Разработка на JAVA

*Документ создан: 20.08.2009*

*Последняя редакция: 06.07.2017*

***При написании данного опуса ни одно животное, кроме автора, не пострадало!***

ОГЛАВЛЕНИЕ

[Часть I. Общие вопросы. 5](#_Toc487305154)

[Глава 1. Немного общих слов и лирики. 5](#_Toc487305155)

[О чем же данное пособие? (зачем тратить на чтение этого опуса свое время) 5](#_Toc487305156)

[Для кого предназначен данный документ? Может для тебя? ☺ 5](#_Toc487305157)

[Необходимые знания, для освоения данного пособия. 5](#_Toc487305158)

[Использованное ПО 6](#_Toc487305159)

[Глава 2. Общие вопросы разработки ПО. 7](#_Toc487305160)

[«Золотые правила программирования» 7](#_Toc487305161)

[«Тест Джоэла» 12](#_Toc487305162)

[Объектно-ориентированное vs функциональное программирование 12](#_Toc487305163)

[Глава 3. Организация работы. 13](#_Toc487305164)

[Часть II. Непосредственно JAVA. 16](#_Toc487305165)

[Глава 4. Начало 16](#_Toc487305166)

[Инсталляция Java. 16](#_Toc487305167)

[Инструменты JDK. 16](#_Toc487305168)

[Объектно-ориентированное программирование. 16](#_Toc487305169)

[Глава 5. Основные типы данных Java. 16](#_Toc487305170)

[Примитивные типы данных 16](#_Toc487305171)

[Типы «оболочки» для примитивных типов данных 19](#_Toc487305172)

[Глава 6. Основные конструкции языка. 19](#_Toc487305173)

[Глава 7. Инкапсуляция и полиморфизм. Абстрактные классы и интерфейсы. 19](#_Toc487305174)

[Глава 8. Некоторые интересные классы java. Краткий обзор. 20](#_Toc487305175)

[Базовый класс Object. 20](#_Toc487305176)

[Класс String. 20](#_Toc487305177)

[Глава 9. GUI. 20](#_Toc487305178)

[Менеджеры компоновки (layout managers). 21](#_Toc487305179)

[Обработка событий в интерфейсе. 21](#_Toc487305180)

[Глава 10. Исключительные ситуации. 21](#_Toc487305181)

[Глава 11. Java Collections Framework 22](#_Toc487305182)

[Глава 12. Generics 23](#_Toc487305183)

[Глава 13. Система ввода-вывода, потоки ввода/вывода 23](#_Toc487305184)

[Интерфейсы 23](#_Toc487305185)

[Потоки ввода байтовые 23](#_Toc487305186)

[Потоки вывода байтовые 23](#_Toc487305187)

[Потоки ввода символьные 24](#_Toc487305188)

[Потоки вывода символьные 24](#_Toc487305189)

[Глава 14. Сериализация 24](#_Toc487305190)

[Глава 15. NIO 24](#_Toc487305191)

[Глава 16. Сетевое программирование. 24](#_Toc487305192)

[Глава 17. Многопоточность. 24](#_Toc487305193)

[Глава 18. Работа с БД (JDBC) 25](#_Toc487305194)

[Глава 19. Java and functional programming. 25](#_Toc487305195)

[Глава 20. Настройки и опции JVM 25](#_Toc487305196)

[Глава 19. Common libraries/frameworks/tools. 25](#_Toc487305197)

[Глава 20. Интересные задачки/Java puzzlers 26](#_Toc487305198)

[Глава 21. Java best practices. 29](#_Toc487305199)

[Глава 22. JEE best practices. 31](#_Toc487305200)

[Часть III. Инфраструктура. Continuos integration/delivery. 32](#_Toc487305201)

[Глава 23. Интегрированные среды разработки (IDE) 32](#_Toc487305202)

[IntelliJIDEA 32](#_Toc487305203)

[Eclipse 32](#_Toc487305204)

[NetBeans (6.9.1). 32](#_Toc487305205)

[Глава 24. Средства автоматизации сборки. 33](#_Toc487305206)

[Apache Ant. 33](#_Toc487305207)

[Установка и конфигурирование 33](#_Toc487305208)

[Apache Maven. 33](#_Toc487305209)

[Установка и конфигурирование 34](#_Toc487305210)

[Работа с проектом 34](#_Toc487305211)

[Gradle 35](#_Toc487305212)

[Глава 25. Системы отслеживания ошибок (bug trackers) 35](#_Toc487305213)

[JTrac. 36](#_Toc487305214)

[JIRA 37](#_Toc487305215)

[BugZilla 37](#_Toc487305216)

[Глава 26. Системы постоянной интеграции (continuous integration) 37](#_Toc487305217)

[Hudson 37](#_Toc487305218)

[Часть IV. Enterprise world (Java EE/DBMS/etc.). 37](#_Toc487305219)

[Глава 27. CDI 37](#_Toc487305220)

[Глава 28. «Легкие» JEE контейнеры (Tomcat/Jetty/Resin). 38](#_Toc487305221)

[Tomcat 38](#_Toc487305222)

[Jetty 38](#_Toc487305223)

[Resin 38](#_Toc487305224)

[Глава 29. Промышленный JEE контейнер WebLogic. 38](#_Toc487305225)

[Основные понятия 38](#_Toc487305226)

[1. Где и что находится (скрипты и прочее) 39](#_Toc487305227)

[2. Установка сервера WL 39](#_Toc487305228)

[Структура каталогов 39](#_Toc487305229)

[Установка 40](#_Toc487305230)

[Установка в графическом режиме. 40](#_Toc487305231)

[Установка в консольном режиме, установка в «тихом» режиме. 40](#_Toc487305232)

[Установка переменных окружения 40](#_Toc487305233)

[3. Создание/конфигурирование домена 41](#_Toc487305234)

[Вопросы, на которые необходим ответ в процессе настройки домена 41](#_Toc487305235)

[Глава 30. СУБД ORACLE 42](#_Toc487305236)

[Утилиты для работы с СУБД 42](#_Toc487305237)

[Sql\*plus 42](#_Toc487305238)

[Instant client 43](#_Toc487305239)

[Full client 43](#_Toc487305240)

[Sqldeveloper 43](#_Toc487305241)

[Утилиты импорта/экспорта баз данных (imp/exp или impdp/expdp) 43](#_Toc487305242)

[FAQ по утилитам Оракл 43](#_Toc487305243)

[Глава 31. СУБД MySql 44](#_Toc487305244)

[Установка 44](#_Toc487305245)

[Пользователи 44](#_Toc487305246)

[Установка с помощью GUI инсталлятора. 45](#_Toc487305247)

[Установка из командной строки. 45](#_Toc487305248)

[Перенос БД (import/export) 45](#_Toc487305249)

[Глава 32. СУБД PostgreSql. 45](#_Toc487305250)

[Установка 45](#_Toc487305251)

[Настройка 45](#_Toc487305252)

[Глава XX. Статический анализатор кода Sonar 46](#_Toc487305253)

[Архитектура Сонара 46](#_Toc487305254)

[Установка сервера Сонар 46](#_Toc487305255)

[Анализ кода 47](#_Toc487305256)

[Анализ кода с помощью Sonar-runner (java) 48](#_Toc487305257)

[Анализ кода с помощью Ant 48](#_Toc487305258)

[Анализ кода с помощью Maven 48](#_Toc487305259)

[Глава 33. Тестирование программ. JUnit 4.x. 89](#_Toc487305260)

[Добавление JUnit 4 в проект. 90](#_Toc487305261)

[Небольшой FAQ/рекомендации. 91](#_Toc487305262)

[Глава 34. Фреймворк Spring 3/4. 92](#_Toc487305263)

[Spring для веб-разработки. 92](#_Toc487305264)

[Глава 35. Spring Security 3. 93](#_Toc487305265)

[Глава 36. Мониторинг приложений/контейнеров Java. 94](#_Toc487305266)

[VisualVM. 94](#_Toc487305267)

[JConsole. 94](#_Toc487305268)

[Полезные ссылки и ресурсы. 94](#_Toc487305269)

[Часть V. Мир \*nix 95](#_Toc487305270)

[Глава 37. Работа с UNIX-like системами. Основные принципы (на примере Ubuntu Server 12.04/14.04/16.04 LTS) и практики. 95](#_Toc487305271)

[Устройство файловой системы 95](#_Toc487305272)

[Работа с командной строкой 95](#_Toc487305273)

[Копирование файлов по протоколу ssh. 95](#_Toc487305274)

[Настройка сети 96](#_Toc487305275)

[Настройка сети для сервера Ubuntu установленного на Virtual Box. 96](#_Toc487305276)

[Работа с текстом 96](#_Toc487305277)

[Редактор vi/vim 96](#_Toc487305278)

[Редактор nano 97](#_Toc487305279)

[Установка сервера ssh. 97](#_Toc487305280)

[Работа с пользователями и группами 97](#_Toc487305281)

[Системные команды 98](#_Toc487305282)

[Работа с переменными окружения 99](#_Toc487305283)

[Некоторые команды для работы с переменными окружения 99](#_Toc487305284)

[Установка ПО (общие сведения). 99](#_Toc487305285)

[Обновление имеющегося ПО 100](#_Toc487305286)

[Установка нового ПО, удаление установленного ПО 100](#_Toc487305287)

[Настройка apt для использования прокси-сервера 100](#_Toc487305288)

[Автозагрузка сервисов (процессов) при старте системы 100](#_Toc487305289)

[Помещение сервиса в автозагрузку 101](#_Toc487305290)

[Удаление сервиса из автозагрузки 101](#_Toc487305291)

[Файловый менеджер MC (Midnight Commander) 101](#_Toc487305292)

[Установка и настройка сервера subversion (с утилитами командной строки). 101](#_Toc487305293)

[LAMP 103](#_Toc487305294)

[Установка mysql (используем версию 5) 104](#_Toc487305295)

[Установка веб-сервера apache2 105](#_Toc487305296)

[Установка PHP5 106](#_Toc487305297)

[Работа с БД – phpmyadmin 106](#_Toc487305298)

[Установка Perl 107](#_Toc487305299)

[Установка системы трекинга ошибок (BugZilla, версия 4.x.x). 108](#_Toc487305300)

[Установка wiki-движка (MediaWiki) 110](#_Toc487305301)

[Установка java (JDK). 111](#_Toc487305302)

[Установка Apache Ant 112](#_Toc487305303)

[Установка Apache Maven 113](#_Toc487305304)

[Установка Ruby (+RVM (Ruby Version Manager), +Rails). 113](#_Toc487305305)

[Установка Redmine (для версии 3.1.1) 113](#_Toc487305306)

[Установка Tomcat 6 116](#_Toc487305307)

[Установка GlassFish 117](#_Toc487305308)

[Часть VI. Big Data. 120](#_Toc487305309)

[Глава 38. Hadoop. 120](#_Toc487305310)

[Часть VII. Management stuff ☺ 121](#_Toc487305311)

[Глава 39. О переговорах. 121](#_Toc487305312)

# Часть I. Общие вопросы.

## Глава 1. Немного общих слов и лирики.

### О чем же данное пособие? (зачем тратить на чтение этого опуса свое время)

Если кратко – данное пособие о технологии JAVA и сопутствующих ей механизмах, которые являются частями других технологий, о том как стоит и как НЕ стоит программировать на JAVA, о различных библиотеках/фреймворках/инструментах для работы с JAVA, немного коснемся и вопросов, связанных с базами данных. Также по ходу изложения даются объяснения/советы общего характера.

В документе приводится описание не всей технологии JAVA (и не полностью описание/раскрытие остальных упомянутых тем), а некоторой части каждой темы/технологии/механизма, которая была использована автором в своей повседневной работе. Т.е. данное пособие не претендует на роль «ответа на все вопросы» - документацию/форум/поиск в интернете еще никто не отменял.

Пособие следует рассматривать как справочник по различным аспектам разработки клиентских и клиент-серверных приложений с применением технологии JAVA.

Надеюсь, прочтение сего опуса сподвигнет читателя к дальнейшему самосовершенствованию в данной области и в смежных областях IT.

### Для кого предназначен данный документ? Может для тебя? ☺

Документ носит справочный характер и предназначен для того, кто хочет лучше понять/изучить некоторые аспекты технологии Java и сопутствующих частей других технологий. Документ не претендует на охват всей технологии Java или полный охват других технологий – в документе описываются только те аспекты/части технологий, с которыми работал или сталкивался автор.

Поэтому рекомендую данный документ тем, кто хочет расширить свои познания в технологии Java и некоторых сопутствующих.

На практике, данный документ можно применить при организации командной разработки (раздел с рекомендациями), при настройке рабочего окружения Java-разработчика, при настройке тестовых/рабочих серверов Tomcat – как standalone, так и в связке с веб-сервером IIS. Также документ буден полезен Java-разработчикам для расширения своего кругозора в технологии и ответа на возникающие вопросы (а вдруг будет ответ именно на твой вопрос?).

### Необходимые знания, для освоения данного пособия.

Пособие не предназначено для совсем новичков в IT. Если вы делаете только первые шаги в сторону освоения ПК – не надо даже и заглядывать сюда. Вот примерный перечень того, что должен знать/уметь читатель пособия:

* включить/выключить компьютер (если вы читаете это в электронном виде – значит умеете хотя бы включать комп… хотя… ☺)
* минимальный уровень – продвинутый пользователь ПК. Подразумевается знакомство на «ты» с ОС семейства Windows (NT/2000/XP/Vista/7)
* понимание работы batch-файлов и навыки их написания и использования. Для чего это нужно? Часть работ по установке/настройке рабочего окружения легко выполняется (автоматизируется) с помощью bat-файлов и этот подход применяется автором там, где возможно.
* для освоения одной части материала необходимо иметь навыки сисадмина (напр., для освоения 5-ой главы - сервер Tomcat), для освоения другой части материала необходимо иметь опыт и навыки программирования, желательно на Java (напр., для того, чтобы освоить «рекомендации по разработке»). Идеальный вариант – сочетание всех навыков.
* неплохо бы иметь представление о серверах баз данных (хотя бы общие принципы функционирования), уметь пользоваться языком запросов SQL и средствами (любыми) для выполнения запросов SQL и просмотра результатов выполнения запросов для конкретной СУБД.
* понимание формата XML – общие принципы. По ходу изложения будут появляться и другие форматы файлов – но для продвинутых пользователей ОС семейства Windows они не должны будут составить проблему.
* знание «технического» английского – не все материалы полностью переведены – на часть материалов просто даны ссылки на инет-ресурсы (они обычно на языке оригинала – на английском)

### Использованное ПО

При написании данного документа я использовал достаточно широкий набор разных программных инструментов:

#### Операционные системы

* Windows 2003 server (standard edition)
* Windows XP/Vista/7/8
* Ubuntu 12.X (desktop)
* Ubuntu 12.X (server)

#### Серверное ПО

* IIS 6.0
* Tomcat 6/7
* Glassfish 3.1
* Apache 2
* MySql 5
* MS Sql 2008/2010
* Oracle DBMS 10/11

#### Среды разработки, различные инструменты и компоненты

* JDK 1.6/1.7
* Ant 1.8.1
* Maven 2/3
* Компонент isapi\_redirect\_1.2.30.dll (для связки Tomcat+IIS)

## Глава 2. Общие вопросы разработки ПО.

В данной главе будут обсуждены различные общие вопросы, связанные с разработкой в целом, а не с отдельно взятыми технологиями.

### «Золотые правила программирования»

Данные правила взяты из замечательной книги Питера Гудлифа «Ремесло программиста». Каждый разработчик должен прочитать эту книгу не один раз! Для того чтобы просто освежить память, основные моменты книги, т.н. «Золотые правила» я перенес сюда. Некоторые правила я пропустил – посчитал их неочень важными, за полным списком – обращайтесь к книге. Еще я добавил немного своих комментариев к некоторым правилам.

#### Перед лицом кода (Часть I)

* Не делайте никаких допущений. Не зафиксированные формально допущения часто служат причиной отказов, особенно с ростом кода.
* Чем больше спешки, тем меньше скорость. Всегда думайте, что вы собираетесь ввести с клавиатуры.
* Не верьте никому. Кто угодно, включая вас самих, может сделать ошибки в логике вашей программы. Ко всем входным и выходным данным относитесь с подозрением, пока не проверите, что они допустимы.
* Простота – это достоинство. Не усложняйте код сверх необходимости.
* Предупреждения компилятора помогаю выявить массу глупых ошибок. Всегда включайте их вывод. Добейтесь того, чтобы ваш код компилировался молча.
* Берегите все ограниченные ресурсы. Тщательно организуйте их захват и освобождение.
* Поймите, кто реально станет читать ваш исходный код: другие программисты. Пишите с расчетом на них.
* Выясните, какие стандарты оформления кода есть для выбранного вами языка, и освойте каждый из них на практике. Оцените преимущества и недостатки каждого.
* Выберите какой-то один правильный стиль кодирования и придерживайтесь его постоянно.
* Если в вашей группе принят некий стандарт кодирования, придерживайтесь его. Свой собственный излюбленный стиль оставьте в стороне.
* Скажи «нет» религиозным войнам. Не вступай в них. Отойди в сторону.
* Учитесь давать объектам прозрачные имена – они должны ясно описывать то, что за ними скрывается.
* Чтобы придумать хорошее имя, главное – четко понимать, для кого оно предназначено. Только в этом случае имя может стать осмысленным. Если не удается придумать хорошее имя объекту, спросите себя, понятно ли вам его назначение.
* Изучите правила образования имен в языке, с которым работаете. Еще важнее изучить идиомы этого языка. Есть стандартные способы формирования имен? Вот ими и пользуйтесь.
* Ясность имени предпочтительнее его краткости.
* Следует учитывать относительные достоинства коротких и длинных имен, особенно в зависимости от области действия переменной.
* Выбирайте для объектов правильные имена сразу и навсегда.
* Предпочтительней такое правило именования, которое различает имена переменных и имена типов.
* Давайте функциям имена с внешней точки зрения, в виде фраз, выражающих действие. Описывайте логическую операцию, а не способ реализации.
* Избегайте в именах типов лишних слов. В частности, в именах типов – таких слов, как class, data, object, type.
* Давайте пространствам имен и пакетам имена, логически связанные с их содержимым.
* Выберите единообразную систему именования и последовательно применяйте ее.
* Степень необходимой детализации имени зависит от контекста его применения. Создавая имена, учитывайте контекстную информацию.
* Не пишите код, который нуждается во внешней документации. Он ненадежен. Пишите такой код, который понятен без посторонней документации.
* Пишите код, который может прочесть нормальный человек, причем без напряжения. Компилятор как-нибудь справится.
* Избегайте волшебных чисел. Пользуйтесь именованными константами.
* Важные участки кода должны выделяться на общем фоне и быть легко читаемыми. Спрячьте все, что не должно интересовать клиентов.
* Старайтесь группировать родственную информацию. Делайте эту группировку наглядной с помощью средств языка.
* Не выводите бессмысленные сообщения об ошибках. В зависимости от контекста представьте наиболее уместную информацию.
* Помещайте в код комментарии только в том случае, если не удается облегчить его понимание иными способами.
* Пользуйтесь инструментарием грамотного документирования для автоматической генерации документации по вашему коду.
* Учитесь писать ровно столько комментариев, сколько необходимо. Отдайте предпочтение качеству, а не количеству.
* Не пожалейте труда, чтобы ваш код не требовал поддержки в виде уймы комментариев.
* Хорошие комментарии объясняют «почему», а не «как».
* Один источник для каждого факта. Не копируйте код в комментариях.
* Обнаружив, что вы пишете многословные комментарии, описывающие ваш код, остановитесь и задумайтесь. Не признак ли это того, что существует некая проблема более высокого порядка.
* Думайте, что пишете в комментариях; не давите бездумно на клавиши. Прочтите комментарий снова в контексте кода. Ту ли информацию он содержит.
* Комментарии – часть повествования кода. Размещайте их так, чтобы порядок чтения был естественным.
* Снабжайте каждый файл исходного кода прологом в виде комментария.
* Комментарии должны касаться настоящего, а не прошлого. Не описывайте того, что претерпело изменения, и не рассказывайте о том, как было раньше.
* Если вы изменили код, проверьте правильность комментариев, находящихся рядом с ним.
* Обработка ошибок – дело серьезное. От нее зависит стабильность вашего кода.
* Не оставляйте сбойную ситуацию без внимания. Если вы не знаете, как справиться с проблемой, сигнализируйте об отказе вызвавшему коду. Не заметайте мусор под ковер в надежде, что все как-нибудь обойдется.
* Никогда не пренебрегайте поступающими сообщениями об ошибках. Если существует канал для сообщений об ошибках, значит, для этого есть причины.
* Обрабатывайте все ошибки в наиболее благоприятном контексте, когда становится ясно, как корректно с ней справиться.
* Игнорирование ошибок не экономит вашего времени. Вы потратите больше времени на выяснение причин некорректного поведения программы, чем вам понадобилось бы для написания обработчика ошибок.
* Если вы пишете код, который может отказать, одновременно с ним пишите код для обнаружения и обработки ошибок. Не откладывайте это на будущее. Если вы все же вынуждены отложить обработку, по крайней мере, напишите оснастку для обнаружения ошибок.

#### Тайная жизнь кода (Часть II)

* Изучите свои стандартные инструменты вдоль и поперек. Время, которое вы потратите на их изучение, незамедлительно окупится.
* Относитесь к инструментам программирования прагматически; пользуйтесь ими, только если они облегчают вашу жизнь.
* Узнайте, какие виды ьинструментов существуют. Выясните, где их можно взять, даже если в данный момент они вам не нужны.
* Для каждой задачи есть свой инструмент. Не стоит щелкать орехи кувалдой.
* Следите за выпуском новейших версий своего инструмента, но проявляйте осторожность при обновлении.
* Выбор редактора кода имеет решающее значение; он оказывает огромное влияние на то, как вы будете писать код.
* Выучите несколько языков. В каждом вы обнаружите особый способ решения задач. Рассматривайте их как набор инструментов и выбирайте тот, который наиболее эффективен в конкретной ситуации.

