。

一些测试可以参考我的《[es——JavaEE快速开发脚手架](https://github.com/zhangkaitao/es" \t "_blank)》中的代码。通过测试我得到了许多好处。

到此我们介绍完成了测试，但是如果我们使用了如集成测试时，测试执行起来可能比较慢，跑一遍测试可能需要5分钟，那怎么办呢？

* 每天下班前跑一遍集成测试，然后修复，下班走人；
* CI：持续集成，交给持续集成服务器，自动地测试完成后把测试报告以邮件的形式发到开发人员邮箱；

------------------------------------分割线----------------------------------

接下来介绍一下CI吧。

**1、为什么需要CI**

**2、CI如何工作的**

**3、travis-ci介绍**

**1、为什么需要CI**

正如前边说的，我们单独测试可能会遇到如下问题：

* 如果写了一个测试，就要把所有测试跑一遍看看整个系统是否是正确的，那么每次等待时间是非常漫长的；
* 如果团队中的其他成员改了功能并提交了，如何快速得到该次提交对当前系统代码是正确还是失败的反馈；

那怎么办呢？自动化地持续集成（CI）！CI的核心就是干这件事情的。自动化持续地集成测试。

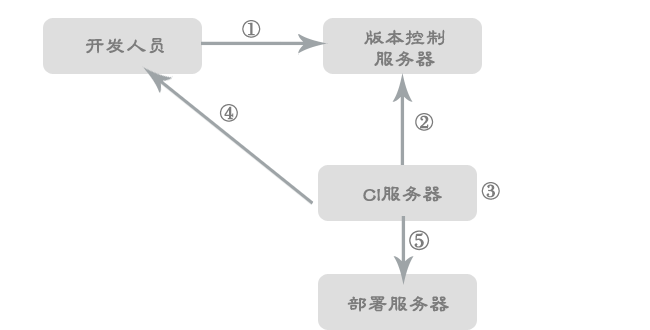
使用CI后，如果使用Maven，可以新建多个profile：

* 本地测试时忽略一些比较慢的测试；
* CI服务器上执行所有测试；

**2、CI如何工作的**

一个典型的持续集成流程：

1. 定期检测版本服务器上是否有代码更新；
2. 如果发现代码更新，从版本服务器下载最新的代码；
3. 自动构建并自动化的测试；
4. 不管错误/失败，生成报告给开发人员；
5. 有些CI服务器还能产生可执行的软件，自动化地部署到测试机器，交给测试人员测试。

如图所示：  
  
 

持续集成服务器其实就是一个定时器，自动帮你下载最新代码、编译、测试、集成及产生报告发给开发人员。

常见的CI服务器有：

* Apache Continuum
* Hudson
* CruiseControl
* Jenkins CI
* TeamCity
* Travis CI

我09年时使用过TeamCity社区版，足够满足常见需求；目前我使用github托管项目，使用Travis CI进行分布式的持续集成，免费，目前看来还是不错的。

**3、travis-ci介绍**

我现在开发的[ES-JavaEE项目开发脚手架](https://github.com/zhangkaitao/es)就是使用travis ci进行持续集成；具体参考《[Getting started](http://about.travis-ci.org/docs/user/getting-started/" \t "_blank)》进行与Github集成，其支持的语言：

* [C](http://about.travis-ci.org/docs/user/languages/c)
* [C++](http://about.travis-ci.org/docs/user/languages/cpp)
* [Clojure](http://about.travis-ci.org/docs/user/languages/clojure)
* [Erlang](http://about.travis-ci.org/docs/user/languages/erlang)
* [Go](http://about.travis-ci.org/docs/user/languages/go)
* [Groovy](http://about.travis-ci.org/docs/user/languages/groovy)
* [Haskell](http://about.travis-ci.org/docs/user/languages/haskell)
* [Java](http://about.travis-ci.org/docs/user/languages/java)
* [JavaScript (with Node.js)](http://about.travis-ci.org/docs/user/languages/javascript-with-nodejs)
* [Objective-C](http://about.travis-ci.org/docs/user/languages/objective-c)
* [Perl](http://about.travis-ci.org/docs/user/languages/perl)
* [PHP](http://about.travis-ci.org/docs/user/languages/php)
* [Python](http://about.travis-ci.org/docs/user/languages/python)
* [Ruby](http://about.travis-ci.org/docs/user/languages/ruby)
* [Scala](http://about.travis-ci.org/docs/user/languages/scala)