*Мой коммент: спорное утверждение – при современных размерах технологий знать хорошо несколько инструментов очень проблематично!*

* Тестирование – это не отладка. Не путайте между собой эти виды работы. Каждый требует особых навыков. Следите за тем, чем вы заняты – тестированием или отладкой.
* **Тестирование может вскрыть только наличие ошибок. Оно не может доказать отсутствие неисправностей. Не поддавайтесь ложному чувству спокойствия, если код прошел ряд неадекватных тестов.**

*Мой коммент: это САМЫЙ важный аспект тестирования!*

* Тестируйте каждый написанный вами фрагмент кода. Не рассчитывайте, что кто-то другой сделает это за вас.
* **Чтобы сделать тестирование эффективным, его нужно начинать заранее, когда выявляемые ошибки еще не могут принести большого вреда. Тестовый код можно писать раньше, чем рабочий!**

*Мой коммент: эта техника называется Test Driven Development (TDD) – разработка, управляемая тестированием.*

* Пишите тесты для всех выявленных ошибок. Прогоняйте тесты как можно чаще.

*Мой коммент: для этой цели (чаще запускать тесты) предназначены системы* ***непрерывной сборки****. Одна из таких систем – Hudson.*

* Читая код, очень легко обмануться и поверить, что он работает правильно. Если вы написали код, то при его чтении вы будете видеть то, что собирались написать, а не то, что написали на самом деле. Учитесь читать код с циничным недоверием.
* Напишите полный набор тестов, каждый из которых будет проверять определенный аспект кода. 15 тестов, которые демонстрируют одну и ту же ошибку, менее полезны, чем 15 тестов, демонстрирующих 15 разных ошибок.
* Архитектура кода должна облегчать его тестирование.

*Мой коммент: следование данному положению приводит нас к «слабо связанным» программным модулям и использованию интерфейсов, а не прямых реализаций. «Слабо связанные» компоненты системы легче тестировать по отдельности и все вместе.*

* Насколько это возможно, автоматизируйте тестирование кода. Это быстрее и проще, чем выполнять тесты вручную, и гораздо надежнее: более вероятно, что тесты станут выполняться регулярно.
* Выполняйте тестирование автоматически в ходе процедуры сборки.
* Компилируйте код при включенном выводе компилятором всех предупредительных сообщений. Тем самым вы обнаружите потенциальные проблемы раньше, чем столкнетесь с ними реально.
* Следуйте золотому правилу отладки: думать головой.

*Мой коммент: это правило должно быть в начале каждого документа по любой технологии, а то бывает, забывают…*

* Изучите отлаживаемый код – трудно найти ошибки в коде, который вам непонятен.
* Когда вы ищете ошибку, не верьте никому. Проверьте самые невероятные причины, вместо того, чтобы сходу отвергнуть их. Не принимайте ничего на веру.
* Если сборка продукта не прошла, смотрите на первую ошибку компилятора. Последующие сообщения заслуживают гораздо меньше доверия.
* Отладка – это методичная работа, медленно сужающая кольцо вокруг места нахождения ошибки. Не следует относиться к ней как к игре в угадайку.
* Первый шаг на пути установления места ошибки – это определение способа ее уверенного воспроизведения.
* Если вам показалось, что вы нашли причину ошибки, досконально исследуйте ее и убедитесь, что не ошиблись. Не принимайте безрассудно первую же гипотезу.
* Отладка заканчивается лишь тогда, когда вы докажете, что ошибка устранена и проблема решена навсегда.
* Проявляйте крайнюю осторожность при исправлении ошибок. Следите, чтобы ваша модификация не покалечила ничего остального.
* Исправляя ошибку, проверьте, не повторяется ли она в близких разделах кода. Уничтожьте ошибку раз и навсегда: исправьте все ее дубликаты немедленно.
* Из каждой исправленной ошибки делайте выводы. Можно ли было ее избежать? Можно ли было обнаружить ее быстрее?
* Считайте систему сборки частью дерева исходного кода и ведите их совместно. Они тесно связаны между собой.
* Все программисты, участвующие в проекте, должны пользоваться единой системой сборки. Иначе они будут собирать разные программные пакеты.
* Правильная система сборки позволяет многократно создавать физически идентичные бинарные файлы.
* Вы должны быть в состоянии достать дерево исходных файлов трехлетней давности и правильно собрать его заново.
* Правильная сборка выглядит как одна операция. Достаточно нажать кнопку или набрать одну команду.
* Для каждого правила сборки напишите соответствующее правило чистки, которое отменяет всю операцию.
* Организуйте автоматическую процедуру сборки своего программного продукта. Проверяйте с ее помощью работоспособность вашего кода.
* Финальные сборки всегда выполняются из чистого исходного кода. Позаботьтесь, чтобы потом всегда можно было получить этот чистый исходный код из архива или системы управления версиями.
* Об эффективности работы программы нужно думать с самого начала – не надейтесь, что в конце разработки вам удастся повысить ее ценой небольших изменений.
* Корректность кода гораздо важнее его скорости. Что толку быстро получить результат, если он неверный!
* Рассмотрите альтернативы оптимизации; не удастся ли вам повысить эффективность программы иными способами?
* Выясните, действительно ли есть необходимость оптимизировать код, но лучше с самого начала писать эффективный код высокого качества.
* Оптимизируйте код отдельно от всякой другой работы, чтобы результаты одной работы не оказывали влияния на другую. Оптимизировать надо окончательные версии программы, а не промежуточные сборки.
* Тщательно отберите входные данные для профилирования, с тем чтобы они отражали реальный мир. В противном случае может оказаться, что вы оптимизируете те части программы, которые в обычных условиях не выполняются.
* Не ограничивайтесь профайлером при поиске причин недостаточной эффективности программы; они могут оказаться более глубокими.
* Обязательно проводите измерения до и после оптимизации.
* Лучше заменить медленный алгоритм быстрым, чем пытаться улучшить реализацию имеющегося алгоритма.
* Следует учитывать, какими важными ресурсами вы располагаете. Есть ли у вас особо деликатная информация или специальные возможности, за которыми могут охотиться? Обеспечьте их защиту.
* Чем сложнее компьютерная система, тем более вероятно, что в ее системе безопасности есть пробелы. Следовательно, пишите как можно более простое программное обеспечение!
* Не проходите мимо уязвимостей и не считайте себя непобедимым. Где-нибудь обязательно найдется тот, кто попытается применить эксплойт к вашему коду.
* Безопасность – важный аспект архитектуры любого программного продукта. Будет ошибкой не позаботиться о ней на ранних стадиях разработки.
* Проектируя программу, рассчитывайте только на известные, защищенные компоненты сторонних разработчиков.
* Готовьтесь к тому, что ваша система будет атакована, и проектируйте с учетом этого все ее части.

#### Проектирование кода (Часть III)

* Программирование – это работа по проектированию. Это творческий и художественный акт, а не механическое написание кода.
* Составьте понятный проект, прежде чем стучать по клавишам.
* Чем меньше, тем лучше. Вашей целью должен быть простой код, который при малом размере решает большие задачи.
* Сделать простую вещь сложно. Если структура кода кажется очевидной, не надо думать, что это досталось без труда.
* Проектируйте такие модули, которые внутренне связны и минимально соединены между собой. Декомпозиция должна отражать правильное разбиение задачи на части.
* Проведите черту, за которую никто не должен переступать: определите четкие API и интерфейсы.
* Проектируйте с учетом последующего расширения, но не перестарайтесь, иначе вы напишете не программу, а ОС.
* Делай один раз. Делай хорошо. Избегай дублирования.

#### Стадо программистов? (Часть IV)

TBD

#### Часть процесса (Часть V)

TBD

#### Вид сверху (Часть VI)

TBD

### «Тест Джоэла»

Имя тесту дал Джоэл Спольски – спросите Гугл или Википедию, кто это такой ☺. Этот тест можно использовать для «грубой» оценки команды разработки/компании по разработке ПО. Итак, вопросы:

1. Используете ли вы систему управления версиями исходного кода?
2. Можете ли вы выполнить сборку продукта за один шаг?
3. Выполняете ли вы ежедневную компиляцию?
4. Ведете ли вы базу данных ошибок в программе?
5. Исправляете ли вы ошибки, прежде чем писать новый код?
6. Есть ли у вас актуальный график работы?
7. Есть ли у вас спецификации?
8. Создали ли вы спокойные условия для работы программистов?
9. Стараетесь ли вы использовать для работы лучшие из существующих инструметов?
10. Привлекаете ли вы к работе тестеров?
11. Предлагаете ли вы соискателям рабочих мест написать во время собеседования код?
12. Проводите ли вы проверку «юзабилити» на случайных людях?

На каждый вопрос необходимо ответить «да» или «нет», за каждый ответ «да» начисляется один балл.

Итоги: если менее 10 баллов – это очень плохо/печально (указывает на наличие серьезных проблем), 11 баллов – хорошо, 12 – совсем отлично! При всех равных условиях, если выполняются все 12 пунктов – это означает, что у вас есть дисциплинированная команда, способная стабильно выдавать готовый продукт.

Данный тест позволяет «на глазок» (грубо) определить необходимость серьезных улучшений в команде/компании.

### Объектно-ориентированное vs функциональное программирование

1. В основе функционального программирования лежит осмысление предметной области в терминах неизменяемых значений и функций, которые их преобразуют.
2. В объектно-ориентированном программировании абстрагируются главным образом данные, а в функциональном – поведение.

## Глава 3. Организация работы.

**Основные принципы организации работы разработчика.**

Организация проекта

build

[PROJECT]

docs

resources

src

web

temp

libs

META-INF

WEB-INF

**build.xml**

* **[PROJECT]** – каталог расположение проекта. Проект может располагаться в любом месте на жестком диске ПК.
* **build** – каталог для компиляции исходного кода и сборки приложения. Компилятор складывает в этот каталог скомпилированные файлы классов (.class) и файлы ресурсов, включаемых в пакеты приложения.
* **libs** – используемые модулем/проектом библиотеки (файлы .jar).
* **libs/docs** – документация к используемым проектом/модулем библиотекам. Опциональный каталог – документация может храниться и в другом месте или не использоваться вообще.
* **resources** – каталог ресурсов приложения/модуля. В данном каталоге можно (и нужно) хранить скрипты для создания БД приложения, схемы и описания для приложения, другие вспомогательные файлы.
* **src** – непосредственно исходный код приложения. Здесь же располагаются ресурсы, которые необходимо включить в пакеты сборки. Обычно эти ресурсы – файлы конфигурации (xml) или файлы свойств (properties).
* **temp** – каталог для хранения временных файлов при выполнении сборки приложения.
* **web** – данный каталог предназначен для хранения ресурсов web-приложения – html/jsp и т.п. Если приложение не содержит веб-интерфейса, данный каталог не нужен.
* **web/META-INF** – в данном каталоге хранятся файлы для контейнера приложения j2ee (для Томката это, например, context.xml файл).
* **web/WEB-INF** – каталог для хранения конфигурации веб-приложения (файл web.xml и т.п.). Данный каталог – каталог для разработки, в него НЕ кладутся скомпилированные классы приложения! При сборке данный каталог копируется в другой (например **temp**) и уже тогда в него помещаются либы и скомпилированные классы приложения (в каталоги **WEB-INF/libs** и **WEB-INF/classes** соответственно).
* **build.xml** – сборочный скрипт ANT для данного приложения/модуля, находится в корневом каталоге модуля.

Требования к ANT'у

Структура сборочного скрипта ANT

* загрузка глобального файла свойств
* установка набора библиотек
* инициализация (создание/очистка необходимых каталогов)
* компиляция исходного кода
* сборка jar/war/ear
* генерация javadoc
* дистрибуция приложения/модуля, выкладывание на сетевой/локальный ресурс, отправка на сервер FTP
* почтовое уведомление коллег о выходе новой версии разрабатываемой системы
* опциональная задача: обновление локальных библиотек проекта из сетевого репозитория

Создание нового или использование уже существующего проекта в среде Intellij IDEA (использовал 10.5.1)

Для создания нового проекта в системе IDEA необходимо выполнить несколько простых шагов, которые будут описаны ниже. Если же у нас уже есть готовый проект, созданный по описанной методологии (см. выше структуру каталогов), то для работы с этим проектом его необходимо подгрузить в среду разработки. Итак, приступим!

1. Предварительные действия:

* **создание нового проекта:** для начала в любом файловом менеджере создаем каталог проекта (в дальнейшем это расположение будет упоминаться как [путь\_к\_проекту]) и два подкаталога в каталоге проекта: [путь\_к\_проекту]\libs и [путь\_к\_проекту]\libs\docs. Это необходимо для создания в проекте в среде разработки объекта «библиотеки проекта» («project libraries»).
* **использование уже существующего проекта:** копируем проект из сетевого репозитория в каталог на нашем локально компе (в дальнейшем это расположение будет упоминаться как [путь\_к\_проекту])

1. Запускаем среду Идея. Если автоматически открывается другой проект – можно его оставить или закрыть – не имеет значения.
2. Выбираем меню: File -> New Project… -> Create project from scratch (Next)
3. Первоначальная настройка нового/используемого проекта:

* создаем имя для нового проекта (поле Name) – для простоты и удобства имя проекта совпадает с каталогом, в котором непосредственно расположен проект
* выбираем расположение файлов проекта (поле Project files location) – созданный нами каталог [путь\_к\_проекту] или каталог, в который мы скопировали проект из сетевого репозитория
* формат проекта (поле Project file format) – выбираем любой

1. Ставим галку в поле Create module – для создания модуля нашего проекта, затем указываем свойства модуля:

* имя (Name) – для удобства имя модуля может совпадать с именем проекта
* пути к содержимому модуля (Content root и Module file location) – эти пути для простоты оставляем совпадающими с расположением файлов проекта (Project files location)
* тип модуля (Select type) – в нашем случае это обычно Java Module.

1. Жмем Next!
2. Указываем каталог для исходников (src) -> Next
3. Следующее окно мастера – используемые фреймворки. Фреймворков не указываем – все библиотеки будут скопированы в проект вручную. Жмем Finish. Вуаля! Теперь необходимо поднастоить созданный проект.
4. File -> Project structure. Далее по разделам данного мастера.
5. Раздел Project. Настройки:

* Project SDK (указать)
* Project language level: 6.0
* Project compiler output: [путь\_к\_проекту]\build

Жмем Apply.

1. Раздел Libraries (немного не по порядку, но так надо!). Настройки:

* добавить библиотеку (+) и добавить эту библиотеку к модулю нашего проекта. Наименование библиотеки – project libraries. Хотя – может быть любым.
* Attach Jar Directories – [путь\_к\_проекту]\libs
* Attach Documentation - [путь\_к\_проекту]\libs\docs

Жмем Apply.

1. Раздел Modules (немного назад). Настройки:

* Paths -> Use module compile output (checked)
* Paths -> Output path -> [путь\_к\_проекту]\build
* Paths -> Test output path -> [пусто – очищаем поле]
* Paths -> Exclude output paths (checked)
* Dependencies -> проверить, что наша библиотека (project libraries) указана в зависимостях модуля приложения. Если нет – указать.

Жмем Apply.

1. Данный пункт только для **импортируемого проекта:** добавляем в среду разработки файл скрипта ANT, который должен быть в проекте. Также настраиваем ANT для работы с внешним движком – не со встроенным в Идею движком!
2. На этом основная настройка нового/импортируемого проекта завершена. Если в пункте 11 (раздел Project мастера Project structure) был изменен уровень языка, то среда предложит перезагрузить проект – соглашаемся.

# Часть II. Непосредственно JAVA.

## Глава 4. Начало

### Инсталляция Java.

Для того, чтобы начать использование Java, необходимо скачать и установить пакет JDK и документацию к нему. С этой задачей по силам справиться даже начинающему программисту.

Также для удобства разработки необходима будет интегрированная среда разработки (IDE) на Java. Основные среды для разработки на Java:

* Intellij IDEA.
* NetBeans.
* Eclipse.

Программы на Java можно писать и в текстовом редакторе, для компиляции/запуска можно использовать компилятор/java-машину из пакета JDK (работая в командной строке), но такой подход напоминает каменный век. Большой проект с большим количеством классов разрабатывать таким способом будет достаточно трудно. Поэтому настоятельно рекомендую использование графической среды.

### Инструменты JDK.

Кроме основных инструментов, которыми мы будем иметь дела 90% времени – java, javac, jar – в пакете JDK есть еще большой набор разных утилит. Вот некоторые из них:

* jstack - позволяет сделать thread dump для любого приложения java по его PID
* jvisualvm - новая (1.6.0\_07+) графическяи оболочка для мониторинга и профайлинга.
* javap – дизассемблер, может показывать инструкции jvm.

Полное описание утилит JDK – см. здесь <http://docs.oracle.com/javase/6/docs/technotes/tools>

### Объектно-ориентированное программирование.

*Данный раздел покрывает темы: классы, объекты, наследование, переопределение (overriding) методов, перегрузка (overloading) методов, переменные, константы, типы данных, области видимости переменных, оберточные классы, autoboxing, unboxing.*

## Глава 5. Основные типы данных Java.

### Примитивные типы данных

В таблице ниже указаны все примитивные типы данных, используемые в Java.

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Диапазон** | **Размерность** | **Описание** |
| boolean | true/false | 1 bit |  |
| byte | от -128 до 127 | 1 byte |  |
| short | от -32768 до 32767 | 2 bytes |  |
| int | от -2147483648 до 2147483647 | 4 bytes |  |
| long | от -9223372036854775808  до 9223372036854775807 | 8 bytes |  |
| float | приблизительно ±3,40282347E+38F  (6-7 значащих десятичных цифр) | 4 bytes |  |
| double | приблизительно ±1,7976931348623157E+308F  (15 значащих десятичных цифр) | 8 bytes |  |
| char | от '\u0000' до '\uFFFF', в десятичной форме от 0 до 65535 | 2 bytes |  |

#### Числовые примитивные типы данных (целые и с плавающей точкой).

* В Жабе диапазоны целочисленных типов не зависят от машины, на которой выполняется программа.
* Длинные целые числа (long) имеют суффикс L (например 4000L). Шестнадцатеричные числа имеют префикс 0x (0xCAFE). Восьмеричные числа имеют префикс 0 (восьмеричное 010 – это 8 десятичное).
* Числа типа float имеют суффикс F (3.402F). Числа с плавающей точкой, не имеющие суффикса F (3.402), всегда рассматриваются как числа double. Для их представления можно (но необязательно) использовать суффикс D (3.402D).
* В Жабе есть три специальных значения с плавающей точкой:
* положительная бесконечность (Double.POSITIVE\_INFINITY, Float.POSITIVE\_INFINITY)
* отрицательная бесконечность (Double.NEGATIVE\_INFINITY, Float.NEGATIVE\_INFINITY)
* NaN (не число – Not a Number) (Double.NaN, Float.NaN)

Эти значения используются для обозначения переполнения и ошибок. Например: результат деления положительного числа на 0 равен положительной бесконечности, вычисление 0/0 или квадратный корень из отрицательного числа равны NaN. Проверка соответствия этим константам:

if (x == Double.NaN) – так низя! А вот так – if (Double.isNaN(x)) – мона!

* Для обозначения арифметических операций используются операторы + - \* /. Оператор / означает целочисленное деление, если оба аргумента целые числа, в противном случае это деление чисел с плавающей точкой. Оператор % - остаток от деления целых чисел.

**Целочисленное деление на 0 генерирует исключение, результатом же деления на 0 чисел с плавающей точкой является бесконечность или NaN.**

* Для точных вычислений с плавающей точкой (для вычислений, гарантирующих воспроизводимость результатов) метод класса помечается ключевым словом **strictfp**.
* Если два значения объединяются бинарным оператором, то перед выполнением операции оба операнда преобразовываются в числа одинакового типа, по следующим правилам:
* если хотя бы один из операндов имеет тип double, то второй тоже преобразовывается в double
* в противном случае, если хотя бы один из операндов имеет тип float, то второй тоже преобразовывается во float
* в противном случае, если хотя бы один из операндов имеет тип long, то второй тоже преобразовывается в long
* в противном случае оба операнда преобразовываются в int

Перед выполнением арифметической операции всегда происходит *повышение* (promotion) типов byte, short, char. Они преобразуются в тип int, а может быть, и в тип long, если другой операнд типа long. Операнд типа int повышается до типа long, если другой операнд типа long. Конечно, числовое значение операнда при этом не меняется. Это правило приводит иногда к неожиданным результатам:

*byte b1 = 50, b2 = -99;*

*short k = b1 + b2; // Неверно!*

В таких случаях следует выполнить явное приведение типа. В данном случае это будет *сужение* (narrowing) типа int до типа short. Оно осуществляется операцией явного приведения, которая записывается перед приводимым значением в виде имени типа в скобках. Определение

*short k = (short)(b1 + b2);*

будет верным.

Сужение осуществляется просто отбрасыванием старших битов, что необходимо учитывать для больших значений. Например, определение

*byte b = (byte) 300;*

даст переменной b значение 44 . Действительно, в двоичном представлении числа 300 , равном 100101100, отбрасывается старший бит и получается 00101100. Таким же образом можно произвести и явное *расширение* (widening) типа, если в этом есть необходимость. Если результат целой операции выходит за диапазон своего типа int или long, то автоматически происходит приведение по модулю, равному длине этого диапазона, и вычисления продолжаются, переполнение никак не отмечается.

* Существует сокращенная запись бинарных арифметических операторов: +=, -=, \*=, /=, %=.

**Отличие сокращенной записи бинарных операторов от обычной достаточно существенно!** Все составные операции присваивания действуют по одной схеме:

**х [орeration]= а**эквивалентно **х = (тип х)(х [орeration] а)**

Это означает следующее:

*byte b = 1;*

*b = b + 10; // Ошибка!*

*b += 10; // Правильно!*

* В результате приведения значения с плавающей точкой к целому типу дробная часть отбрасывается.
* Целые константы хранятся в формате типа int. В конце целой константы можно записать букву прописную L или строчную l , тогда константа будет сохраняться в длинном формате типа long: +25L, -0371, OxffL, OXDFDF1. В конце действительной константы можно поставить букву F или f , тогда константа будет сохраняться в формате типа float: 3.5f, -45.67F, 4.7e-5f . Можно приписать и букву D (или d ): 0.045D, -456.77889d , означающую тип double, но это излишне, поскольку действительные константы и так хранятся в формате типа double.
* Операция *деление по модулю* определяется так: а % b = а - (а / b) \* b ; например, 5%2 даст в результате 1 , а 5% (-3) даст, 2 , т.к. 5 = (-3) \* (-1) + 2 , но (-5)%3 даст -2 , поскольку -5 = 3 \* (-1) - 2.

#### Логический тип данных.

* Для типа boolean есть всего два значения false/true, которые не преобразуются в целочисленные и наоборот.
* Логические операции:
* отрицание (NOT), обозначается ! (меняет значение истинности)
* конъюнкция (AND), обозначается & или && (ИСТИНА, только если оба операнда ИСТИНА)
* дизъюнкция (OR), обозначается | или || (ЛОЖЬ, только если оба операнда ЛОЖЬ)
* исключающее ИЛИ (XOR), обозначается ^ (ИСТИНА, только если значения операндов различны)
* Логические операторы: |, ||, &, &&. Разница между операторами | и ||, & и && проста: одинарные операторы вычисляют логические выражения всегда до конца, двойные операторы (их еще называют короткозамкнутыми), вычисляют выражение не всегда до конца, а до тех пор, пока не будет известен результат, например: if ((x != 0) && (1/x > x + y)) – в данном выражении при x=0 результат всегда будет false – соответственно не происходит деления на 0 во втором логическом операнде.
* Для логических значений существует тернарный оператор **:?** , который иногда помогает. Оператор записывается так: **условие ? выражение1 : выражение2**, если условие истинно, выполняется/вычисляется первое выражение, в противном случае – второе.

#### Символьный тип данных

* Тип char занимает два байта, в арифметических вычислениях он участвует как тип int , ему выделяется 4 байта, два старших байта заполняются нулями.
* Код любого символа с десятичной кодировкой от 0 до 255 можно задать, записав его не более чем тремя цифрами в восьмеричной системе счисления в апострофах после обратной наклонной черты: '\123' — буква S, '\346' — буква Ж в кодировке СР1251. Не рекомендуется использовать эту форму записи для печатных и управляющих символов, перечисленных в предыдущем пункте, поскольку компилятор сразу же переведет восьмеричную запись в указанную выше форму. Наибольший код '\377' — десятичное число 255.
* Код любого символа в кодировке Unicode набирается в апострофах после обратной наклонной черты и латинской буквы ц ровно четырьмя шестнад-цатеричными цифрами: '\u0053' — буква S, '\u0416' — буква Ж. Прописные русские буквы в кодировке Unicode занимают диапазон от '\u0410' — заглавная буква А, до '\u042F' — заглавная Я , строчные буквы от '\u0430' — а, до '\044F' — я.

#### Общие заметки для всех примитивных типов данных

* Преобразование (приведение) числовых типов:

**byte -> short -> int -> long** - преобразование без потери информации

**float -> double** - преобразование без потери информации

**char -> int** - преобразование без потери информации

**int -> double** - преобразование без потери информации

**int -> float** – преобразование, при котором может произойти потеря точности

**long -> float** – преобразование, при котором может произойти потеря точности

**long -> double** – преобразование, при котором может произойти потеря точности

Пример потери точности: количество цифр в целом числе 123456789 превышает количество цифр, которое может быть представлено типом float. Число, преобразованное в этот тип имеет тот же порядок, но меньшую точность:

int n = 123456789;

float f = n; // Содержимое f = 1.234567892E8

### Типы «оболочки» для примитивных типов данных

Для примитивных типов, не являющихся обектами, существуют соответствующие «объектные» типы-оболочки. Их иерархия представлена на рисунке.

Object

Number

AtomicInteger

AtomicLong

BigDecimal

BigInteger

Float

Double

Long

Integer

Short

Byte

## Глава 6. Основные конструкции языка.

Этот раздел содержит описание следующих тем: отладка, циклы, массивы, условный оператор.

## Глава 7. Инкапсуляция и полиморфизм. Абстрактные классы и интерфейсы.

Данный раздел покрывает темы:

## Глава 8. Некоторые интересные классы java. Краткий обзор.

### Базовый класс Object.

Данный класс является предком (базовым классом) для всех классов JAVA (этот класс – корень иерархии классов). Каждый класс имеет своим суперклассом Object, если явно не указан предок класса. Все объекты, включая массивы, реализуют методы данного класса. Непосредственно сами методы:

* public final Class getClass()
* public int hashCode()
* public boolean equals()
* protected Object clone() throws CloneNotSupportedException
* public String toString()
* public final void notify()
* public final void notifyAll()
* public final void wait(long timeout) throws InterruptedException
* public final void wait(long timeout, int nanos) throws InterruptedException
* public final void wait() throws InterruptedException
* protected void finalize() throws Throwable

### Класс String.

## Глава 9. GUI.

В настоящее время пользователи привыкли работать с «богатыми» (rich) интерфейсами.

Если необходимо разработать такой интерфейс для Интернет приложения, то следует использовать новую технологию (платформу) JavaFx.

Если же необходимо разработать графический интефейс пользователя (GUI) для настольного (desktop) приложения или для специального типа веб-приложений – апплета (applet), платформа Java предлагает набор компонентов Swing.

В самом начале истории Java для построения графических интерфейсов использовался пакет Abstract Windowing Toolkit (AWT). Этот пакет был очень «тяжел» и созданные с его помощью интерфейсы были медленными и неудобными. Ему на смену пришел другой пакет классов – Swing. Этот пакет предлагает легковесные платформонезависимые компоненты пользовательского интерфейса – компоненты-контейнеры, элементы управления и т.п. В настоящее время большой популярностью пользуются веб-интерфейсы, однако разработка с помощью Swing все еще занимает большой сегмент и достаточно востребована.

Типичный процесс разработки интерфейса с применением Swing состоит из компоновки элементов управления в компонент-контейнер, который поддерживает различные виды компоновок (layouts) с помощью выбранного менеджера компоновки. Компонент-контейнер с элементами управления добавляется в контейнер верхнего уровня – обычно это базовый компонент из пакета AWT – JFrame, для добавления в него контейнера с элементами управления используется метод setContentPane().

Или, по шагам:

1. Создаем необходимый набор компонентов
2. Создаем компонент-контейнер
3. Создаем экземпляр менеджера компоновки и связываем его с контейнером.
4. Добавляем компоненты в контейнер.
5. Контейнер с компонентами добавляем в контейнер верхнего уровня (обычно JFrame).
6. Пишем код обработчиков событий и регистрируем эти обработчики в контейнере компонентов.

### Менеджеры компоновки (layout managers).

* **FlowLayout**. Располагает компоненты в контейнере строка за строкой по мере их добавления. При изменении размеров окна расположение компонентов может измениться (перенос на следующую/предыдущую строку).
* **GridLayout**. Располагает компоненты в контейнере в строках и столбцах – как в сетке (grid). Компоненты добавляются в воображаемые ячейки сетки, которые меняют свой размер при изменении размеров контейнера (окна). При этом все ячейки и расположенные в них элементы управления имеют одинаковый размер.
* **BoxLayout**. Данный менеджер компоновки позволяет элементам управления располагаться вдоль горизонтальной оси (x-axis) или вдоль вертикальной оси (y-axis). В отличие от FlowLayout, при изменении размеров окна с BoxLayout компоненты не меняют своего положения (перенос компонента на следующую строку, в другой столбец и т.п.). У данного менеджера есть отличие и от менеджера GridLayout – элементы управления могут иметь разный размер.
* **BorderLayout**. Этот менеджер делит пространство контейнера на несколько частей – Южная (South), Западная (West), Северная (North), Восточная (east) и Центральная (Center). Расположение частей соответствует их наименованию. Компоненты (элементы управления) могут располагаться в любой из этих частей, все части использовать необязательно – менеджер компоновки сам подстроится под добавленные компоненты. Неиспользуемые части пространства будут заняты частями, содержащими компоненты.
* **CardLayout**. Данный менеджер компоновки реализует механизм компонента с табличными вкладками – tabbed folder.
* **GridBagLayout**. Это «продвинутый» менеджер компоновки типа «сетка» (grid), который позволяет ячейкам сетки иметь *разный* размер. Этот менеджер компоновки работает совместно с классом GridBagConstraints – набор ограничений. Ограничения – свойства ячейки сетки и их необходимо задавать отдельно для каждой ячейки.

Для использования любого из менеджеров компоновки необходимо создать его экземпляр и связать с компонентом-контейнером с помощью метода setLayout().

Компоненты контейнеры могут быть вложены один в другой, глубина вложенности явно не ограничена (но не стоит этим злоупотреблять!). Т.е. вложенность может быть такой:

*top-level container -> container1 -> (container2+(container3->container4))*

Есть еще одна возможность компоновки элементов управления – указание точных абсолютных позиций компонентов, и тогда размеры компонентов будут оставаться прежними вне зависимости от размеров окна.

### Обработка событий в интерфейсе.

## Глава 10. Исключительные ситуации.

(см. главу 11 книги Корнелла-Хорстманна Java 2)

Иерархия ИС в Java выглядит следующим образом:

**Error**

**Exception**

**Runtime exceptions**

**(unchecked)**

**Throwable**

**Checked exceptions**

Все ИС наследуются от общего класса Throwable (класса, не интерфейса!) и делятся на две большие группы: Error (ошибки, это также класс) и Exception (исключительные ситуации, и это класс!). В чем отличие?

Группа error: внутренние ошибки JVM (нехватка ресурсов и т.п.), объекты данного типа сгенерировать нельзя, также нельзя перехватить и обработать (крах JVM), возможно лишь уведомить юзера. **Кстати:** средства языка позволяют объявлять данные ошибки, генерировать и перехватывать их, только вопрос: если возникнет такая ошибка, успеет ли выполниться наш обработчик или Java-машина рухнет первой?

Группа exception: ошибки программы, которые возможно сгенерировать, перехватить, обработать. Данная группа также делится на две: runtime exceptions (ошибки времени выполнения, unchecked - неконтролируемые) и checked exceptions (контролируемые). Исключения типа RuntimeException (это класс-родитель всех неконтролируемых исключений) – практически всегда – ошибки программиста (неверное приведение типов, выход за пределы массива и т.п.), исключения типа checked – ожидаемые и обрабатываемые ситуации (отсутствие нужного файла, недоступность сетевого адреса и т.п.). Для всех контролируемых (checked) исключений компилятор выполняет поиск (и выполнение, если найден) обработчика.

Контролируемые ИС (checked) объявляются в заголовке метода (или обрабатываются внутри него), неконтролируемые же объявлять не нужно (также, как и обрабатывать их – это дурной тон программирования!). ИС типа error ни объявлять, ни обрабатывать/перехватывать нельзя (так делать совсем не стоит!) – это ошибки JVM и работать с ними должна она! (ИС типа Error можно объявить в заголовке метода, можно создать экземпляр данного класса или его подклассов, можно перехватить возникновение в блоке try…catch)

ИС и наследование. Метод подкласса не может генерировать более общие исключения, чем переопределяемый им метод суперкласса (исключения могут быть такими же, конкретизироваться – быть подклассами исключений метода суперкласса или не возникать вовсе). Также метод подкласса может объявить RuntimeException или его потомка/потомков.

Метод, объявляющий о генерации ИС определенного класса, может также генерировать ИС, являющиеся подклассами объявленной ИС.

**Рекомендации по использованию ИС.**

1. Обработка ИС не может заменить собой проверку. Простая проверка выполняется намного быстрее, чем перехват и обработка ИС.
2. Не увеличивайте без надобности количество блоков try.
3. Пользуйтесь возможностями, которые предоставляет иерархия исключений. Не перехватывайте Throwable и Exception. Различайте checked и unchecked ИС. При необходимости выполняйте преобразование типов исключений.
4. Не подавляйте исключения.
5. Не избегайте генерации исключений, информирующих об ошибках (например, об ошибочных входящих параметрах методов).
6. Не бойтесь передавать исключения другим обработчикам.

## Глава 11. Java Collections Framework

## Глава 12. Generics

## Глава 13. Система ввода-вывода, потоки ввода/вывода

Иерархия классов потоков ввода вывода выглядит следующим образом:

Object

InputStream

ByteArrayOutputStream

FileInputStream

AutoClosable (java.lang)

Reader

OutputStream

Writer

Closable (java.io)

Flushable (java.io)

FileOutputStream

ByteArrayInputStream

FilterOutputStream

FilterInputStream

BufferedInputStream

BufferedOutputStream

PushbackInputStream

SequenceInputStream

PrintStream

DataInputStream

DataOutputStream

DataOutput

DataInput

RandomAccessFile

Readable (java.io)

Appendable (java.io)

FileReader

FileWriter

CharArrayReader

CharArrayWriter

BufferedReader

BufferedWriter

PushbackReader

FilterReader

PrintWriter

Console

ObjectInput (java.io)

ObjectOutput (java.io)

ObjectOutputStream

ObjectOutputStream

### Интерфейсы

* **AutoClosable.** Интрефейс.
* **Closable.** Интрефейс.
* **Flushable.** Интрефейс.

### Потоки ввода байтовые

* **InputStream.** Абстрактный класс. Байтовый поток ввода (ASCII, 8bits).
* **FileInputStream**
* **ByteArrayInputStream**
* **SequenceInputStream**
* **FilterInputStream**
* **BufferedInputStream**
* **PushbackInputStream**
* **DataInput.** Интерфейс.
* **DataInputStream.** Позволяет читать элементарные данные (элементарных типов) из потока.

### Потоки вывода байтовые

* **OutputStream.** Абстрактный класс. Байтовый поток вывода (ASCII, 8bits).
* **FileOutputStream**
* **ByteArrayOutputStream**
* **FilterOutputStream**
* **BufferedOutputStream**
* **PrintStream**
* **DataOutput.** Интерфейс.
* **DataOutputStream.** Позволяет писать элементарные данные (элементарных типов) в поток.

### Потоки ввода символьные

* **Reader.** Абстрактный класс. Символьный поток ввода (Unicode, 16 bits).

### Потоки вывода символьные

* **Writer.** Абстрактный класс. Символьный поток вывода (Unicode, 16 bits).

## Глава 14. Сериализация

TBD

## Глава 15. NIO

Система NIO построена на **буферах** и **каналах**. Буфер – хранит данные, канал – представляет открытое соединение с устройством ввода-вывода (файлом, сокетом).

Буферы определены в пакете java.nio и все являются подклассами класса Buffer.

## Глава 16. Сетевое программирование.

TBD

## Глава 17. Многопоточность.

(см. главу 14 книги Корнелла-Хорстманна Java 2)

Состояния потока

## Глава 18. Работа с БД (JDBC)

TBD

## Глава 19. Java and functional programming.

Base functional interfaces in java (functional interface – interface with single abstract method, if such interface is used for lambda – it’s type is a lambda type too):

|  |  |  |  |
| --- | --- | --- | --- |
| **Interface** | **Arguments** | **Returns** | **Example** |
| Predicate <T> | T | Boolean | Condition match |
| Consumer <T> | T | Viod | Print some value |
| Function <T, R> | T | R | Get some info from object |
| Supplier <T> | - | T | Factory method |
| UnaryOperator <T> | T | T | Logic NOT |
| BinaryOperator <T> | (T, T) | T | Two numbers addition |

## Глава 20. Настройки и опции JVM

<http://docs.oracle.com/javase/8/docs/technotes/tools/windows/java.html>

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Глава 19. Common libraries/frameworks/tools.

Java не заканчивается знаниями лишь о возможностях JSDK. Чтобы чувствовать себя комфортно необходимо знать еще десятка два сторонних библиотек, используемых Java программистами повсеместно. Итак, начнем с основных:

* [Commons Lang](http://commons.apache.org/lang/) — то что «забыли» включить в JDK
* [Commons Math](http://commons.apache.org/math/) — отличное дополнение для java.math
* [Commons Logging](http://commons.apache.org/logging/) — логирование должно быть грамотным. За System.out.println для вывода логов начинающим программистам уже через неделю обучения следует отрубать руки.
* [Commons Net](http://commons.apache.org/net/) — логическое продолжение для пакета java.net. Множество классов для работы с основными сетевыми протоколами.
* [Commons VFS](http://commons.apache.org/vfs/) — отличная библиотека для абстрагирования от способа хранения файла. Позволяет достаточно обобщено иметь доступ до файлов по FTP, SFTP, WEBDAV, (G)ZIP и т.д.
* [Commons IO](http://commons.apache.org/io/) — работа с вводом-выводом часто получается весьма муторной, но с этой библиотекой все становится несколько веселее.
* [HttpClient](http://hc.apache.org/httpclient-3.x/) — библиотека по работе с http ресурсами.
* [JUnit](http://www.junit.org/) — любой код, должен сопровождаться тестированием. Для автоматизации тестирования и предназначена эта библиотека.

## Глава 20. Интересные задачки/Java puzzlers

В данном разделе собраны интересные задачи для решения или самостоятельной реализации. Некоторые задачи требуют ответа прямо сейчас, некоторые – решения в виде кода их реализующего. Just try!

1. **Вопрос.** Дан код на Java:
2. *public class Foo*
3. *{*
4. *private static Foo instance = new Foo();*
5. *private static final int DELTA = 6;*
6. *private static int BASE = 7;*
7. *private int x;*
8. *private Foo() { x = BASE + DELTA; }*
9. *public static void main(String... argv) {System.out.println(Foo.instance.x);}*
10. *}*

Что будет выведено и почему?

Ответ. Будет выведено 6. Пояснение: final-атрибуты как таковые не инициируются, их значения могут быть подставлены на этапе компиляции, то есть мы сразу имеем: [x = BASE + 6], а статическая переменная DELTA (не-final!) инициализируется значением по умолчанию = 0 при вызове конструктора (обращение к ней в конструкторе происходит до явной инициализации).

1. **Вопрос.** Дан код на Java:
2. *public class HelloWorld {*
3. *public static void main(String... args) {*
4. *http://habrahabr.ru;*
5. *System.out.println("Hello World!");*
6. *}*
7. *}*

Вопрос: скомпилируется или нет?

Ответ. Скомпилируется. «http:» — метка, "//habrahabr.ru;" — комментарий.

1. **Реализация.** Написать аналог консольной UNIX утилиты grep.

Необходимо реализовать консольную программу, которая бы фильтровала поток текстовой информации подаваемой на вход и на выходе показывала лишь те строчки, которые содержат слово передаваемое программе на вход в качестве аргумента.  
  
Варианты усложнения:

* Программа не должна учитывать регистр
* В аргументах может быть передано не одно слово, а несколько
* В качестве аргумента может быть задано не конкретное слово, а [регулярное выражение](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%BE%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5)

1. **Реализация.** Написать аналог консольной UNIX утилиты sort.

Написать консольную программу, которая бы сортировала текст поданный ей на стандартный вход по алфавиту.

Варианты усложнения:

* Программа должна игнорировать регистр при сортировке
* Программа должна сортировать не по алфавиту, а по количеству символов в строке
* Программа в качестве аргумента может получать порядковый номер слова в строке, по которому надо сортировать строки

1. **Реализация.** Написать программу чата – клиентскую и серверную часть. Задача состоит из этапов, в конце которых должен получиться стандартный консольный чат.  
   Этапы реализации программы:

* Реализовать консольную программу, которая бы открывала серверный сокет на порту 1234. При подключении клиента программа должна выводить все то, что передал клиент. Удостовериться в работоспособности можно попробовав подключиться с помощью системной программы telnet. В дальнейшем будем эту программу называть *серверной частью.*
* Усложнить программу и сделать так, чтобы можно было подключиться одновременно нескольким клиентам посредством telnet
* Реализовать другую консольную программу, которая по сути была бы аналогом telnet — т.е. подключалась к заданному IP на порт 1234 и отправляла бы на сервер строчку введённую пользователем по нажатию enter. В дальнейшем будем эту программу называть *клиентской частью* или просто *клиентом*
* Усложнить серверную часть программы таким образом, чтобы пришедшее сообщение от одного пользователя отправлялось сразу всем пользователям, которые подключены в данный момент к серверу
* Усложнить серверную часть программы так, чтобы при подключении нового пользователя ему показывались последние 10 сообщений
* При запуске клиентская часть должна спрашивать у пользователя его имя. В дальнейшем при отсылке сообщений на сервер должен использоваться следующий шаблон: «Имя пользователя: текст»
* До этого момента сообщения между сервером и клиентом — обычный текст. Это с трудом позволяет передавать так же сопроводительную информацию например: имя пользователя, дату приёма сообщения сервером и т.д. В этом задании необходимо перевести обмен информации на использование [Serialization](http://java.sun.com/developer/technicalArticles/Programming/serialization/). Т.е. обмен информацией между клиентом и сервером должно производится посредством сериализованного класса Message, который в свою очередь содержит помимо текста дополнительную атрибутику: дата создания сообщения, имя пользователя, IP отправителя, режим в котором находится отправитель (как в аське: сплю, ем, работаю:) )
* Дописать серверную часть таким образом, чтобы она следила за количеством подключённых пользователей и не позволяла подключаться большему количеству пользователей нежели 10
* На данный момент многие параметры сервера прописаны в коде самой программы, например: порт на котором открывать сервер, количество сообщений выдаваемых при подключении, максимальное количество подключённых пользователей и т.д. Перенести все эти конфигурационные параметры в XML файл.
* Сервер должен позволять подключаться только тем пользователям, которые знают соответствующий пароль для их имени пользователя. Для этого при запуске клиент должен спрашивать так же и пароль. А сервер при подключении нового пользователя должен сверять имя пользователя/пароль с имеющимися данными в конфигурационном файле. Если пароль не подходит, или такой пользователь не существует, то сервер должен отключить клиента с соответствующим сообщением.