支持的数据库：

* MySQL
* PostgreSQL
* MongoDB
* CouchDB
* Redis
* Riak
* RabbitMQ
* Memcached
* Cassandra
* Neo4J
* ElasticSearch
* Kestrel
* SQLite3

[更多请参考其官网的介绍。](http://about.travis-ci.org/docs/user/database-setup/)

如果是Java开发人员，支持的JDK包括：OpenJDK 和 OracleJDK。 如果使用的是OpenJDK，Maven中使用ascii2native插件时，需要如下配置：

**Java代码  [收藏代码](javascript:void())**

1. <plugin>
2. <groupId>org.codehaus.mojo</groupId>
3. <artifactId>native2ascii-maven-plugin</artifactId>
4. <version>1.0-alpha-1</version>
5. <executions>
6. <execution>
7. <phase>generate-resources</phase>
8. <goals>
9. <goal>native2ascii</goal>
10. </goals>
11. <configuration>
12. <encoding>UTF-8</encoding>
13. <src>src/main/messages</src>
14. <dest>target/${project.artifactId}/WEB-INF/classes</dest>
15. <includes>messages.properties</includes>
16. </configuration>
17. </execution>
18. </executions>
19. <!-- native2ascii 使用的tools.jar -->
20. <dependencies>
21. <dependency>
22. <groupId>com.sun</groupId>
23. <artifactId>tools</artifactId>
24. <version>1.7.0</version>
25. <scope>system</scope>
26. <systemPath>${java.home}/../lib/tools.jar</systemPath>
27. </dependency>
28. </dependencies>
29. </plugin>

如果使用mysql，端口只能是3306。

如果想开端口测试，这是不允许的。

如下是我项目中的一个配置[.travis.yml](https://github.com/zhangkaitao/es/blob/master/.travis.yml" \t "_blank)，放到项目的根下即可：

-----------------------------------

language: java           语言

env:                           环境

  - DB=mysql              使用mysql

jdk:

  - openjdk                jdk使用openjdk

mysql:

  database: es         数据库名为es

  username: root     用户名为root

  password :            密码为空

  encoding: utf8      编码为utf8

install:                     安装时执行的脚本

  - mvn install -Dmaven.test.skip=true     mvn安装并跳过测试

before\_script:        script之前执行的测试

  - cd web

  - mvn db:create  创建数据库的mvn命令（此处使用了 maven-db-plugin 插件）

  - mvn db:schema  创建脚本的mvn命令

  - mvn db:data        安装数据的mvn命令

  - cd ..

script:                      测试时执行的脚步

  - cd common

  - mvn test              测试common子模块

  - cd ..

  - cd web

  - mvn test -Pit       测试web子模块，并指定使用it profile测试（即集成测试的配置，具体参考[pom.xml](https://github.com/zhangkaitao/es/blob/master/web/pom.xml" \t "_blank)中的profile/it）

notifications:          触发

  email:                  测试完成后测试报告发到哪

    - zhangkaitao0503@gmail.com

-----------------------------------

**持续集成不能修复代码的错误，而是和单元测试一样，缩短发现问题带解决问题的时间，这样可以提高开发效率，降低项目风险，提高项目的稳定性。而且尤其是团队协作时，可以发现其他人的代码是否对自己的代码产生影响。**

到此我们利用单元测试+CI可以加速开发人员的开发速度。利用好单元测试和CI，不要纯粹为了单元测试和CI而去做这些事情。

本文没有介绍TDD，TDD并不会那么美好，我认为我们可以借鉴TDD的一些思想，但决不能迷信TDD，有时候，尤其如开发企业应用，先写功能再写测试可能效率更高，而且大部分时候是不需要TDD的。而且我也没能在实际项目中获取太多TDD的好处，但是我获得了测试的好处。

本文也没有介绍测试覆盖率，我认为不要一味的追求覆盖率，有时候有的覆盖率没有任何意义。所以不要让为了覆盖率而覆盖率拖慢了项目开发进度。

正如stackoverflow上的一篇帖子《[How deep are your unit tests?](http://stackoverflow.com/questions/153234/how-deep-are-your-unit-tests)》上[Kent Beck](http://en.wikipedia.org/wiki/Kent_Beck" \t "_blank)的回答：

**写道**

老板为我的代码付报酬，而不是测试，所以，我对此的价值观是——测试越少越好，少到你对你的代码质量达到了某种自信。

可以前往[coolshell](http://coolshell.cn/)的[“单元测试要做多细？”](http://coolshell.cn/articles/8209.html" \t "_self)去得到一些经验。

# 事务

# Bank accout 8

# 电商平台事务

# 金融业务中的事务

# 海量数据与性能

# spring multi datasource

# IoC, AOP

# Thread

# Package of your using daily

# 脚本化

如果是一些需要固定周期执行的命令/脚本，可以使用定时调度，如linux的crontab定时调度，如典型的数据库备份、清理垃圾文件、压缩及删除过期的日志等等。

比如maven中jar包的安装：mvn install:install-file -DgroupId=\* -DartifactId=\* -Dversion=\* -Dpackaging=jar -Dfile=\*

# 高速编辑代码

，作用环境是java，模板是<嘿嘿，你在干嘛？>。也就是说你在打出了<aa>然后按 [ALT+/]键则会加载你写的这个模板。如图所示：

# MyGenertate

# 中国人的jeecg

# Inteli IDEA

# Eclipse模板生成器

# Hibernate, code Smith

# <http://git.oschina.net/jeecg/jeewx>

# Groovy模板引擎

### [slave4j2.0.0版本说明](http://slave4j.iteye.com/blog/716327)

# Freemarker

# 技术维度

# 第一阶段：Java基础，包括java语法，面向对象特征，常见API，集合框架；

# \*第二阶段：java界面编程，包括AWT，事件机制，SWING，这个部分也可以跳过，用的时候再看都能来及；

# \*第三阶段：java API：输入输出，多线程，网络编程，反射注解等，java的精华部分；\*第四阶段：数据库SQL基础，包括增删改查操作以及多表查询；

# \*第五阶段：JDBC编程：包括JDBC原理，JDBC连接库，JDBC API，虽然现在Hibernate比JDBC要方便许多，但是JDBC技术仍然在使用，JDBC思想尤为重要；\*第六阶段：JDBC深入理解高级特性：包括数据库连接池，存储过程，触发器，CRM思想；