Задача: Java Command Line (JCL)

Данная задача обычно дается мной в качестве зачетной по всему курсу J2SDK.  
Реализовать консольную программу на Java, которая бы представляла некую реализацию командной строки, то есть могла бы выполнять команды вводимые построчно пользователем. Под командой понимается следующая строка:  
"имя команды" "аргумент №1" "аргумент №2" ... "аргумент №N"  
Код, который выполняет необходимую команду пользователя, должен оформляться как отдельный Java класс. Соответствие между классом и именем команды должно задаваться в конфигурационном (XML) файле программы. Команда во время выполнения может так же взаимодействовать с пользователем используя стандартный ввод/вывод.  
Программа должна поддерживать следующие команды:

* dir — выводит список файлов в текущей директории
* cd «путь» — перейти в директорию, путь к которой задан первым аргументом
* pwd — вывести полный путь до текущей директории

Варианты усложнения:

1. Программа должна поддерживать команду "! имя\_системной\_программы аргумент№1… аргумент№N", которая запускает системную программу с соответствующими аргументами — так же ввод JCL должен подаваться на вход системной программы и тоже самое с ее выводом
2. Программа должна поддерживать следующий синтаксис: «команда1 аргументы && команда2 аргументы» и «команда1 аргументы || команда2 аргументы». В первом случае «команда2» запускается только если «команда1» выполнилась успешно. Во втором случае «команда2» запускается только тогда, когда выполнение «команда1» завершилось не успешно
3. Программа должна поддерживать возможность запускать команды в фоновом режиме. Для этого достаточно в конце командной строчки ввести знак "&". Так же программа должна поддерживать команду «jobs», которая выводит список задач, которые выполняются в фоне
4. Реализовать возможность запускать программу в сетевом варианте: т.е. программа открывает порт и работать с ней можно используя обычный telnet с отдалённой машины. Программа должна поддерживать подключение сразу нескольких пользователей, а так же команды who и write, которые показывают кто подключён и посылают сообщение всем соответственно.

№1: Написать функцию int atoi(String), для парсинга чисел в -7-ичной записи (основание системы счисления — минус семь). Спрашивают можно ли написать функцию format() для этой системы счисления (надо объяснить почему).  
№2: Написать класс «синглетон». А потом найти ошибку (все забывают synchronized).   
№3: Написать редиректор потока с методом redirect( InputStream sfrom, OutputStream sto ) (внутри надо создать поток и догадаться его закрыть вовремя). Предложить использовать это для управления консольным приложением (читаем out-пишем in по строчкам). Запустить под линь (в их случае правда был AIX) и объяснить почему не работает (кэширование труб в ОС)  
№4: Дают двухтердовое приложение, которое хватает дедлок. Предлагают найти проблему.  
№5: Дают аппликуху которая выносит яву по нехватке памяти. Делается это путем создания большого графа ссылок. Предлагают объяснить что случилось и «поправить». Проблема как я понял была в том, что gc не успевал прошуршать по всему графу, а аппликуха создавала уже новый.  
№6: Есть лог (w3c, например), надо заменить все похожие строчки идущие подряд на одну. Типа 01/01/2001 login bob\n 01/01/2001 login sam \n заменить на 01/01/2001 login \*\*\*\* (x 2). Смотрят на то, как пытаешься сформулировать «похожесть» и как пытаешься ее вычислить.  
№7: Дают интерфейс 3-мерного куба (get(x,y,z)/set(x,y,z),sizeX(),sizeY(),sizeZ()). Предлагают сделать реализацию. Потом с ней построить кубик 1Mx1Mx1M, заполнить в нем 2M точек целыми значениями. Затем каждую точку куба заменить на среднее арифметическое точек в округе. Если делаешь трехмерный массив предлагают увеличить сторону куба до 10^10. Хотят увидеть объектное моделирование и аналог two phase commit (чтобы точки расчитанные на i-m шаге НЕ влияли друг на друга).  
№8: Просят написать кэширующий калькулятор. Типа ему дают арифметические выражения с переменными и без скобок и (a+b-c\*d/e), он должен найти в кэше подвыражения (a+b и c\*d/e) и сделать расчет с использованием результатов выражений. Хотят увидеть класс Expression, который умеет парсить выражения, приводить их к нормальному виду и делать хитрые compareTo(), hashCode() и equals() чтобы Map с ним работал.  
№9: Просят сделать переменное количество вложенных циклов (дали N-мерный массив каждую точку заполнить суммой ее координат). Хотят увидеть злобный итератор. Стек и рекурсию не принимают.

## Глава 21. Java best practices.

By book of Joshua Bloh. Effective Java.

У каждого языка программирование есть свои «лучшие практики» (best practices). Есть такие и у Java. Практически все они собраны и описаны товарищем Блохом в своей знаменитой книге. Данный материал является переводом/компиляцией/интерпретацией этой книги (Effective JAVA (second edition) by Joshua Bloch). Оригинал книги можно найти в свободном доступе на сайте производителя JAVA – компании ORACLE.

**Создание и разрушение объектов.**

**Пункт 1. Рассматривайте (используйте) статические фабричные методы вместо конструкторов.**

Первое преимущество статических фабричных методов – они имеют значащие имена, которые могут описать создаваемый экземпляр класса (в отличие от конструктора).

Второе преимущество – в отличие от конструктора, статический фабричный метод не всегда при вызове может создавать новый объект.

Третье преимущество – в отличие от конструктора, такие методы могут возвращать объекты любого подтипа их возвращаемого типа объекта.

Четвертое преимущество статических фабричных методов – они уменьшают «многословность» при создании параметризованных экземпляров типов (подробнее см. саму книгу).

Однако, данные методы имеют и недостатки. Рассмотрим их.

Главный недостаток – класс без public или protected конструктора не может быть расширен. Второй недостаток – они не очень хорошо различимы среди других статических методов класса. Они не выделяются в API – искать их надо. Обычно для статических фабричных методов, создающих экземпляры класса, используются следующие имена (описание см. в книге):

* valueOf
* of
* getInstance
* newInstance
* getType
* newType

**Пункт 2. Рассматривайте «построитель» («builder»), когда сталкиваетесь с многопараметрическим конструктором.**

**Пункт 3. Реализовывайте свойство singleton с помощью private конструктора или перечислимого типа (enum).**

Синглтон можно реализовать несколькими разными способами.

**Пункт 4. Реализуйте свойство невозможности инстанцировать класс (noninstantiability) с помощью private конструктора.**

**Пункт 5. Избегайте создания ненужных (излишних) объектов.**

**Пункт 6. Исключайте (уничтожайте) устаревшие ссылки на объекты.**

**Пункт 7. Избегайте финализаторов (методов finalize()).**

**Пункт 8. Придерживайтесь основного контракта при переопределении метода equals().**

Вот сам основной контракт (см. спецификацию класса Object в доке java):

метод equals() реализует сравнение для не нулевых (non-null) ссылок на объекты. Свойства данного метода:

* Рефлексивность: для любой ненулевой ссылки на объект (объекта) x, выражение x.equals(x) должно возвращать true.
* Симметричность: для любых ненулевых объектов x и y, выражение x.equals(y) = true только тогда, когда выражение y.equals(x) = true.
* Транзитивность: для любых ненулевых объектов x, y и z, если x.equals(y) = true и y.equals(z) = true, то обязательно x.equals(z) = true.
* Последовательность: для любых ненулевых объектов x и y множественные вызовы x.equals(y) должны последовательно возвращать значение true (или false), если исходные объекты неизменны.
* Для любого ненулевого объекта x, x.equals(null) = false.

В самом классе Object метод equals() реализован самым взыскательным способом по отношению к эквивалентности объектов, то есть для любых ненулевых объектов x и y, этот метод возвращает истину, если и только если x и y относятся к одному объекту (х == у имеет значение true).

При переопределении метода equals() объекта ВСЕГДА необходимо переопределять метод hashCode()!

**Пункт 9. Всегда переопределяйте метод hashCode(), если переопределяется метод equals().**

**Пункт 10. Рекомендуется всегда переопределять метод toString().**

**Пункт 11. Переопределяйте метод clone() рассудительно ☺.**

**Пункт 12. Рассмотрите вопрос реализации интерфейса Comparable.**

**Пункт 13. Минимизируйте доступность классов и их полей/методов (членов класса – members).**

ИС (Exceptions).

67. Классы ИС и механизмы их использования необходимо использовать только для перехвата и обработки ошибок (исключительных ситуаций – ИС). ИС не должны использоваться для контроля потока выполнения программы – как конструкции типа if-else. Также правильно спроектированный API не должен заставлять своих клиентов (пользователей API) использовать ИС для контроля потока выполнения программы.

## Глава 22. JEE best practices.

В данном разделе собраны и обобщены некоторые «лучшие практики» для разработки JEE приложений. Исходным материалом данной главы являются:

* Professional Oracle Weblogic server 2009
* Ffffff

Непосредственно «лучшие практики»:

1. Сервлеты, обрабатывающие множественные параллельные запросы, должны быть потокобезопасны. Не создавайте переменных уровня класса (class) или объекта (instance), если не используется логика синхронизации (synchronized blocks).
2. Избегайте использования однопотоковых сервлетов (implements SingleThreadModel). Проектируйте сервлеты потокобезопасными и минимизируйте использование блоков синхронизации (эти блоки приводят к проблемам с производительностью).
3. Используйте механизм сессий для хранения состояния и контекстной информации между запросами к сервлету. При использовании сохранения сессий (session persistence) избегайте больших объектов в сессии, если приложение меняет только малую часть такого большого объекта для любого запроса. Вместо этого, используйте множество небольших «мелкозернистых» (fine-grained) объектов для снижения издержек на сохранение сессий.
4. Явно запрещайте отслеживание сессий для JSP страниц, где сессии не используются. Это поможет избежать ненужного сохранения сессий (session persistence). Выполняется следующим образом: на JSP странице (в начале) указывается директива <%@ page session=”false” %>
5. Избегайте однопоточных JSP страниц. Потоконебезопасный код должен быть в каком-либо другом java классе и контролироваться блоками синхронизации. Указание того, что JSP страница должна быть однопоточной: <%@ page isThreadSafe=”false”%> (результирующий сервлет будет реализовывать интерфейс SingleThreadModel).
6. Всегда используйте буферизацию вывода в JSP страницах. Размер буфера увеличивайте, как минимум, до 32Kb, это поможет избежать проблем с редиректами, кукисами, jsp:forward директивами, страницами ошибок.
7. Создайте дружественную и полезные страницы ошибок

# Часть III. Инфраструктура. Continuos integration/delivery.

## Глава 23. Интегрированные среды разработки (IDE)

## IntelliJIDEA

## Eclipse

## NetBeans (6.9.1).

Несколько юзабельных советов, которые пригодятся при установке/работе с данной средой. Эти советы могут упростить жизнь.

* Для увеличения размеров шрифта интерфейса и диалоговых окон используется параметр командной строки (для запуска среды) [**--fontsize 14**].

## Глава 24. Средства автоматизации сборки.

Если наш проект будет немного больше, чем просто “Hello, World!”, то нам обязательно понадобятся средства его сборки. Почему же нельзя просто использовать IDE для сборки? Можно. Но в большом проекте, когда над кодом трудятся как минимум несколько разработчиков, у каждого может быть своя любимая IDE и он будет настаивать, чтобы использовалась именно она – представляете, что начнется? Помимо данного очевидного плюса внешние (по отношению к IDE) системы сборки имеют другие достоинтсва:

* независимость сборки проекта от используемой разработчиками IDE
* намного бОльшая гибкость сборки, по сравнению со встроенными возможностями СУБД
* возможность добавления различных вспомогательных действий, помимо сборки приложения (напр., прогон тестов перед окончательной сборкой)
* возможность использования сборочного скрипта/механизма с системами постоянной интеграции (CI)

## Apache Ant.

Классика жанра – скриптовая система сборки (конфиг в виде XML). По аналогии, можно сказать, что похожа на кроссплатформенный batch, система написана на Java. В основе системы лежит файл конфигурации сборки, в котором последовательно описаны задачи для выполнения. Файл изначально создается пользователем. Система очень гибкая, никаких установок по умолчанию (кроме имени сборочного файла – по умолчанию это build.xml, но это тоже можно изменить)

### Установка и конфигурирование

Все предельно просто:

* скачиваем архив с сайта Анта: <http://ant.apache.org>
* распаковываем в нужный каталог – это установка
* необходимо установить переменную окружения ANT\_HOME – она должна указывать на каталог, куда установлен (распакован) Ант
* путь ANT\_HOME\bin необходимо добавить к переменной PATH – для запуска Анта из командной строки из любого каталога (для Windows-систем)

Наш Ант установлен и готов к работе. Проверить установку можно выполнив в командной строке команду [*ant -version]* - должна быть выведена версия установленного Анта.

## Apache Maven.

Такая же классика, как и Ант. Тоже скриптовая система сборки (конфиг также в виде XML), также написана на Java. Но есть кардинальное отличие – весь жизненный цикл сборки проекта зашит в систему как поведение по умолчанию. И для того, чтобы его изменить под себя, необходимо конфигурирование. Файл конфигурации сборки по умолчанию называется pom.xml, в минимальной конфигурации он содержит всего несколько тегов. Каждая фаза сборки управляется плагинами, дополнительные возможности также появляются с помощью добавления новых плагинов, какждый новый плагин «встраивается» в определенную фазу цикла сборки проекта.

Отличия Анта и Мавена хорошо передаются так:

*- можно ли то-то и то-то сделать в Анте? – Да! (без раздумий)*

*- а в Мавене? –Ну, надо посмотреть…*

### Установка и конфигурирование

Также, как и с Антом все достаточно просто:

* скачиваем архив с сайта Мавен: <http://maven.apache.org>
* распаковываем в нужный каталог – это установка
* необходимо установить переменную окружения M2\_HOME – она должна указывать на каталог, куда установлен (распакован) Мавен. Имя переменной одинаковое и для Мавен2, и для Мавен3.
* путь M2\_HOME\bin необходимо добавить к переменной PATH – для запуска Мавена из командной строки из любого каталога (для Windows-систем)
* конфигурирование Мавена осуществляется через основной конфигурационный файл Мавена - <MAVEN\_HOME>/conf/settings.xml. В этом файле можно сконфигурировать расположение локального репозитория Мавена (см. сам файл – там все понятно ☺).

Все, Мавен установлен и готов к работе. Проверить установку можно выполнив в командной строке команду [*mvn -version]* - должна быть выведена версия установленного Мавена.

### Работа с проектом

Координаты Maven. Координаты Maven — это набор взаимосвязанных значений, уникально определяющих любой артефакт. Координата включает в себя три компонента:

* ID группы **group ID**: Объект или организация, ответственная за производство артефакта. Например, com.ibm.devworks может быть ID группы.
* ID артефакта **artifact ID**: Имя данного артефакта. Например, проект с основным классом OpsImp может использовать OpsImp как ID артефакта.
* Версия **version**: Номер версии артефакта. Поддерживается форма записи формата *mmm.nnn.bbb-qqqqqqq-dd* , где *mmm* — это основной номер версии, *nnn* — это дополнительный номер версии, и *bbb* — это уровень коррекции ошибок. Необязательными являются поля *qqqqq* (спецификатор) и *dd* (номер сборки).

Создание проекта. Проект в Мавене создать достаточно просто, достаточно выполнить следующую команду:

*mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DinteractiveMode=false*

и система создаст простой проект. С ключом –D указываются опции создания проекта: groupId и artifactId – координаты Мавена (см. выше), interactiveMode – отключение интерактивного режима, т.е. для всех необходимых параметров будут взяты значения по умолчанию. Для дополнительных опций – см. доку к Мавену.

Репозиторий

Общее правило для расположения артефактов (библиотек) в репозитории Мавена:



Если groupId – полное доменное имя вида z.y.x, то мы увидим следующую структуру каталогов:



Жизненный цикл (проекта)

По умолчанию, жизненный цикл проекта (**Default lifecycle**) состоит из следующих основных фаз (полный список фаз смотри здесь <http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference>):

* **validate** – проверка проекта на корректность и доступность всей необходимой инфы.
* **compile** – компиляция исходников проекта.
* **test** – тестирование скомпилированного исходного кода. Тесты не должны использовать упакованный (packaged) или установленный (deployed) код.
* **package** – упаковка скомпилированного кода в формат для распространения (например JAR).
* **integration-test** – развертывание упакованного кода в окружении, где могут быть запущены интеграционные тесты.
* **verify** – запуск проверок для проверки пакета (package) на валидность и удовлетворение критериям качества.
* **install** – инсталляция пакета кода в локальный репозиторий, для того, чтобы он мог быть использован как зависимость в других проектах (локально).
* **deploy** – выполняется в окружении для релиза или интеграционных тестов, копирует конечный пакет в удаленный репозиторий для разделения с другими разработчиками/проектами.

Помимо **Default lifecycle**, у мавена есть еще пара (несколько?) жизненных циклов:

**Clean lifecycle.** Цикл очистки проекта от файлов предыдущей сборки.

* **pre-clean** – запуск служебных процессов перед очисткой.
* **clean** – удаление всех файлов, сгенерированных предыдущей сборкой.
* **post-clean** – запуск служебных процессов для завершения очистки.

**Site lifecycle.** Цикл генерации сайта проекта.

* **pre-site** - запуск служебных процессов перед генерацией сайта.
* **site** – генерация сайта документации проекта.
* **post-site** - запуск служебных процессов для завершения генерации сайта.
* **site-deploy** – установка (развертывание) сгенерированного сайта на указанный веб-сервер.

## Gradle

## Глава 25. Системы отслеживания ошибок (bug trackers)

Вот что говорит по поводу данных систем Википедия:

**система отслеживания ошибок** — прикладная программа, разработанная с целью помочь разработчикам программного обеспечения учитывать и контролировать ошибки, найденные в программах, пожелания пользователей, а также следить за процессом устранения этих ошибок и выполнения или невыполнения пожеланий ([http://ru.wikipedia.org/wiki/Багтрекер](http://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3%D1%82%D1%80%D0%B5%D0%BA%D0%B5%D1%80)).

## JTrac.

Простая в развертывании/использовании/обслуживании система, распространяемая бесплатно. Написана на Java, использует встроенный jee контейнер Jetty и базу данных HSQLDB. Легка в развертывании (установке) на любой платформе – windows/\*nix (java - кроссплатформенность).

На данный момент (12.11.2013) активно не разрабатывается, новые версии не выходят, однако доступна для скачивания и работоспособна.

Я использую данную систему в проекте МЕС Петросталь, в течение 2 месяцев полет нормальный, «из коробки» работает вся функциональность.

#### Установка и запуск.

Вся установка заключается в скачивании zip-архива и его распаковке в нужный каталог. Архив доступен здесь: <http://www.jtrac.info> Здесь же можно найти документацию по системе. Некоторые важные параметры (например, порт, на который вешается система) можно быстро поменять в скрипте запуска (start.bat). По умолчанию используется порт 80. После распаковки и запуска система доступна по адресу:

**Ошибка! Недопустимый объект гиперссылки.**, учетка админа по умолчанию: admin/admin.

Система запускается вручную, с помощью батника. Также возможно установить систему как сервис ОС для автозапуска (см. в интере).

#### Backup/restore.

Система достаточно стабильна и надежна, но подстраховаться не помешает.

**Резервирование.**

Данный процесс очень прост и не требует никаких специальных навыков.

1. В самом простом случае, когда используется встроенная БД HSQL, можно просто забекапить весь каталог с трекером (полностью) – это займет примерно 65+ Мб (в зависимости от количества и размеров файлов аттачей).
2. Другой простой вариант (также при использовании встроенной БД HSQL) – резервирование содержимого каталога jtrac.home – при использовании каталогов по умолчанию, это каталог jtrac/data.
3. Если же используется сторонняя СУБД (напр., mysql), то ндо отдельно резервировать каталог jtrac.home и саму СУБД (ее средствами).

**Восстановление.**

Процесс восстановления не более сложен, чем процесс резервирования. Также по пунктам (соответственно пунктам резервирования):

1. Берем архив с полным резервированным каталогом трекера и просто его разворачиваем и запускаем. Вауля! Сохраняются все настройки/аккаунты/данные и т.д.
2. Скачиваем (или берем из заначки) чистую инсталляцию трекера (архив), распаковываем, копируем в нее (из резерва/архива) каталог jtrac.home, запускаем – также видим все настройки/данные/аккаунты. Профит!
3. Берем чистую инсталляцию, копируем в нее из резерва каталог jtrac.home, восстанавливаем используемую стороннюю СУБД, запускаем – все должно быть ОК.

Все проще некуда. На мой взгляд, возможностей встроенной СУБД HSQL более, чем достаточно для небольших проектов. Остальную инфу можно найти на сайте системы (см. выше в пункте «Установка»).

### JIRA

Одна из самых мощных (наверное, самая мощная) из существующих на данный момент систем багтрекинга. Система коммерческая, написана на Java.

### BugZilla

Наверное, самая популярная (из бесплатных/freeware) систем багтрекинга. Написана на PHP, лучше всего ставить ее на ОС \*nix.

## Глава 26. Системы постоянной интеграции (continuous integration)

### Hudson

# 

# Часть IV. Enterprise world (Java EE/DBMS/etc.).

## Глава 27. CDI

TBD

## Глава 28. «Легкие» JEE контейнеры (Tomcat/Jetty/Resin).

### Tomcat

### Jetty

### Resin

## Глава 29. Промышленный JEE контейнер WebLogic.

### Основные понятия

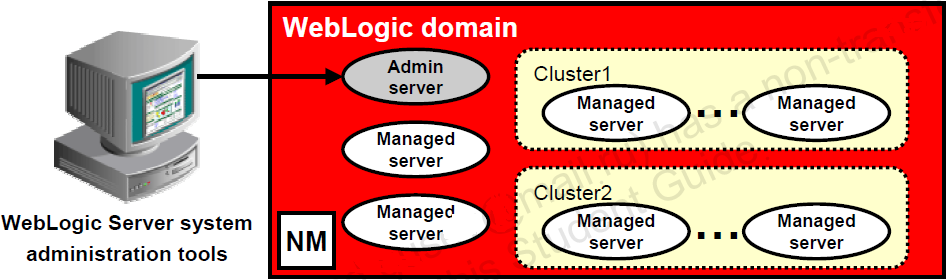
#### Suite/Fusion Middleware

В терминах Oracle – набор ПО (Suite) или набор ПО промежуточного уровня (Fusion Middleware). У оракла есть много разных сьютов и мидлверов – наборов софта.

#### Домен

Домен – это базовая административная единица (unit) для WL. Он состоит из одного или более инстансов WL и логически связанных ресурсов и сервисов, которые управляются как один юнит. Домен включает в себя один админ сервер (всегда), менеджед сервера (0 и более), может включать консоль администрирования (веб приложение на админском сервере) и утилиту Node Manager (NM). Если в домене всего один сервер WL, то он является админ и менеджед сервером одновременно. Домен может включать в себя кластер (0 и более кластеров).

Все менеджед сервера в домене должны иметь одинаковую версию WL, админ сервер может иметь версию старше.



#### Node manager

NM – java программа для удаленного запуска/останова/рестарта админ сервера и менеджед серверов. Также эта программа мониторит сервера и может их автоматически рестартовать при неожиданном останове. Обычно устанавливается в виде службы (вин) или демона (никс) в автозапуск системы.

#### Cluster

Группа инстансов сервера WL (админ и менеджед сервера), работающих совместно для предоставления масштабируемости и высокой доступности. Для клиента, кластер выглядит как один инстанс WL. Может иметь одну точку входа, в качестве которой могут выступать: Oracle HTTP Server, WL proxy servlet (специальный сервлет, поставляемый с WL), Oracle Web Cache, DNS, web-server (почти любой – для которого есть специальный модуль).

#### Admin Server

Сервер управления доменом, в каждом домене всегда есть один админ сервер. На данном сервере располагается веб консоль управления WL (веб приложение, задеплоенное на админ сервер). На этот сервер можно деплоить приложения, но делать это не стоит.

#### Managed Server

Сервер для публикации приложений и обслуживания клиентов, рабочий узел домена WL.

## Где и что находится (скрипты и прочее)

Определимся с терминами:

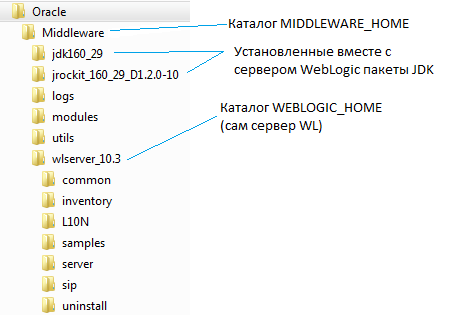
* MW\_HOME – каталог установки пакета софта Oracle Middleware. Кроме WL там может быть и любой другой софт Oracle (и не только).
* WL\_HOME – каталог установки сервера WL (обычно находится внутри MW\_HOME).
* DOMAINS\_HOME – каталог, указанный при создании доменов как хранилище для файлов доменов. Каждый домен лежит в своем каталоге.
* DOMAIN\_HOME – каталог конкретного домена

Непосредственно расположение и назначение различных утилит/скриптов, поставляемых с WL.

* <WL\_HOME>/common/bin/config.cmd (.sh) - скрипт конфигуратора домена. Запуск без параметров – режим графического мастера, запуск с ключом -mode=console запустит мастер в консольном режиме.
* <WL\_HOME>/server/bin/setWLSEnv.cmd (.sh) – установка переменных окружения для консольных утилит WL.

## Установка сервера WL

### Структура каталогов

После установки сервера WL структура каталогов будет похожа на показанную на рисунке (структура приведена для ОС Windows 7, на юникс/линукс машинах структура будет аналогична). В данном случае установлен только сервер WL, при наличии другого ПО от Oracle возможно появление других каталогов. Немного подробнее каталоги с описанием приведены ниже. Для middleware home указан стандартный каталог, он может быть другим.

#### Общая структура каталогов

* c:/oracle/middleware -> <MW\_HOME>. Этот каталог на машине с виндой, на никсах каталог будет другим.
* <MW\_HOME>/wlserver\_10.3 -> <WL\_HOME>. Каталог установки WL.
* <MW\_HOME>/jdk160\_29 – пакет Sun JDK
* <MW\_HOME>/jrockit\_160\_29\_xxxxxxx – пакет JRockit JDK.
* <MW\_HOME>/logs – логи сервера
* <MW\_HOME>/<domain\_folder>/<domainXXXX> - каталоги с доменами WL обычно располагаются внутри <MW\_HOME>, каждый домен в своем каталоге
* <WL\_HOME>/common – разделяемые (shared) компоненты сервера
* <WL\_HOME>/common/bin - исполняемые модули и скрипты общего назначения (для доменов например)
* <WL\_HOME>/server – компоненты ПО сервера WL
* <WL\_HOME>/server/bin – различные исполняемые модули и скрипты сервера

#### Структура каталогов домена

<DOMAIN\_HOME> - каталог, в котором находится домен, все каталоги будут указаны относительно него.

* <DOMAIN\_HOME>/autodeploy – папка для автодеплоя приложений.
* <DOMAIN\_HOME>/bin – скрипты домена (запуск/останов и т.п.)
* <DOMAIN\_HOME>/config – конфигурация домена. Основной файл конфига домена – config.xml – также лежит в этом домене.
* <DOMAIN\_HOME>/servers – данные серверов (админ и менеджед), входящих в данный домен.
* <DOMAIN\_HOME>/servers/domain\_bak – резервная копия конфигов домена

### Установка

Установка может быть выполнена:

* в графическом режиме (интерактивно) – запускаем инсталлятор и отвечаем на вопросы (win/unix)
* в режиме командной строки (интерактивно) – установка в текстовом режиме, также отвечаем на вопросы (win/unix)
* в «тихом» режиме командной строки (автоматически) – требуется заранее подготовленный XML файл с ответами на вопросы (win/unix)

WebLogic позиционируется как часть Oracle Fusion Middleware – ПО промежуточного уровня, поэтому устанавливается в т.н. middleware home каталог – каталог всего софта Oracle Middleware. Если уже есть какой-то middleware софт от oracle – ставим WL туда же (это необязательно, но желательно – для порядка), если нет – будет создан новый middleware home каталог. Понятие MIDDLEWARE\_HOME (MW\_HOME) очень важно, также надо понимать, что WEBLOGIC\_HOME (WL\_HOME) – это другой каталог (возможно внутри MW\_HOME).

### Установка в графическом режиме.

Запускаем инсталлятор и отвечаем на несложные вопросы. Основные вопросы:

* выбор каталога установки (см. выше про MW\_HOME)
* можно (нужно) отказаться от security updates – будут страшные предупреждения, но все ок
* выбор используемой JVM – возможно установить JRockit от Oracle (при установке WL) или использовать локально установленную другую JVM
* установка Node Manager – да, устанавливать и регистрировать как сервис в ОС

Остальные вопросы интуитивно понятны.

### Установка в консольном режиме, установка в «тихом» режиме.

#### Консольный режим.

Установка выполняется из консоли (win/\*nix) с добавлением ключа -mode=console :

**<install\_media\_name>.<exe|bin> -mode=console**

Будут заданы те же вопросы, что и при использовании графического инсталлятора.

#### «Тихий» (silent) режим установки.

Необходимо заранее подготовить файл silent.xml с ответами на все вопросы, пример есть в документации.

### Установка переменных окружения

Для установки необходимых переменных окружения можно использовать скрипт

<WL\_HOME>/server/bin/setWLSEnv.cmd (или .sh) Установка переменных окружения будет действительна для текущего сеанса пользователя. Все консольные команды (для сервера WL должны предваряться запуском данного скрипта).

## Создание/конфигурирование домена

Домен – общее окружение приложения (total application environment). Перед тем, как что-либо настраивать на сервере или использовать его, необходимо сконфигурировать домен (создать новый, настроить имеющийся, создать на основе шаблона).

Конфигурирование домена достаточно сложная процедура и она вовлекает в себя множество понятий и действий, часть из них будет объяснена тут.

Скрипты для запуска конфигуратора домена: <WL\_HOME>/common/bin/config.cmd (или .sh для никсов). Простой запуск скрипта вызовет графический конфигуратор, запуск скрипта из консоли с параметром

**-mode=console**

запустит конфигуратор в консольном режиме.

### Вопросы, на которые необходим ответ в процессе настройки домена

* Создание нового или модификация существующего? Выбираем сами.
* Генерация домена для поддержки перечисленных продуктов (список) или создание на основе шаблона домена (если он есть)? При генерации по умолчанию выбран базовый домен WL, опционально можно добавить еще пункты (особо не надо – см. по ситуации).
* Имя домена и расположение файлов домена, логин/пароль админа (мои по умолчанию weblogic/weblogic1)
* Режим домена: development/production. Отличий не очень много, но они есть (в продакшене нет автодеплоя и т.п. – см. доку). Также выбираем используемую JVM для домена.
* Выбор дополнительной конфигурации – можно сразу почти полностью настроить весь домен – admin server, managed server(s), cluster(s), machine(s), rdbms. Рекомендую сразу настроить админ сервер и менеджед.
* Конфиг admin server’a.
  + - name
    - listen address – адрес, на котором будет работать админ сервер (отвечать). Если указать конкретный адрес, то именно он и будет использован (напр.: укажем 127.0.0.1, localhost – не будет отвечать; укажем все (all local addresses) – будут отвечать все; можем указать свой)
    - port – должен быть уникальным
    - SSL – если необходимо
  + Конфиг managed server. Добавляем столько серверов, сколько нам надо. Они все будут располагаться на данной физической машине (компе). Параметры аналогичны админ серверу – name, address, port, ssl.
  + Конфиг кластера (cluster).
* name
* cluster messaging mode – лучше всего установить unicast (проще). Подробнее – см. доку.
* Добавление серверов в кластер. Мы увидим все ранее сконфигурированные managed серверы (если они есть) и сможем добавить их в кластер.
* Конфиг машины (machine). В данном случае машина – виртуальная сущность (см. выше описание).
* Выбор Machine (не никс, винда например) или Unix Machine (ясно)
* Name
* Адрес и порт для Node Manager (порт был указан при инсталляции сервера WL, используем его)
* Связывание машины (логической) с настроенными северами (админ и менеджед).
* Просмотр саммари. Можно вернуться назад и что-нибудь исправить. После создания домена и связанных с ним данных можно сразу стартовать админ сервер. После его старта будет доступна админская консоль, из которой уже можно стартовать менеджед сервера (при наличии NodeManager'a).

WebLogic JAR builder tool (создание jar-файла для использования в standalone client application): <http://download.oracle.com/docs/cd/E12840_01/wls/docs103/client/jarbuilder.html>

## Глава 30. СУБД ORACLE

СУБД распространяется свободно (под лицензией OTN – Oracle Technical Network), но только для целей разработки (как и почти все продукты Оракл). СУБД достаточно тяжеловесна и рекомендую ставить ее только на сервера.

Для разработки (на машине разработчика или на выделенном сервере) можно использовать бесплатную версию СУБД - Oracle XE (eXpress Edition). Эта версия с некоторыми ограничениями, но вполне подходит для разработки и тестирования, также подойдет и для простых проектов.

## Утилиты для работы с СУБД

Для работы с СУБД Оракл существует большое количество различных утилит, эти утилиты являются клиентами, по отношению к СУБД. Часть утилит входит в поставку СУБД, часть можно поставить отдельно. Далее о некоторых их утилит.

### Sql\*plus

Утилита командной строки для соединения с локальным/удаленным СУБД и выполнения sql-запросов. Утилита может быть установлена:

* Вместе с сервером БД (обычной версией или XE)
* В составе Instant client
* В составе полного клиента СУБД

#### Соединение с локальным сервером

В командной строке вводим *sqlplus* – происходит соединение. В процессе соединения будут запрошены логин и пароль (интерактивный режим). Параметры соединения (имя БД/схемы, логин, пароль) можно указать как параметры командной строки для sqlplus:

*sqlplus login@connect\_identifier* (будет запрошен пароль)

*sqlplus login/password@connect\_identifier* (пароль указан как параметр)

или после запуска sqlplus с помощью опции CONNECT (предполагаем, что sqlplus уже запущен):

*CONNECT login@connect\_identifier* (будет запрошен пароль)

*CONNECT login/password@connect\_identifier* (пароль указан как параметр)

В данных примерах connect\_identifier – Oracle Net идентификатор соединения (см. ниже).

#### Соединение с удаленным сервером

Также в командной строке вводим *sqlplus* – происходит соединение. Если необходимо подключиться к удаленной системе при старте sqlplus, то используем параметры командной строки:

*sqlplus username@[//]host[:port][/service\_name][:server][/instance\_name]*

*sqlplus username/password@[//]host[:port][/service\_name][:server][/instance\_name]*

Если sqlplus уже запущен, то можно также использовать опцию CONNECT:

*CONNECT username@[//]host[:port][/service\_name][:server][/instance\_name]*

*CONNECT username/password@[//]host[:port][/service\_name][:server][/instance\_name]*

Строка после символа @ - это и есть Oracle Net идентификатор соединения.

Для соединения с СУБД с помощью sqlplus нет необходимости устанавливать полный клиент Оракл, можно установить легковесный Instant client. Он легко скачивается с сайта Оракл в виде архива и просто распаковывается как каталог на диск – состоит из двух частей: базовая (основные библиотеки) и непосредственно sqlplus (исполняемый файл sqlplus) – см. на сайте Оракла. При этом нет необходимости в файле конфигурации -  tnsnames.ora.

### Instant client

Небольшой набор клиентских утилит/библиотек, не требующий установки и предоставляющий основные возможности для работы с СУБД (в качестве клиента). На данный момент (12.11.2013) состоит из двух частей:

* Базовый набор библиотек Instant client – общие библиотеки для всех клиентских утилит.
* Архив с конкретной утилитой/драйвером для работы с СУБД.

Например, для работы с БД с помощью sql\*plus необходимы два архива (базовый набор и непосредственно архив с утилитой sql\*plus), распакованные/установленные в один каталог.

*Instant client не включает в себя утилиты импорта/экспорта баз данных Оракл (imp/exp или impdp/expdp)! Утилиты импорта/экспорта включаются в любую установку сервера Оракл – обычный сервер, сервер XE (eXpress Edition).*

Для пользования этими утилитами необходимо установить полный клиент СУБД Оракл.

### Full client

### Sqldeveloper

### Утилиты импорта/экспорта баз данных (imp/exp или impdp/expdp)

## FAQ по утилитам Оракл

1. **Установка языка (локали) для утилит командной строки.**

Установленный клиент Оракл (Instant client или Full client) на машину с русифицированной ОС Windows (у меня была 7-ка 64 бит) использует для утилит командной строки язык по умолчанию – русский. В консоли Виндовс русские буквы рисуются «кракозябрами». Чтобы это исправить, есть много способов. Один из них: установка переменных окружения NLS\_LANG и COREID\_NLS\_LANG в значения американских локали и кодировки:

NLS\_LANG=AMERICAN\_AMERICA.US7ASCII

COREID\_NLS\_LANG=AMERICAN\_AMERICA.US7ASCII

Эти настройки также укажут серверу баз данных, что данный клиент использует американские локаль и кодировку.

1. **Соединение с Oracle XE из Sqldeveloper (ошибка соединения).**

Часто возникает проблема соединения с только что установленным сервером БД Oracle XE с помощью утилиты Sqldeveloper - возникает следующая ошибка:

*An error was encountered performing the requested operation:*

*ORA-00604: error occurred at recursive SQL level 1*

*ORA-12705: Cannot access NLS data files or invalid environment specified*

Чаще всего эта ошибка возникает на русифицированных ОС семейства Виндовс. Для того, чтобы исправить ситуацию, необходимо указать утилите Sqldeveloper, что мы будем использовать английский язык. Сделать это можно, добавив следующие две строчки в файл конфига sqldeveloper'a:

AddVMOption -Duser.region=us

AddVMOption -Duser.language=en

Файл конфига sqldeveloper'a: {sql\_developer\_home}\ide\bin\ide.conf

## Глава 31. СУБД MySql

В данной главе приведены лишь основы работы с СУБД MySql. Данная СУБД снабжена неплохой документацией, которая поможет практически во всех случаях. В особо сложных случаях – воспользуйтесь опытом сообщества (community forums).

Для выполнения большинства действий с СУБД используется язык SQL. Запросы можно выполнять как из командной строки, с помощью утилиты mysql (входит в поставку сервера), так и через графическую оболочку Workbench (может быть установлена вместе с сервером или отдельно).

## Установка

СУБД распространяется в двух вариантах: Enterprise (коммерческий) и Community (бесплатный). Для разработки и небольших проектов вполне подходит вариант Community.

### Пользователи

Пользователь СУБД записывается (указывается) в виде <username>@<hostname>.

Часть <hostname> можно опустить, тогда при обращении к СУБД по умолчанию подставляется значение localhost: <username>@localhost. Существуют специальное значения для hostname = %, означает «любой хост»: myuser@% - пользователь myuser, который может обращаться с любого хоста.

Если при обращении к СУБД не указан и пользователь тоже – по умолчанию используются значения: ODBC (Windows системы) или <login name> (Unix системы).

#### Создание пользователя

Для создания пользователя используется следующий запрос:

create user [username] identified by [password];

при этом username и password указываются в одинарных кавычках.

#### Добавление прав пользователю

Для добавления прав пользователю используется запрос grant:

grant all on [db\_name].[db\_object] to [username] identified by [password];

в данном запросе:

* all – в данном запросе выдаются все права, можно выдавать только некоторые права (см. документацию). Также выдача прав может быть с «правом передачи» - with grant option (см. документацию)
* db\_name – указание БД
* db\_object – объект в БД, на который выдаются права, можно указать значение \* - означает всю БД целиком
* username – пользователь в формате <username>@<hostname>
* password – пароль пользователя

#### Обновление данных о правах

Для того, чтобы изменения прав сразу же вступили в силу, необходимо выполнить запрос:

flush privileges;

#### Получить список пользователей

Для получения списка пользователей, надо выполнить следующий запрос:

select user from mysql.user;

### Установка с помощью GUI инсталлятора.

Установка достаточно проста – скачиваем необходимый инсталлятор и запускаем его. Выбираем необходимые компоненты, пароль пользователя root (главный пользователь СУБД), при необходимости – конфигурируем нужные параметры по ходу инсталляции. Никаких «сюрпризов» быть не должно. При установке есть возможность сразу добавить необходимых пользователей.

### Установка из командной строки.

(добавлю позднее)

## Перенос БД (import/export)

Если есть необходимость перенести уже существующую БД с другого сервера mysql, то можно воспользоваться механизмом «импорт/экспорт».

1. Экспорт. Делаем «дамп» необходимой БД с работающего сервера. Если мы находимся на том же хосте, где работает СУБД, то можно воспользоваться командой:

mysqldump -u [username] -p [database\_name] > [dumpfilename.sql]

Если ситуация сложнее – смотри хелп к команде mysqldump.

1. Импорт. Перед импортом БД на другой сервер на этом сервере (куда импортируем) необходимо создать целевую БД:

create database [db\_name] character set [set\_name] collate [collate\_name];

Указание character set и collate – необязательно.

1. Импорт. Для выполнения импорта БД на сервер используется команда:

mysql -u [username] -p [database\_name] < [dumpfilename.sql]

## Глава 32. СУБД PostgreSql.

## Установка

При установке возможны проблемы на ОС Windows 2003 Server. Описание тут:

<http://forums.enterprisedb.com/posts/list/3984.page>

## Настройка

После установки сразу же возможно подключиться к СУБД с локальной машины. Если необходимо подключение по сети, необходимо настроить СУБД для приема подключений – для этого служит файл pg\_hba.conf, расположенный в каталоге [PostgreSQL Data Catalog], который был настроен при установке сервера. В указанный файл надо добавить строку

*host all all all md5*

Это позволит соединяться с сервером удаленно. Подробнее – см. документацию, раздел Client Authentication.

# Глава XX. Статический анализатор кода Sonar

**!!! move it to Infrastructure chapter**

Статический анализатор исходного кода предназначен для автоматизированного контроля качества кода.

## Архитектура Сонара

(картинка)

## Установка сервера Сонар

Сонар распространяется в виде zip-архива и установка сводится к скачиванию этого архива и распаковыванию в нужный каталог, будем называть этот каталог SONAR\_HOME. После этого необходимо сконфигурировать Сонар. Хорошая статья по установке и первоначальной настройке Сонара есть здесь: <http://mamikon.net/blog/installing-sonarqube-ubuntu/>

Далее описан процесс установки и настройки Сонара вкратце:

1. По умолчанию Сонар использует встроенную СУБД Apache Derby, однако ее лучше заменить на полноценную, сам Сонар об этом напомнит. Драйвера (JDBC) для наиболее популярных СУБД идут в комплекте с Сонаром. Необходимо выбрать СУБД.
2. После выбора СУБД для Сонара – устанавливаем ее. Наиболее простой вариант (IMHO) – СУБД MySql. Ее и будем использовать. О СУБД MySql см. соответствующую главу.
3. Создаем БД для Сонара, пусть БД называется sonar:

***CREATE DATABASE sonar CHARACTER SET utf8 COLLATE utf8\_general\_ci;***

1. Создаем пользователя СУБД для Сонара. В нашем случае пусть это будет sonar/sonar:

***CREATE USER 'sonar' IDENTIFIED BY 'sonar';***

***GRANT ALL ON sonar.\* TO 'sonar'@'%' IDENTIFIED BY 'sonar';***

***GRANT ALL ON sonar.\* TO 'sonar'@'localhost' IDENTIFIED BY 'sonar';***

***FLUSH PRIVILEGES;***

1. Необходимо скачать и установить дистрибутив Сонара. Дистрибутив Сонара – обычный zip-архив, который распаковывается в целевой каталог. В ОС Linux я обычно ставлю Сонар в /opt
2. После установки необходимо настроить сервер Сонар. Делается это с помощью конфига [SONAR\_HOME]/conf/sonar.properties. В данном файле настраиваются практически все аспекты работы Сонара (настройки сервера, настройки соединения с СУБД, другие настройки):

* Соединение с СУБД. Необходимо найти и настроить (раскомментировать) параметры подключения к СУБД MySql:

***sonar.jdbc.username=sonar***

***sonar.jdbc.password=sonar***

***sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8&rewriteBatchedStatements=true&useConfigs=maxPerformance***

* Настройка веб-сервера Сонар:

***sonar.web.host=127.0.0.1***

***sonar.web.context=/sonar***

***sonar.web.port=9000***

При этом рекомендуется указать контекст Сонара (context), остальные параметры можно оставить по умолчанию (port будет использован 9000, host – текущий).