# \*第七阶段：HTML语言学习，包括HTML标签，表单标签以及CSS，这是Web应用开发的基础；

# \*第八阶段：JavaScript脚本语言，包括javaScript语法和对象，就这两个方面的内容；

# \*第九阶段：DOM编程，包括DOM原理，常用的DOM元素以及比较重要的DOM编程思想；

# \*第十阶段：Servlet开发，从此开始踏入java开发的重要一步，包括XML，Tomcat服务器的安装使用操作，HTTP协议简单理解，Servlet API等，这个是java web开发的基础;

# \*第十一阶段：JSP开发：JSP语法和标签，自定义标签，EL,JSTL库了解以及MVC三层架构的设计模式理念；

# \*第十二阶段：AJAX开发：AJAX原理，请求响应处理，AJAX开发库；

# \*第十三阶段：轻量级框架，三大框架之一Struts框架的学习，自此踏入java web开发的精华部分，包括Struts体系架构，各种组件，标签库和扩展性的学习；

# \*第十四阶段：Hibernate框架学习，三大框架之一，包括检索映射技术，多表查询技术，缓存技术以及性能方面的优化；

# \*第十五阶段：Spring框架的学习，三大框架之一，包括了IOC,AOP,DataSource，事务，SSH集成以及JPA集成；

第十16阶段

第十17阶段

第十18阶段

eclipse插件

http://download.jboss.org/jbosstools/updates/stable/juno/

# 只有部分页面需要静态化

# 页面静态化，通常有两种方式。一种是整个页面静态化为html文件；另一种是整个页面还是动态文件，只是页面的部分栏目静态化为一个模块，然后在页面中进行include

# 技术维度

 第一阶段：Java基础，包括java语法，面向对象特征，常见API，集合框架；

 \*第二阶段：java界面编程，包括AWT，事件机制，SWING，这个部分也可以跳过，用的时候再看都能来及；

\*第三阶段：java API：输入输出，多线程，网络编程，反射注解等，java的精华部分；\*第四阶段：数据库SQL基础，包括增删改查操作以及多表查询；

\*第五阶段：JDBC编程：包括JDBC原理，JDBC连接库，JDBC API，虽然现在Hibernate比JDBC要方便许多，但是JDBC技术仍然在使用，JDBC思想尤为重要；\*第六阶段：JDBC深入理解高级特性：包括数据库连接池，存储过程，触发器，CRM思想；

\*第七阶段：HTML语言学习，包括HTML标签，表单标签以及CSS，这是Web应用开发的基础；

\*第八阶段：JavaScript脚本语言，包括javaScript语法和对象，就这两个方面的内容；

\*第九阶段：DOM编程，包括DOM原理，常用的DOM元素以及比较重要的DOM编程思想；

\*第十阶段：Servlet开发，从此开始踏入java开发的重要一步，包括XML，Tomcat服务器的安装使用操作，HTTP协议简单理解，Servlet API等，这个是java web开发的基础;

\*第十一阶段：JSP开发：JSP语法和标签，自定义标签，EL,JSTL库了解以及MVC三层架构的设计模式理念；

\*第十二阶段：AJAX开发：AJAX原理，请求响应处理，AJAX开发库；

 \*第十三阶段：轻量级框架，三大框架之一Struts框架的学习，自此踏入java web开发的精华部分，包括Struts体系架构，各种组件，标签库和扩展性的学习；

\*第十四阶段：Hibernate框架学习，三大框架之一，包括检索映射技术，多表查询技术，缓存技术以及性能方面的优化；

\*第十五阶段：Spring框架的学习，三大框架之一，包括了IOC,AOP,DataSource，事务，SSH集成以及JPA集成；

# 架构维度

# 

Int

String

Date

Image

Bin

Float

Id

Sequence

Name

Text

BusiName

BusiValue

bTime

eTime

Log

Transaction

# Bean.xml

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | @Test  **public void** testEnv() {  ApplicationContext ctx = **new**  ClassPathXmlApplicationContext("beans.xml");  } | beans.xml 可以在类路径下进行配置，也可以在具体的目录下配置。可以是一个配置文件，也可以是多个配置文件组成String 数组传入。 | Classpath: |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | dao |  |
| 1 |  |  |  |
| 2 |  | **package**com.asm.dao;  **public interface** UserDao {  **void**save();  } |  |
| 3 |  | **package**com.asm.dao.impl;  **import**com.asm.dao.UserDao;  **public class** UserDaoImpl **implements** UserDao{  **public void** save() {  System.*out*.println("执行save方法...");  }  } | In bean.xml |
| 4 |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 图化分析

# 