1. После настройки Сонар надо запустить. Вручную это можно сделать путем запуска соответствующего скрипта из каталога [SONAR\_HOME]\bin\<platform>\<script> (каталог <platform> выбираем в зависимости от текущей платформы, <script> - выбираем из представленных скриптов). После успешного запуска переходим по URL: http://<host>:<port>/<sonar\_context> и убеждаемся, что наш сервер работает (host – настройки текущей платформы, port/context – настройки из sonar.properties или по умолчанию).
2. **Опционально.** По умолчанию Сонар может работать на JRE, однако для production use лучше использовать JDK. Для настройки этого используется параметр sonar.web.javaOpts=-server в файле sonar.properties
3. **Опционально.** Можно явно задать JVM для использования Сонаром, если на машине установлено несколько JVM. Для этого служит параметр wrapper.java.command=[PATH\_TO\_JVM] в файле [SONAR\_HOME]\conf\wrapper.conf
4. **Опционально.** Каждый раз запускать сервер вручную неудобно, поэтому Сонар необходимо установить как службу в используемой ОС:

* Для ОС Windows используется соответствующий скрипт для установки службы Сонара (каталог [SONAR\_HOME]\bin\<platform> – скрипт выбирается в этом каталоге)
* Для ОС Linux (Ubuntu) для установки Сонара как службы и автостарта при старте системы необходимо выполнить следующее:
* Скопировать файл sonar.sh ([SONAR\_HOME]\bin\<platform>) в /etc/init.d/sonar

***sudo cp bin/linux-x86-64/sonar.sh /etc/init.d/sonar***

* Отредактировать файл /etc/init.d/sonar:
* Добавить строки

***SONAR\_HOME=[SONAR\_HOME]***

***PLATFORM=[PLATFORM]***

* Отредактировать следующие строки

***WRAPPER\_CMD="${SONAR\_HOME}/bin/${PLATFORM}/wrapper"***

***WRAPPER\_CONF="${SONAR\_HOME}/conf/wrapper.conf"***

***...***

***PIDDIR="/var/run"***

* Зарегистрировать Сонар как сервис в ОС:

***sudo update-rc.d -f sonar remove***

***sudo chmod 755 /etc/init.d/sonar***

***sudo update-rc.d sonar defaults***

Установленным сервисом Сонара можно управлять командой:

***sudo /etc/init.d/sonar <start|stop|restart>***

1. **Возможные проблемы.**

* Для ОС Linux (Ubuntu) необходимо убедиться, что файлы sonar.sh и wrapper из каталога [SONAR\_HOME]\bin\<platform> имеют атрибут x (запускаемый). При необходимости надо установить этот атрибут командой chmod (значение 755 – см. пример выше).

## Анализ кода

Для анализа кода используются разные способы – внешние утилиты.

### Анализ кода с помощью Sonar-runner (java)

#### Установка

Sonar-runner, как и сервер Сонар, распростарняется в виде zip-архива. Вся установка сводится к распаковке архива в нужный каталог, будем называть этот каталог RUNNER\_HOME. После установки раннер необходимо сконфигурировать. Вот необходимые шаги установки:

1. Скачать zip-архив сонар-раннера и распаковать в рабочий каталог. Для Linux я обычно использую каталог /opt.
2. **Для Linux.** Зайти в каталог [RUNNER\_HOME] и убедиться, что файл sonar-runner имеет атрибут x (запускаемый).
3. Необходимо отредактировать файл [RUNNER\_HOME]\conf\sonar-runner.properties:

***sonar.host.url=http://localhost:9000/sonar***

***sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8***

***sonar.jdbc.username=sonar***

***sonar.jdbc.password=sonar***

Используемые настройки зависят от конфигурации сервера Sonar (см. файл [SONAR\_HOME]\conf\sonar.properties используемого сервера).

1. Необходимо настроить окружение для раннера:

* **Для Windows.** Установить переменную окружения SONAR\_RUNNER=[RUNNER\_HOME] (указывает на каталог с установленным раннером). Добавить каталог [RUNNER\_HOME]\bin в список путей для поиска исполняемых файлов (добавить в переменную окружения PATH).
* **Для Linux.** В файл ~/.profile добавить:

***SONAR\_RUNNER=”/opt/sonar-runner-2.4”***

***PATH=”$SONAR\_RUNNER\_HOME/bin:$PATH”***

Для применения параметров возможно следует logout/login или перезагрузиться.

1. После настроки и применения параметров окружения проверяем раннер командой

***sonar-runner -h***

#### Запуск

Для запуска раннера нет препятствий, кроме одного – что он будет анализировать. Для того, чтобы указать раннеру проект для анализа, необходимо в этом проекте создать файл \*.properties с указанием ключевых параметров проекта:

### Анализ кода с помощью Ant

### Анализ кода с помощью Maven

**Механизм переопределения признанных стандартов Java (Java Endorsed Standards Override Mechanism – для краткости Java ESOM)**

Оригинал статьи находится тут (на 21.01.2011): **http://download.oracle.com/javase/6/docs/technotes/guides/standards/index.html**

Точность перевода относительная (но очень высокая! ☺), нижеприведенный текст скорее перевод-компиляция указанной статьи. Итак…

Время от времени необходимо обновлять используемую платформу Java для того, чтобы включить в нее новые версии стандартов, созданных вне Java Community Process (Endorsed Standards – признанные стандарты) или для обновления версий технологий, включенных в платформу Java (в поставку JRE или JDK), для того, чтобы соответствовать последним автономным версиям этих технологий (Standalone Technologies).

*Небольшое пояснение: Java Community Process – это открытое сообщество для разработки стандартов и спецификаций для Java (см. список ссылок в конце).*

Endorsed Standards Override Mechanism – этот механизм как раз и предоставляет такую возможность. Как это реализовано технически? Очень просто! Пакеты классов Java (packages) должны быть запакованы в JAR-файлы. Системное свойство *java.endorsed.dirs* указывает один или несколько каталогов, в которых среда выполнения Java (Java Runtime) будет искать созданные JAR-файлы. Если в данном свойстве будут указаны несколько каталогов, то они должны разделяться символом File.pathSeparatorChar (см. javadoc) – для ОС Windows это обычно ; (точка-с-запятой). Если же значение указанного системного свойства не установлено, то Sun'овская реализация платформы Java будет искать эти JAR-файлы в расположении по умолчанию:

*<java-home>\lib\endorsed* (для Windows. Для \*никсов слеш будет обратный.)

В данном указании пути *<java-home>* указывает на каталог, в котором находится ПО Java Runtime (т.е. Java-машина для исполнения кода – JRE) – каталог верхнего уровня для JRE или каталог *jre* внутри каталога JDK. Java runtime будет использовать классы в найденных JAR-файлах вместо классов платформы Java.

Данный механизм (Java Endorsed Standards Override Mechanism) может использоваться:

* для переопределения всех классов и интерфейсов находящихся в нижеследующем списке пакетов Java (этот список классов и интерфейсов и составляет Java Endorsed Standards – одобренные стандарты Java):

*javax.rmi.CORBA*

*org.omg.CORBA*

*org.omg.CORBA.DynAnyPackage*

*org.omg.CORBA.ORBPackage*

*org.omg.CORBA.portable*

*org.omg.CORBA.TypeCodePackage*

*org.omg.CORBA\_2\_3*

*org.omg.CORBA\_2\_3.portable*

*org.omg.CosNaming*

*org.omg.CosNaming.NamingContextExtPackage*

*org.omg.CosNaming.NamingContextPackage*

*org.omg.Dynamic*

*org.omg.DynamicAny*

*org.omg.DynamicAny.DynAnyFactoryPackage*

*org.omg.DynamicAny.DynAnyPackage*

*org.omg.IOP*

*org.omg.IOP.CodecFactoryPackage*

*org.omg.IOP.CodecPackage*

*org.omg.Messaging*

*org.omg.PortableInterceptor*

*org.omg.PortableInterceptor.ORBInitInfoPackage*

*org.omg.PortableServer*

*org.omg.PortableServer.CurrentPackage*

*org.omg.PortableServer.POAManagerPackage*

*org.omg.PortableServer.POAPackage*

*org.omg.PortableServer.portable*

*org.omg.PortableServer.ServantLocatorPackage*

*org.omg.SendingContext*

*org.omg.stub.java.rmi*

*org.w3c.dom*

*org.xml.sax*

*org.xml.sax.ext*

*org.xml.sax.helpers*

Для переопределения стандартов, указанных в списке выше могут использоваться только версии новее (старше), чем используемые платформой Java по умолчанию.

Вдобавок к пакетам, описанным выше, которые являются частью **спецификации** Java SE, пользователи **реализации** Java SE от компании Sun (Sun’s Java SE Reference Implementation) могут использовать ESOM для переопределения специфичных для данной реализации классов, ассоциированных с указанными пакетами, таких как подпакеты (sub-packages) *org.w3c.dom* , поставляемые в Sun's Reference Implementation.

* для переопределения автономных технологий Java (Standalone Technologies for Java SE), перечисленных в нижеследующем списке:

*Java API for XML Processing (JAXP), version 1.4*

*Java Architecture for XML Binding (JAXB), version 2.0*

*Java API for XML-Based Web Services (JAX-WS), version 2.0*

*Java Compiler API, version 1.0*

*Pluggable Annotation Processing API, version 1.0*

*Common Annotations for the Java Platform, version 1.0*

*Scripting for the Java Platform, version 1.0*

*SOAP with Attachments API for Java (SAAJ), version 1.3*

Кроме одобренных стандартов Java и автономных технологий для Java перечисленных выше никакие другие пакеты/классы/интерфейсы Java переопределить нельзя.

Вот. Как-то так. ☺

**Определение задачи. Тип каталога, дополнительные сервисы каталога, иерархия каталога и глубина вложенности каталога.**

**Данные. Что мы будем хранить? Где мы все это будем хранить?**

**Хранение данных.**

Чаще всего для постоянного хранения данных используются специальные сервера – СУБД – Сервер Управления Базами Данных. Существует достаточно большое количество таких серверов, как платных, так и бесплатных\*:

* MySQL (бесплатно)
* mSQL (бесплатно)
* PostgreSQL (бесплатно)
* JavaDB (Apache Derby) (бесплатно)
* Oracle (платно)
* Informix (платно)
* DB2 (платно)

\*возможно, указаны не все имеющиеся серверы, а только наиболее популярные среди них. Если что-то упустил – помогите дополнить список. ☺

Можно хранить данные и просто в файлах на диске, но тогда мы потеряем ряд преимуществ, предоставляемых нам СУБД:

* Скорость доступа к данным. СУБД кэширует большое количество данных, хранит индексы для ускорения поиска данных.
* Разделение доступа к данным (многопользовательский режим) – при одновременном доступе к одним и тем же данным большого количества пользователей, возникает конфликт, который в случае файлового хранения данных нам необходимо будет решать вручную (написание кода с использованием специализированных библиотек), а в случае СУБД – многопользовательская работа уже предусмотрена производителем СУБД (весь необходимый код уже написан!). Также в большинстве СУБД предусмотрены средства для балансировки нагрузки на сервер при большом количестве одновременных запросов к данным.
* Права доступа к данным – какие пользователи (или группы) имеют право читать/изменять данные – в случае файлового хранения данных эта ситуация также решается вручную – настройка файловых разрешений и т.п. В случае же СУБД -
* Возможность программного взаимодействия с СУБД с помощью встроенного стандартного языка SQL. Этот язык запросов стандартизован и, при его использовании, программный продукт получается переносимым из одной СУБД в другую (из платной в бесплатную, например) и очень гибким.

Таким образом, преимущества хранения данных в СУБД очевидны (ну наверное очевидны для большинства людей – всегда есть яркие индивидуумы… ☺).

Возможно, есть и другие способы хранения данных, но оставим их описания смешным бородатым дядькам ☺.

Также стоит учитывать следующий момент, который возможен при размещении каталога у провайдера. Если хранить данные в файле на диске, то при их изменении могут возникнуть проблемы с доступом к диску и к этим файлам – многие провайдеры настраивают права доступа таким образом, что изменять файлы внутри каталога сами сценарии каталога не могут (т.е. это может потребовать дополнительной настройки каталога). Такая настройка диктуется обычно соображениями безопасности и удобством обслуживания большого количества java-приложений (у провайдера). Глупые словеса вроде «ну у меня на машине же все работает» не пойдут, потому что настройки машины разработчика обычно сильно отличаются от настроек провайдера, также может сильно различаться набор ПО (кроме версии контейнера Tomcat) – разные ОС, разные файловые системы и т.п.

Еще один немаловажный момент при хранении данных в файле (файлах) – при чтении/записи всегда нужно учитывать кодировку и принудительно ее устанавливать. Этот момент особенно важен при работе на разных ОС – у разработчика одна (обычно семейства MS Windows), у провайдера – другая (обычно семейства UNIX), а, для того чтобы сделать java-код именно кросс-платформенным, необходимо всегда указывать все параметры явно.

Резюмируя все вышесказанное – удобнее, практичнее, надежнее и быстрее хранить данные в БД - те данные, которые часто меняются. Постоянные данные, которые не меняются или меняются очень редко, можно хранить и в файлах. В любом случае, способ хранения данных выбирается разработчиком в зависимости от задачи.

Хотя каждый разработчик волен выбирать способ сам, дальнейшее изложение подразумевает хранение данных каталога в БД. Но многие из нас не ищут легких путей, не правда ли? ☺

**Проектирование Базы данных. Общие рекомендации.**

1. У каждой таблицы БД должно быть ключевое поле – уникальный первичный ключ. Обычно это поле имеет целочисленные положительные значения. В большинстве случаев поле автоинкрементное – его инкрементирование возлагается на СУБД.
2. Для обеспечения целостности данных рекомендуется в каждую таблицу БД включать признак удаления записи – поле deleted (целочисленное или логическое). То есть записи из таблиц физически не удалять, а только помечать как удаленные. Из плюсов – автоматическое обеспечение целостности данных (данные не удаляются). Из минусов – при большом количестве добавлений и удалений (пометки записей как удаленных) записей таблицы начинают распухать, увеличивается время поиска.
3. [Длина записи]
4. [Индексы]
5. [Избыточность данных]
6. [Нормализация таблиц]
7. [Хранимые процедуры]
8. [Триггеры]
9. [Представления]
10. [пример sql-скрипта для создания БД, примеры триггеров, процедур, представлений]

**Для тех, кто в теме. Немного общих рекомендаций по разработке. Для JAVA и не только.**

Предположу, что читатель данного пособия немного знаком с программированием и уже создавал свои программы на каком-либо языке (программирования). А может быть, я ошибаюсь. Но для тех и для других читателей рискну дать немного рекомендаций по разработке исходного кода. Эти рекомендации не претендуют на универсальность и полный охват всех аспектов разработки. Применять рекомендации можно практически к любому языку программирования, в случае же применимости конкретного пункта только к языку JAVA это будет специально указано.

*Стиль программирования*

ОБЩАЯ РЕКОМЕНДАЦИЯ. При разработке следует придерживаться определенного стиля написания кода. Этот стиль может быть определен самим разработчиком для себя (в случае одиночной разработки) или стиль может быть частью командных соглашений (в случае разработки приложения командой разработчиков). В любом из этих случаев при разработке следует всегда придерживаться выбранного стиля – это очень сильно облегчит повторное использование кода (см. ниже), разбор и модификацию старых задач, отладку различных частей приложения. Стиль может включать в себя множество параметров, вот основные из них (читатель может сам дополнить этот список):

* правила наименования имен полей классов, констант, методов, классов
* форматирование исходного кода приложения
* форматирование содержимого журнала (отладочного вывода) приложения
* правила наименования ресурсов
* правила проверок значений переменных, перехвата и обработки исключений

Повторюсь – данный список далеко не полон и каждый разработчик (или команда) может дополнить его самостоятельно.

*Нотация имен (правила создания имен полей, констант, методов, классов)*

ОБЩАЯ РЕКОМЕНДАЦИЯ. Для создания имен полей, констант, методов и классов рекомендуется придерживаться соглашений об именовании, принятых по умолчанию в используемом для разработки языке. Для этого загляните в документацию к используемому языку. Также помощником послужит используемая среда разработки – обычно в любой IDE (Integrated Development Environment) есть различные мастера/помощники по созданию имен различных конструкций языка.

*Документирование (комментирование) исходного кода*

Зачем вообще нужны комменты? Да еще и в коде, где даже картинок нет? Можно провести простой эксперимент – вспомните в деталях какую-либо сложную математическую задачку, которую удалось решить или давным-давно написанную программу (год-два-три назад). Вспомнили? А теперь попробуйте, зная ответ на задачку (результат работы программы) восстановить решение (исходный текст)… Сложно? А если с подсказкой? Если найти тот старый клочок бумаги, на котором и была написана эта бесценная информация. Вот таким клочком бумаги с бесценными данными окажутся для вас комментарии в исходном коде, когда сев изменить/дополнить программу вы поймете, что совершенно забыли – а как я это сделал тогда и зачем здесь эти странные символы… Будем считать, что в необходимости комментариев я вас убедил. Тогда продолжаем. (Если не убедил – смело пропускаем этот раздел ☺)

1. Автодокументирование кода классов и методов (JAVADOC).ПРИМЕНИМО К JAVA. Обязательно документирование кода в формате *javadoc*. Комментарии вносятся в заголовок *каждого класса* – краткое описание назначения и функциональности класса, в заголовок *каждого метода* – описание функциональности метода и его параметров и перед *каждым полем/константой класса*. Комментарии в формате *javadoc* заключаются внутри следующих символов: **/\*\* Комментарий. \*/.** Также в комментариях к классу/методу (но не к полю/константе) должны быть использованы теги *javadoc*:

* **@param <имя параметра> <тип параметра> <описание параметра> -** указывается для каждого входящего параметра метода (если параметры у метода есть)
* **@return <возвращаемый тип> <описание возвращаемого значения> -** указывается только если метод возвращает значение (указывается один раз!)
* **@throws <класс ИС> <описание ИС> –** указывается только если метод возбуждает ИС (указывается для каждой возбуждаемой ИС)

Теги javadoc могут использоваться и в комментариях к классу в целом:

1. **@author <автор>**
2. **@version <версия класса>**

Специальный тег **@deprecated** следует использовать для классов или методов класса, которые не рекомендуются к использованию. После тега следует указывать причину запрещения. Использование данного тега позволяет сохранять совместимость новых версий библиотек со старым исходным кодом – старый класс/метод не удаляется, а просто помечается как «не рекомендуемый к использованию».

1. *Комментирование непосредственно кода методов/классов/объектов.*ОБЩАЯ РЕКОМЕНДАЦИЯ. Вот несколько простых правил для составления комментариев к исходному коду методов/классов:

* рекомендуется выравнивать фигурные скобки по одной колонке и строки по отступам, также рекомендуется форматировать код так, чтобы внутри фигурных скобок код был выровнен по левому краю и с отступом от скобок. Для этого может использоваться табуляция. Рекомендуется объединять несколько строк кода в группы по 3-5 строчек, которые выполняют однородное по своему смыслу действие, такие группы отделяются друг от друга пустыми строками.
* если в коде есть громадные условия и циклы, которые до полного счастья вложены друг в друга, то разобраться в последовательности закрывающих фигурных скобок очень сложно! Для того чтобы не сойти с ума, рекомендуется писать строчный комментарий после «длинной» закрывающей фигурной скобки с оператором, их предваряющим.
* если вы объединяете строки кода в группы по смыслу, то старайтесь писать каждой группе строчку, которая описывает, что эта группа делает. Не имеет смысла комментировать каждую строку: старайтесь все сводить к макроуровню комментариев. Оптимальным считается соотношение строк кода к строкам комментариев — 1/3-1/5.
* никакие комментарии не помогут, если вы используете трехэтажные программные конструкции. Также не стоит злоупотреблять длинными строками (длиннее 80-100 символов) и длинными именами (равно как и короткими названиями). Переменная с именем **the\_sign\_of\_specified\_mathematical\_operation** может довести до колик в животе не одну сотню «читателей» кода. Старайтесь писать более простой код, так чтобы комментарии его дополняли, а не дублировали. Код сам по себе является самым совершенным и лаконичным комментарием, но только в том случае, если он логичен, последователен и прост.
* если по имени переменной нельзя понять, для чего она и что хранит, то на чтение кода приходится тратить лишнее время. Необходимо сначала найти место, где объявляется переменная, а потом определить, что в нее записывается. А то иногда без бутылки пива с кодом разобраться просто невозможно. ☺
* одна переменная никогда не должна выполнять сразу два действия в одном и том же блоке кода! Например, внутри одной процедуры/метода переменная *iFileLength* не должна сначала содержать длину файла, а потом использоваться в качестве счетчика в цикле. Впоследствии очень просто забыть этот нюанс и неправильно использовать значение переменной, что приведет к проблемам безопасности.

*Проверка ввода пользователя.*

ОБЩАЯ РЕКОМЕНДАЦИЯ. Пользователю никогда нельзя доверять. Нет, не так. ПОЛЬЗОВАТЕЛЮ НИКОГДА НЕЛЬЗЯ ДОВЕРЯТЬ! При наличии любых инструкций по заполнению какой-либо формы пользователь, в большинстве случаев, умудряется заполнить ее неправильно. Отсюда следует простой вывод – необходимо ВСЕГДА проверять ввод пользователя на соответствие требованиям программы, поправлять пользователя при ошибочном вводе. Также в текстовых полях ввода рекомендуется обрезать начальные и конечные пробелы, но это решается для каждой задачи индивидуально (может как раз для вашей задачи эти пробелы необходимы).

*Проверка данных в коде. Проверка граничных значений.*

ОБЩАЯ РЕКОМЕНДАЦИЯ. Помимо проверок данных, полученных от пользователя, необходимо проверять и все внутренние данные приложения – проверка на граничные значения (попадает ли параметр в необходимый диапазон) и проверка на пустые (null) значения. Наличие этих проверок может увеличить объем кода приложения и несколько его замедлить (относительно), но спасет от долгой и нудной отладки и от потери времени на обработку возникшей ошибки (из-за неверных данных) – обычно проверка значения выполняется быстрее обработки возникшей ошибки. Также наличие проверок может спасти программу от полного разрушения (во время работы) при поступлении неверных данных для обработки (код ожидающий значение получает пустое (null) значение – без проверки скорее всего возникнет ошибка, ведущая к краху программы). Количество и объем проверок каждый может определить для себя самостоятельно, руководствуясь здравым смыслом.

*Журналирование приложения.*

ОБЩАЯ РЕКОМЕНДАЦИЯ. Что такое журналирование? Это ведение журнала (лога) работы программы, в который программа записывает описание своих текущих действий, описание возникающих ошибок. Для standalone-программы журналирование может и не очень необходимо (хотя наличие журнала очень сильно облегчает поиск и устранение ошибок программы), а вот для серверной системы (каталога), которая обслуживает множество клиентов одновременно, журналирование – насущная необходимость, и пренебрегать этим не стоит. Журнал может сэкономить много часов нудной отладки, увидеть ошибку раньше, чем о ней узнают пользователи! Также в серверной системе журнал можно использовать для выполнения аудита действий пользователей (да и в standalone-приложении эта возможность может сильно пригодиться). Также журналирование следует использовать для наглядного представления состояния объектов/классов системы – в журнал выводятся значения всех полей определенного класса – этот прием также способствует (облегчает) отладку и поиск ошибок в приложении. В отладочный вывод также следует выводить состояние суперклассов (предков) журналируемых классов – если классы приложения являются членами иерархии классов.

Журналирование обычно разделяют на несколько уровней – один уровень при разработке системы, другой – при ее тестировании, третий – для опытной эксплуатации, четвертый – для обычной работы системы (указанное деление носит условный характер и не является шаблоном!). Очень рекомендую использовать разные уровни журналирования для различных стадий жизненного цикла системы/программы. Это позволит получать на каждом этапе жизненного цикла системы только необходимую информацию, что, в конечном итоге, сэкономит нам пару-тройку минут на распитие бутылочки пива…

РЕКОМЕНДАЦИЯ ДЛЯ JAVA.

Для журналирования на JAVA разработано множество модулей. Я настоятельно рекомендую модуль log4j – данный модуль практически стал стандартом «де-факто» в мире JAVA. О его конфигурировании и использовании написано большое количество литературы, модуль прост в освоении и использовании.

Вне зависимости от используемой системы журналирования есть несколько общих простых рекомендаций:

* никогда (кроме разработки и тестирования) не используйте оператор <Exception class>.printStackTrace(). Данный оператор выводит свои данные в стандартный поток вывода ошибок System.error - обычно это консоль системы, что может приводить к сильному засорению логов сервера/системы в стадии нормальной работы, а также имеет еще один существенный минус (см. следующий пункт).
* не используйте операторы System.out.printXX() для журналирования системы – они осуществляют вывод в стандартный поток вывода System.out - обычно это системная консоль. Я обещал рассказать про большой минус использования системной консоли для журналирования? Извольте:
  + **для standalone-приложения** **(J2SE)** системная консоль не показывается вовсе (ну только если вы не хотите видеть на фоне окошка своей программы черное окошко консоли), а даже если и показывается, то т.к. буфер консоли имеет определенный размер, то более ранние сообщения будут затерты более поздними, что не позволит увидеть весь лог. И еще – логи из системной консоли не могут сохранять историю и сохраняться в файле (только копированием из консоли вручную и вставкой в файл также вручную). В общем – неудобно и не функционально. С другой стороны – при разработке приложения с использованием IDE консоль поможет быстро что-то отладить/проверить (большинство IDE имеют возможность пока консоли работающей JVM и запущенной программы). Резюмируем: консоль для standalone-приложения может быть необходима только в процессе разработки и начального тестирования, в нормальном рабочем режиме консоль не нужна!
  + **для web-приложения (J2EE)** системная консоль обычно это лог сервера. Админ не будет рад увидев в своем серверном логе ваши отладочные/информационные сообщения (о чем скорее всего сообщит с топаньем ног и применением местных идиоматических выражений). Также, т.к. в лог сервера пишет сам сервер, то найти ваши сообщения среди серверных может оказаться проблематично. Резюмируем: использоавние консоли для журналирования J2EE приложения нецелесообразно.

*Использование ресурсов.*

ОБЩАЯ РЕКОМЕНДАЦИЯ. Если ваша программа использует какие-либо ресурсы системы или сети (файлы, соединения с базой данных, порты удаленного сервера и т.п.), которые кроме вас используются еще кем-то (общие или разделяемые ресурсы), то после использования таких ресурсов их ВСЕГДА необходимо освобождать. Даже в том случае, если программа аварийно завершается. В противном случае возможно появление «демонов» - ресурсов, которые никем не используются, но доступ к ним для всех закрыт.

РЕКОМЕНДАЦИЯ ДЛЯ JAVA. При работе с ресурсами следует использовать конструкцию *try…catch…finally* и в блоке *finally* освобождать используемые ресурсы. Конечно, можно попробовать положиться на то, что при завершении работы JVM будут автоматически освобождены все используемые ресурсы, но при неудачном освобождении ресурсов никто не сообщит об этом пользователю/системе и очень трудно будет это отловить (отловить ошибку). Также и в случае с технологией J2EE – захваченные из общего (для всех приложений на сервере) серверного набора соединения с СУБД при крахе приложения могут так и остаться заблокированными и остальные приложения на сервере не смогут их получить, что приведет к неработоспособности остальных приложений.

*Использование библиотек сторонних разработчиков (не стоит все писать самостоятельно).*

ОБЩАЯ РЕКОМЕНДАЦИЯ. Нет ничего нового под солнцем, все уже было когда-то. Эти слова в полной мере относятся и к программированию. Большинство алгоритмов уже имеют реализации практически на любом языке, большинство задач кем-то когда-то уже было реализовано. Такое количество наработанного материала позволяет нам всесторонне его использовать в своих разработках, тем более что все это многообразие предоставляется в свободное использование различными сообществами или производителями программных средств и технологий (только не забывайте о лицензировании – всегда учитывайте условия использования выбранных вами библиотек/алгоритмов сторонних разработчиков).

РЕКОМЕНДАЦИЯ ДЛЯ JAVA. Для JAVA предлагается очень большой набор бесплатных библиотек, который стоит немного изучить, для того, чтобы не написать повторно уже написанное. Ресурсы, где можно найти эти библиотеки, указаны в конце пособия.

*Повторное использование кода.*

ОБЩАЯ РЕКОМЕНДАЦИЯ. При написании приложений можно заметить, что некоторые участки кода в них совпадают – например, реализация каких-либо алгоритмов, доступ к ресурсам и т.п. Такие участки кода целесообразно не плодить методом «COPY»-«PASTE», а выделять в отдельные общие для нескольких приложений библиотеки/модули. Количество повторений участка кода для выделения его в библиотеку устанавливается разработчиком самостоятельно, но рекомендуется это делать после повтора кода более чем в двух местах.

*Использование автоматизированных средств сборки и публикации конечного проекта.*

ОБЩАЯ РЕКОМЕНДАЦИЯ. Если вы используете графическую среду для разработки, то возможно более никакие средства автоматизации сборки/публикации не нужны – среда предоставляет все возможности для этого. Но возможностей среды может не хватить для выполнения всех необходимых при сборке действий, тогда используются сторонние средства автоматизации – средства выполнения сценариев, командные файлы (batch-файлы) и т.п.

РЕКОМЕНДАЦИЯ ДЛЯ JAVA. Основным средством автоматизации сборки/публикации проекта служит ANT – средство выполнения сценариев. Подробнее о нем читайте ниже, также в списке рекомендуемых ресурсов можно найти ссылку на домашний сайт ANT’a.

*Резервное копирование проекта.*

ОБЩАЯ РЕКОМЕНДАЦИЯ. Резервное копирование данных напоминает страхование машины/квартиры – если страховка не понадобилась, то потом многие жалеют о зря потраченных средствах/времени, а вот если понадобилась… И наоборот, если страховка понадобилась, но ее нет, то остается только кусать локти. Чтобы нам не пришлось потом кусать локти рекомендуется периодически делать резервную копию проекта, периодичность и объем резервирования каждый может определить для себя сам.

**Попьем кофейку? JAVA. Описание языка. Версии и компоненты.**

**J2EE. Корпоративная Ява – самый крепкий сорт. ☺**

[технологии JSP и Servlet - вкратце]

[специальные сервлеты – фильтры и слушатели]

**Инструменты создателя. А чем собственно пользоваться?**

[описание и скриншоты сред разработки IDEA и NetBeans]

[сравнение платных и бесплатных сред разработки]

**ГЛАВА 5. СЕРВЕР TOMCAT.**

**Ловим нашего Кота Тома (TOMCAT). Часть I. Первое знакомство.**

[описание Томката, зачем, для чего и т.п.]

**Ловим нашего Кота Тома (TOMCAT). Часть II. Как поймать и заставить работать. Начало.**

1. Подразумеваем, что на компе у нас установлена одна из ОС семейства виндовз – Windows XP (SP1/2/3), Windows 2000 Server\Workstation, Windows 2003 Server. На Висте – не факт, что заработает... по крайней мере – не проверял.
2. Скачиваем дистрибутив Томката с сайта tomcat.apache.org. Рекомендуется скачивать зип-архив, а не инсталлятор – с ним проще работать (это мое мнение).
3. Кладем (распаковываем) архив с Томкатом в папочку на диске. Пусть у нас это будет d:\java\tomcat6. Далее в этом документе эту папочку будем называть CATALINA\_HOME.
4. Необходимо, чтобы на компе была установлена Жава – не жава-машина (JRE), а полноценный пакет JDK (он необходим, чтобы компилировать JSP в Servlet). Томкат 6 работает с Жавой 5 (1.5) и выше. Путь к установленной Жаве (к ее домашнему каталогу) должен быть прописан в переменной окружения JAVA\_HOME. Например, если JDK у нас установлен в каталог d:\java\jdk1.6.0, то необходимо иметь установленную переменную окружения JAVA\_HOME=d:\java\jdk1.6.0 Также необходимо добавить в переменную окружения PATH путь к каталогу bin установленного JDK (в нашем случае это будет d:\java\jdk1.6.0\bin).
5. Проверяем, что все работает – из командной строки, находясь в каталоге CATALINA\_HOME\bin запускаем [catalina.bat run] <- запуск сервера с консолью в том же окне, или [catalina.bat start] <- запуск сервера с консолью в новом окне. Читаем лог сервера – ошибок в нем не должно быть (по идее… ☺). Запускаем браузер и вводим <http://localhost:8080> – мы должны увидеть стартовую страничку Томката. Завершение работы сервера – комбинация клавиш Ctrl+C.
6. Также рекомендуется установить т.н. Native Connector – библиотека, которая связывает сервер Томкат с платформой, на которой он запущен – это несколько ускоряет работу сервера. Где скачать – см. сайт tomcat.apache.org. Для виндов коннектор – это обычная библиотека DLL. Коннектор кидаем в папку CATALINA\_HOME\bin – при запуске сервака будет запись о загрузке коннектора – обязательно проверить!

**ВАЖНО!** Для Виндовс ХаРе (а может и для других виндов семейства NT (Win2000, Win2003Server) тоже) при включенном файерволе при останове Томката возникает ошибка – она происходит в Native коннекторе и видимо связана с работой файервола. Ошибка видна в виде файла **hs\_err\_pidXXXX.log** , который появляется в каталоге CATALINA\_HOME\bin (версия кошака 6.0.18)

1. Логи сервера смотреть в каталоге CATALINA\_HOME\logs
2. Основная конфигурация сервера находится в файлах CATALINA\_HOME\conf\server.xml (конфигурация сервера) и CATALINA\_HOME\conf\web.xml (конфигурация по умолчанию для всех приложений на сервере). Перед использованием Томката его необходимо сконфигурировать – см. документацию к самому Томкату!
3. В зависимости от выбранной конфигурации, необходимо добавить в каталог библиотек Кошака – это каталог [CATALINA\_HOME]\lib – необходимые нам библиотеки (например, JDBC-драйвера используемой СУБД).

**Ловим нашего Кота Тома (TOMCAT). Часть III. Конфигурирование.**

Авторизация на сервере Томкат. Использование БД или текстового файла пользователей.

Для создания механизма авторизации сервера Томкат на основании БД пользователей делаем следующее (пример – СУБД MySQL) – описание действий для выполнения на сервере Томкат.

* Создаем БД пользователей, например такую (скрипт MySQL):

[authdb.sql]

**create database if not exists authdb;**

**use authdb;**

**create table users**

**( id int auto\_increment primary key,**

**userName varchar(50) not null unique,**

**userPass varchar(50) not null unique,**

**userDesc varchar(255));**

**create table roles**

**( id int auto\_increment primary key,**

**roleName varchar(50) not null unique,**

**roleDesc varchar(255));**

**create table users\_roles**

**( id int auto\_increment primary key,**

**userName varchar(50) not null,**

**roleName varchar(50) not null,**

**unique (userName, roleName));**

* В файле server.xml внутри тега <Engine></Engine> вставить описание РЕАЛМА (realm):

**<!-- MYSQL SECURITY REALM -->**

**<Realm className="org.apache.catalina.realm.JDBCRealm" debug="0"**

**driverName="com.mysql.jdbc.Driver"**

**connectionURL=**

**"jdbc:mysql://localhost:3306/authdb?user=root&amp;password=mysql"**

**userTable="users" userNameCol="userName" userCredCol="userPass"**

**userRoleTable="users\_roles" roleNameCol="roleName" />**

Реалм – это «комната, подвал, ограниченное пространство» - область с ограничениями доступа. Описание параметров реалма смотри в доке к Томкату.

* В каталог CATALINA\_HOME\lib скопировать JDBC-драйвер для соединения с СУБД

(в нашем примере – mysql.jar)

* Стартовать сервак и проверить, что он стартует без ошибок! ☺

Авторизация на сервере Томкат на основании файла пользователей довольно проста.Этот способ является способом авторизации по умолчанию – см. записи реалма в файле server.xml, поставляемом с сервером по умолчанию.

Использование авторизации для защиты ресурсов приложения.

Если необходимо защитить какие-либо ресурсы (URL) приложения от неавторизованного доступа, необходимо сконфигурировать приложение. Для этого нужно в файл web.xml этого приложения (а не всего сервера), сделать записи:

*[описание набора ресурсов данного приложения, для которого вводится ограничение доступа, роль, используемая для доступа к данному набору ресурсов, транспорт обеспечения доставки логина/пароля. Подробнее – доки Кошака.]*

**<security-constraint>**

**<display-name>admin\_constraint</display-name>**

**<web-resource-collection>**

**<web-resource-name>admin\_resources</web-resource-name>**

**<url-pattern>/admin/\*</url-pattern>**

**<http-method>GET</http-method>**

**<http-method>POST</http-method>**

**</web-resource-collection>**

**<auth-constraint>**

**<role-name>admin</role-name>**

**</auth-constraint>**

**<user-data-constraint>**

**<transport-guarantee>NONE</transport-guarantee>**

**</user-data-constraint>**

**</security-constraint>**

*[конфигурация для логина (авторизации) данного приложения – метод авторизации и наименование РЕАЛМА, в который будет стучать пользователь. Подробнее – доки Кошака.]*

**<login-config>**

**<auth-method>BASIC</auth-method>**

**<realm-name>[RESTRICTED AREA]</realm-name>**

**</login-config>**

*[перечисление ролей, используемых данным приложением. Только роли, перечисленные здеся можно включать в описание ресурсов – теги* ***security-constraint****]*

**<security-role>**

**<role-name>admin</role-name>**

**<role-name>manager</role-name>**

**</security-role>**

При внесении этих записей очень важно соблюдать порядок следования тегов! Он должен быть именно таким. Также тег **security-constraint** должен следовать ЗА тегом(тегами) **resource-ref** или **servlet-mapping.**

Логгирование (использование в пользовательском приложении логгера log4j).

Для того, чтобы наше приложение могло использовать расширенное логгирование с помощью модуля log4j необходимо выполнить простые шаги:

* Положить файло log4j.jar (имя может быть и вида log4j-1.2.9.jar) в каталог

**WEB-INF\lib** нашего приложения.

* В каталог **WEB-INF\classes** (где лежат скомпилированные сырцы нашего приложения) положить файл **log4j.properties** следующего вида (данный файл является примером, полную инфу о настройках логгера логФоДжей смотри в доке):

**log4j.logger.avstore=debug, fAppender, cAppender**

**log4j.logger.org=info, fAppender, cAppender**

**log4j.logger.jdb=info, fAppender, cAppender**

**# Console appender for this logger - ONLY FOR DEVELOPMENT**

**log4j.appender.cAppender=org.apache.log4j.ConsoleAppender**

**log4j.appender.cAppender.layout=org.apache.log4j.PatternLayout**

**log4j.appender.cAppender.layout.ConversionPattern=%d{dd/MM/yyyy HH:mm:ss} {%10M} %p %c - %m%n**

**# File appender for this logger - FOR DEPLOYED APPLICATION**

**log4j.appender.fAppender=org.apache.log4j.RollingFileAppender**

**log4j.appender.fAppender.File=d:/JAVASERVER/logs/avstore/avstore\_new.log**

**log4j.appender.fAppender.MaxFileSize=10MB**

**log4j.appender.fAppender.MaxBackupIndex=10**

**log4j.appender.fAppender.layout=org.apache.log4j.PatternLayout**

**log4j.appender.fAppender.layout.ConversionPattern=%p %t %c - %m%n**

* Непосредственно в тексте программы использовать конструкцию вида

**private Logger logger = Logger.getLogger(avstore);** для получения доступа к журналированию (в нашем примере имя логгера **avstore** – имя может быть произвольным).

Страница, открываемая сервером по умолчанию (по адресу <http://localhost:8080>). Ну и заколебался же я с этим!!!!

По умолчанию сервак (после вышеприведенной процедуры инсталляции) по данному адресу откроет (в браузере) страничку, которая будет находиться здесь: CATALINA\_HOME\webApps\ROOT (обычно это страничка типа «Велкам ту Томкат» со ссылками на манагер Томката и страничку статуса). Для того, чтобы сервак открывал нашу страничку по адресу по умолчанию (**Ошибка! Недопустимый объект гиперссылки.>**) необходимо:

* Удалить (или архивнуть в рар и удалить) папку CATALINA\_HOME\webApps\ROOT
* Скопировать наше приложение в виде файла **ROOT.WAR** в папку CATALINA\_HOME\webApps – сервак сам развернет приложение в папку ROOT (если в теге HOST файла server.xml есть следующий атрибут – [unpackWARs="true"]) и будет открывать его по умолчанию
* Если наше приложение существует в виде раскрытого каталога – скопировать содержимое этого каталога в папку CATALINA\_HOME\webApps\ROOT (**ТОЛЬКО ОДИН ИЗ ВАРИАНТОВ – ЛИБО ROOT.WAR, ЛИБО РАСКРЫТЫЙ КАТАЛОГ!!!**)
* Проверить, что в корне папки ROOT (или нашего каталога с приложением) существует один из файлов, перечисленных в списке **welcome-file-list** файла web.xml, который находится здеся: CATALINA\_HOME\conf\web.xml (файл должен физически существовать – различные маппинги не подойдут!)
* Все что относится к глобальному файлу web.xml (который находится тут CATALINA\_HOME\conf\web.xml), применимо и к локальному файлу web.xml (он находится тут: КАТАЛОГ\_ПРИЛОЖЕНИЯ\WEB-INF) нашего приложения – в нем тоже можно прописывать список **welcome-file-list**.
* **ВАЖНО! Если мы используем СТРАТС 1 (STRUTS 1).** При использовании фреймворка СТРАТС (1) мы можем использовать его маппинги. Мап по умолчанию в стратсе - **\*.do** Т.е. маппинг на главную страничку будет примерно таким **/index.do** а на самом деле в корне на будет файла index.do – при этом, если прописать его (файл index.do) в списке **welcome-file-list** файла web.xml (CATALINA\_HOME\conf\web.xml), то он найден не будет (маппинг-то виртуальный – от так! Гы.) Решение? ОНО ЕСТЬ!!!! Решений даже несколько.

**Решение №1.** В корне создаем реальный файл index.jsp (прописываем его в велкам-файл-лист) следующего содержания:

<%@ page contentType="text/html; charset=windows-1251" %>

<jsp:include page="/index.do"/>

**Решение №2.** Можно еще вот так:

***Пишем struts-config такого типа***<global-forwards>  
 <forward name ="welcome" path="XXXXX.do">  
<global-forwrds>  
***Создаем* *welcome.jsp вот так***<logic forward name='welcome'>  
***В глобальном web.xml пишем так***<welcome-file-list>  
<welcome-file>welcom.jsp<welcom-file>  
<welcome-file-list>

Инсталляция Кошака как сервиса в Виндах

Полностью материал находится тута:

<http://tomcat.apache.org/tomcat-6.0-doc/windows-service-howto.html>

Возможна ситуация, когда для тех или иных целей будет необходимо иметь постоянно запущенный сервер Tomcat. Запускать его вручную каждый раз – не самое удачное решение. А что делать? Эврика! Можно инсталлировать Tomcat как сервис в ОС Виндовз! О как! А теперь, мой маленький друг, предлагаю этим и заняться.

Замечание. Не трудно догадаться – сервисов Томката может быть запущено на одной машине много. Необходимо внимательно следить за их конфигурациями, чтобы запущенные сервисы не дрались из-за портов компа. Вроде это очевидно, но все же – будь внимателен, дружок!

Рассмотрим наш сервер Tomcat пристально – заглянем в файловую его структуру, в каталог bin. В этом каталоге находится множество файлов, пока нас интересуют только два:

* **tomcat6.exe** – сервисное приложение для запуска сервера Tomcat как сервиса Виндовз (для семейства NT – 2000/2003/XP/Vista/7). А сейчас опишем директивы (опции) командной строки для данного приложения. Каждая директива (опция) должна иметь следующую форму **//XX//ServiceName**. Непосредственно директивы командной строки:
  + - //TS//ServiceName - запуск сервиса как консольного приложения. Это директива по умолчанию, используется если нет других директив (опций). **ServiceName** – наименование запускаемого файла без суффикса **exe** (означает Tomcat6).
    - //RS//ServiceName - запуск сервиса. Вызывается только из ServiceManager.
    - //SS//ServiceName - останов сервиса.
    - //US//ServiceName - обновление параметров сервиса.
    - //IS//ServiceName - инсталляция сервиса (службы).
    - //DS//ServiceName - удаление сервиса (службы). Если служба запущена – она будет остановлена и удалена.
* **tomcat6w.exe** – графическое приложение для мониторинга и конфигурирования сервиса (-ов) Tomcat. Директивы командной строки для данного приложения:
  + - //ES//ServiceName - редактировать конфигурацию сервиса. Операция по умолчанию. Вызывается, если опции не указаны, а запускаемый файл сервиса переименован в **servicenameW.exe**
    - //MS//ServiceName - выполнять мониторинг сервиса. В системный трей помещается иконка данного приложения.

Параметры командной строки (используются для приложения Tomcat6.exe)

У данного сервиса есть большой набор параметров командной строки, сейчас попробуем его описать. Описаны только основные параметры – полное описание см. по ссылке на англоязычную доку. Также с помощью данных параметров возможно переопределить/дополнить переменные окружения процесса – это также см. в полном описании.

Каждый параметр начинается с символа -- (минус-минус). Также в качестве префикса параметра может быть указан символ ++ (плюс-плюс).

Если параметр командной строки начинается символом ++, то значение указанного параметра добавляется к существующей опции (существующему значению опции). Это свойство можно использовать, например, при обновлении параметров установленного сервиса для того, чтобы не затереть уже имеющееся значение параметра.

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Значение по умолчанию** | **Описание** |
| --Description |  | Описание сервиса (макс. 1024 символа) |
| --DisplayName | ServiceName | Наименование сервиса |
| --Install | procrun.exe  //RS//ServiceName | Инсталлируемый образ |
| --Startup | manual | Режим запуска сервиса. Может быть **auto** или **manual** |
| --DependsOn |  | Список сервисов, от которых зависит данный сервис. Наименование сервисов разделяются символом # или символом ; |
| --Environment |  | Список переменных окружения, передаваемых сервису. Формат key=value. Пары разделяются символом # или символом ; |
| --User |  | Аккаунт пользователя, используемый для запуска исполняемого файла сервиса. Используется только для значений java или exe параметра StartMode (см. ниже). Параметр позволяет запускать приложение как сервис под аккаунтом, который даже не имеет привелегии LogonAsService. О как. Везде есть лазейки… |
| --Password |  | Пароль аккаунта, указанного параметром --User |
| --JavaHome | JAVA\_HOME | Установка значения JAVA\_HOME отличного от указанного переменной окружения JAVA\_HOME |
| --Jvm | auto | Использование JVM автоматически (где найдет) – auto, или указание полного пути к файлу jvm.dll. В данном параметре можно использовать расширение переменных окружения (см. Полный оригинальный текст статьи). |
| --JvmOptions | -Xrs | Список опций JVM в форме -D или -X, которые передаются жаба-машине. Опции разделяются символом # или символом ; |
| --Classpath |  | Указание значения classpath |
| --JvmMs |  | Начальный размер пула памяти JVM (в Мб) |
| --JvmMx |  | Максимальный размер пула памяти JVM (в Мб) |
| --JvmSs |  | Размер стека потоков (в Кб) |
| --StartImage |  | Исполняемый файл (executable) для запуска Кошака |
| --StartPath |  | Путь к исполняемому файлу для запуска Кошака |
| --StartClass |  | Класс, используемый для запуска Кошака |
| --StartParams |  | Список параметров, передаваемых StartImage и StartClass. Параметры разделяются символом # или символом ; |
| --StartMethod | Main executable | Имя запускаемого метода класса. Указывается только если имя метода для запуска отличается от main |
| --StartMode |  | Может принимать значения jvm java или exe |
| --StopImage |  | Исполняемый файл (executable), запускаемый при останове сервиса |
| --StopPath |  | Путь к исполняемому файлу, запускаемому при останове сервиса (StopImage) |
| --StopClass |  | Класс, используемый для останова сервиса |
| --StopParams |  | Список параметров, передаваемых StopImage и StopClass. Параметры разделяются символом # или символом ; |
| --StopMethod | Main executable | Имя метода для останова. Указывается только если имя метода отличается от main |
| --StopMode |  | Может принимать значения jvm java или exe |
| --StopTimeout | нет таймаута | Время таймаута (в секундах) для ожидания останова сервиса. |
| --LogPath | рабочий путь | Путь для логгирования |
| --LogPrefix | jakarta\_service | Имя лог-файла сервиса |
| --LogLevel | INFO | Уровень логгирования. Может иметь значения: error, info, warn, debug. |
| --StdOutput |  | Перенаправление стандартного файла stdout |
| --StdError |  | Перенаправление стандартного файла stderr |

Непосредственно инсталляция сервиса

Для инсталляции сервиса Томката в виндах есть более предпочтительный путь – не ломать моск этими параметрами, а воспользоваться готовым скриптом **service.bat** , расположенным также в каталоге **bin** Кошака. Для запуска скрипта как минимум необходимо обладать правами администратора. Однако использование данного файла не предполагает тонких настроек сервиса – для этого все же придется поломать моск. Посмотреть как устроена инсталляция – также можно в файле service.bat – он написан достаточно понятно и снабжен комментариями. Не рекомендую модифицировать стандартный файл – лучше скопировать его и уже тогда проволить эксперименты.

**ВАЖНО!** Для ОС Windows VISTA/7 или других с функцией UAC (User Account Control, Контроль Учетных Записей) перед запуском скрипта необходимо отключить эту функцию или запускать скрипт командой «Запуск от имени администратора» («Run as administartor») из контекстного меню (не знаешь, что такое контекстное меню – иди учи матчасть, Дятел!) ВНИМАНИЕ! Если функция UAC включена, просто войти (залогиниться) как Administrator (Администратор) не достаточно для запуска скрипта (точнее, для его успешной отработки)!

Пример использования скрипта:

*C:\> service.bat install*

После выполнения данной команды будет установлена служба с именем “Tomcat6”.

Важно помнить, что данная команда успешно выполнится, только если путь к каталогу bin экземпляра сервера Томкат буден прописан в переменной окружения PATH процесса (процесса, из которого запускается батник). Поэтому необходимо или прописать этот путь в переменную окружения (этот подход имеет существенный минус – см. ниже), или запускать батник используя полный путь:

*C:\> d:\java\apache-tomcat\bin\service.bat install*

**О минусе прописывания пути в переменную PATH.** Этот минус проявляется только если нам необходимы несколько экземпляров Томката на машине – при выполнении команды запуска батника просто по имени буден запущен тот файл, путь к которому прописан первым (т.е. первый найденный файл). Т.е. в случае нескольких экземпляров Томката на машине лучше всего запускать батник по полному имени – будет уверенность, что запущен батник из нужного экземпляра Томката.

Еще об инсталляции нескольких сервисов Томката на машине. Для удобства инсталляции можно создать batch-файл примерно такого содержания (сам использую такой же файлец):

*@echo off*

*setlocal*

*set JAVA=d:\java\*

*set TOMCAT\_1=apache-tomcat-6.0.18-first*

*set TOMCAT\_1\_SERVICE=FIRST*

*set TOMCAT\_2=apache-tomcat-6.0.18-second*

*set TOMCAT\_2\_SERVICE=SECOND*

*set TOMCAT\_3=apache-tomcat-6.0.18-third*

*set TOMCAT\_3\_SERVICE=THIRD*

*echo -- Installing Tomcat service %TOMCAT\_1\_SERVICE%*

*set CATALINA\_HOME=%JAVA%%TOMCAT\_1%*

*call %JAVA%%TOMCAT\_1%\bin\service.bat install %TOMCAT\_1\_SERVICE%*

*%JAVA%%TOMCAT\_1%\bin\tomcat6.exe //US//%TOMCAT\_1\_SERVICE% --Startup auto*

*echo -----------------------------------------------*

*echo -- Installing Tomcat service %TOMCAT\_2\_SERVICE%*

*set CATALINA\_HOME=%JAVA%%TOMCAT\_2%*

*call %JAVA%%TOMCAT\_2%\bin\service.bat install %TOMCAT\_2\_SERVICE%*

*%JAVA%%TOMCAT\_2%\bin\tomcat6.exe //US//%TOMCAT\_2\_SERVICE% --Startup auto*

*echo -----------------------------------------------*

*echo -- Installing Tomcat service %TOMCAT\_3\_SERVICE%*

*set CATALINA\_HOME=%JAVA%%TOMCAT\_3%*

*call %JAVA%%TOMCAT\_3%\bin\service.bat install %TOMCAT\_3\_SERVICE%*

*%JAVA%%TOMCAT\_3%\bin\tomcat6.exe //US//%TOMCAT\_3\_SERVICE% --Startup auto*

Для использования данного файла – отредактируй его под себя.

Командный файл установки службы (**service.bat**) имеет несколько параметров:

* **install [service name]** – установка службы с указанным именем [service name]. Если имя службы указано не будет, то используется имя службы по умолчанию (для Кошака версии 6.0.18 это имя “Tomcat6”)
* **remove [service name]** – удаление службы с указанным именем [service name]. Если имя службы указано не будет, то используется имя службы по умолчанию (для Кошака версии 6.0.18 это имя “Tomcat6”)

Если для инсталляции сервиса используется приложение **tomcat6.exe**, то необходимо использовать директиву командной строки //IS//ServiceName. Пример использования данного приложения (в несколько строк указана одна команда!):

*C:\> tomcat6 //IS//Tomcat6 --DisplayName="Apache Tomcat 6" \*

*C:\> --Install="C:\Program Files\Tomcat\bin\tomcat6.exe" --Jvm=auto \*

*C:\> --StartMode=jvm --StopMode=jvm \*

*C:\> --StartClass=org.apache.catalina.startup.Bootstrap --StartParams=start \*

*C:\> --StopClass=org.apache.catalina.startup.Bootstrap --StopParams=stop*

Другие операции с сервисом

Сервис Кошака можно изменить (обновить) – изменить его параметры. Для этого используется директива //US//. Пример использования:

*C:\> tomcat6 //US//Tomcat6 --Description="Apache Tomcat - http://tomcat.apache.org/ " \*

*C:\> --Startup=auto --Сlasspath=%JAVA\_HOME%\lib\tools.jar;%CATALINA\_HOME%\bin\bootstrap.jar*

Еще одна важная операция – удаление сервиса. Для этого используется директива //DS//. Также приводим пример использования:

*C:\> tomcat6 //DS//Tomcat6*

Также нам возможно понадобится отслеживать работу нашего установленного сервиса в реальном времени (ну неохота нам логии читать, хочется консольку глянуть!). Это возможно реализовать! Для запуска сервиса в консольном режиме необходимо использовать директиву //TS//. Завершить запущенный таким образом сервис возможно нажатием одной из комбинаций клавиш: Ctrl+C или Ctrl+BREAK. Консольный режим запуска применяется по умолчанию – для этого просто необходимо запустить сервисное приложение (исполняемый файл).

Запуск сервиса, называющегося 'Tomcat6' в консольном режиме:

*C:\> tomcat6 //TS//Tomcat6 [additional arguments]*

Или можно просто выполнить такую команду:

*C:\> tomcat6*

По опыту могу сказать – удобнее всего использовать именно **service.bat** для установки/удаления сервиса, а приложение **tomcat6.exe**  - использовать только для манипуляций с уже установленным сервисом (например, обновления параметров установленного сервиса). Однако каждый сам находит себе развлечение, поэтому – выбирай, мой юный друг!

**Еще немного о сервисах.**

* если для установки сервиса использовался **service.bat**, то и для удаления сервиса следует использовать его же (с ранее указанным именем сервиса). Если использовалась установка вручную или с помощью своих батников, то удалять сервис необходимо с помощью команды **sc delete [service\_name]** (sc – services control - смотри справку в виндах).
* некоторые параметры сервиса Tomcat не могут быть изменены после установки сервиса, они могут быть указаны только при инсталляции сервиса. Об этом в доке не нашел ни слова (вот ведь…). Один из таких параметров – зависимости службы Томката – параметр

*--DependsOn [service1;….;serviceN]*

(от каких служб будет зависеть устанавливаемая/установленная служба), этот параметр работает только при указании его при инсталляции службы. Подозреваю, есть еще такие параметры.

Опасные связи или чудеса генетики: скрещивание Кошака. Установка и настройка редиректора для связки IIS+Tomcat.

Как всегда, самая полная и свежая версия документации находится на оф. сайте Томката: <http://tomcat.apache.org/connectors-doc/reference/iis.html>. Эта статья состоит из перевода оф. сайтовой доки и из опыта работы автора, с его же дополнениями. Итак.

Для данного экзерсиса мы использовали следующее ПО:

* Windows 2003 server standard
* IIS 6.0
* Apache Tomcat connector 1.2.30 (версию поновее мона взять тут: <http://tomcat.apache.org/download-connectors.cgi>)
* Apache Tomcat 6.0.18

Из предыдущего раздела мы узнали как устанавливать службу Tomcat на машине и автоматически ее стартовать. Что же еще нам может понадобиться? Сам по себе Кошак является контейнером сервлетов, который умеет отвечать на HTTP-запросы, еще он может обрабатывать статический контент (отдавать его по запросам пользователей) – т.е. может работать как полноценный веб-сервер. Но возможности полноценных веб-серверов, таких как Apache или IIS гораздо шире, поэтому использовать стоящий отдельно Томкат нецелесообразно. Также соображения безопасности нашептывают нам, что не стоит оставлять Кошака одного. Поэтому обычно Кошак находится «за спиной» полноценного веб-сервера (как IIS или Apache).

Мы сейчас рассмотрим вариант скрещивания Томката и веб-сервера IIS (версии 6.0). Для этого используется специальный модуль – редиректор. Интересно? Тогда вперед! (если неинтересно – смело пропускаем этот раздел)

**ВАЖНО!**

* Еще раз повторюсь – при наличии на машине нескольких Кошаков необходимо следить, чтобы они висели на РАЗНЫХ портах машины, иначе работать ничего не будет. Как настроить Кошак на определенный порт см. на сайте tomcat.apache.org (или может появится такой раздел в данной доке – 04.02.2011)
* Еще один немаловажный момент. Если конфигурируется связка IIS + Tomcat с использованием редиректора, то для обеспечения безопасности сервера Томкат и его содержимого рекомендуется в конфигурации сервера Томкат (файло server.xml) отключать HTTP-connector (он позволяет напрямую соединяться с сервером Томкат по протоколу HTTP) и оставлять только AJP-connector (этот коннектор позволяет обмениваться данными с Кошаком только по протоколу AJP). Подробнее о коннекторах будет рассказано выше… но потом… (ремарка от 14.02.2011)

**Редиректор Томката для работы нуждается в нескольких вещах:**

* конечно же сам сервер **Tomcat**. Установленный и настроенный.
* **isapi\_redirect\_1.2.30.dll** – плагин для сервера IIS. Файл скачивается с сайта tomcat.apache.org – поисчи там, должон быть. Имя библиотеки может быть любым. Удобно совмещать его (имя библиотеки) с номером версии самой библиотеки.
* опционально – при конфигурировании модуля редиректора с помощью конфигурационного файла свойств (properties-файл) – сам файл свойств **isapi\_redirect\_1.2.30.properties**.Имя файла свойств (без расширения) должно совпадать с именем самого модуля редиректора.
* конфигурационный файл **workers.properties** – файл, описывающий хост (хосты) и порт (порты), используемые воркерами (workers). Расположение файла задается конфигурацией модуля редиректора. Что такое воркер (worker) – будет рассказано чуть ниже.
* конфигурационный файл **uriworkermap.properties** – файл описывающий (осуществляющий маппинг) отношение URL (URL-Path patterns) к воркерам (какой воркер обрабатывает конкретный путь). Расположение файла задается конфигурацией модуля редиректора. Что такое воркер (worker) – будет рассказано чуть ниже.
* Каталог для хранения лог-файлов. Этот каталог должен существовать перед установкой и запуском редиректора, т.к. если каталога не будет (редиректор сам его не создает) редиректор не сможет стартовать из-за невозможности ведения лога. Лог-файл создавать необязательно – его редиректор создает в каталоге сам.

**Непосредственно установка и первичное конфигурирование редиректора**

Ниже описаны шаги конфигурирования связки двух серверов IIS + Apache Tomcat. Связь осуществляется с помощью специального программного модуля – редиректора. Конфигурирование показано на примере ОС Виндовс. Аналогично конфигурируется модуль связи серверов Apache + Apache Tomcat. Точнее говоря – точно так же, только в случае такой связки (Apache + Apache Tomcat) модуль используется другой. Но о таких связках – читай источник (онлайновую доку по Томкату). При конфигурировании редиректора используются несколько параметров – их описание находится далее по тексту, поэтому для начала прочитайте раздел полностью, а потом выполните все указанные действия по шагам. Также используются конфигурационные файлы – workers.properties и uriworkermap.properties, их примеры можно найти в пакете с исходниками модуля редиректора.

Предполагается, что используется Томкат в том виде, в котором он был получен с сайта загрузки, значения параметров конфигурации в файлах workers.properties и uriworkermap.properties соответствуют начальным установкам Томката и эти файлы также в неизменном виде взяты из дистрибутива (или исходника) модуля редиректора. Все описанное ПО легко взять с сайта <http://tomcat.apache.org>

1. Для начала надо проверить, что наш сервер Tomcat работоспособен сам по себе. Самый лучший вариант – установка сервера Tomcat в качестве сервиса/службы (для ОС Win – см. как это сделать выше). Будем предполагать именно этот вариант (при использовании других вариантов необходимо просто запустить имеющийся сервер Tomcat). Для проверки работоспособности сервера запускаем web-браузер и вводим адрес <http://localhost:8080> – мы должны увидеть страницу приглашения по умолчанию (предполагается, что изначальная конфигурация сервера не изменена, если же это не так могут помочь другие адреса: <http://localhost:8080/examples> (открытый адрес), <http://localhost:8080/manager/status>).

Страница по умолчанию для только установленного сервера Tomcat 6.0.18 выглядит примерно так (ОС Windows 2003 server standard):



Если проверка работоспособности прошла удачно – идем дальше, если же нет – то необходимо заставить сервак работать, без этого дальнейшее повествование и, соответственно чтение, бессмысленно.

1. Теперь необходимо определиться со способом конфигурирования редиректора. Их есть несколько:
2. вручную, через реестр ОС Windows (нужные ключи и их значения создаются руками). По-моему самый непростой способ. Да еще и с реестром париться…
3. автоматически, через реестр ОС Windows (инфа в реестр вносится с помощью предварительно подготовленного reg-файла). Ковырялки в реестре? Фуу….
4. используя файл свойств (файл properties в формате имя=значение). Лично мне нравится именно данный способ. Он наиболее простой и имеет важную особенность: он позволяет иметь несколько разных настроенных редиректоров на компе. Также мы избавлены от необходимости работы с реестром Виндовзз – на мой взгляд это есть плюс!

Все параметры и опции, используемые для конфигурирования модуля редиректора (и указанные ниже в примерах конфигурационных параметров и в примерах файлов), будут расписаны далее в документе. Теперь подробнее распишем каждый способ конфигурирования.

*Способ* ***a****. Ручная конфигурация с помощью реестра Виндовс.*

Запускаем regedit и понеслось…

1. Создать в реестре новый ключ “**HKEY\_LOCAL\_MACHINE\SOFTWARE\Apache Software Foundation\Jakarta Isapi Redirector\1.0**”
2. Добавить строковый ключ **extension\_uri** со значением **/jakarta/isapi\_redirect.dll** (наименование файла библиотеки)
3. Добавить строковый ключ **log\_file** со значением – указанием на файл лога (например **c:\tomcat\logs\isapi.log**)
4. Добавить строковый ключ **log\_level** со значением уровня логгирования (возможные значения: debug, info, error, emerg)
5. Добавить строковый ключ **worker\_file** со значением, указывающим полный путь к файлу **workers.properties** (например **c:\tomcat\conf\workers.properties**)
6. Добавить строковый ключ **worker\_mount\_file** со значением, указывающим полный путь к файлу **uriworkermap.properties** (например **c:\tomcat\conf\uriworkermap.properties**)

Для того, чтобы изменения отразились в системе обычно даже не надо перегружать машину. Но для полной уверенности, что все выполнено правильно рекомендую машину все-таки перегрузить.

*Способ* ***b****. Автоматическая конфигурация с помощью реестра Виндовс.*

Вышеуказанные ручные операции с реестром можно заменить созданием файла **\*.reg** следующего содержания:

**REGEDIT4**

**[HKEY\_LOCAL\_MACHINE\SOFTWARE\Apache Software Foundation]**

**@=""**

**[HKEY\_LOCAL\_MACHINE\SOFTWARE\Apache Software Foundation\Jakarta Isapi Redirector]**

**@=""**

**[HKEY\_LOCAL\_MACHINE\SOFTWARE\Apache Software Foundation\Jakarta Isapi Redirector\1.0]**

**@=""**

**"extension\_uri"="/jakarta/isapi\_redirect.dll"**

**"log\_file"="D:\\JAVASERVER\\isapi\_redirector\\log\\isapi\_redirect.log"**

**"log\_level"="info"**

**"worker\_file"="D:\\JAVASERVER\\isapi\_redirector\\conf\\workers.properties.minimal"**

**"worker\_mount\_file"="D:\\JAVASERVER\\isapi\_redirector\\conf\\uriworkermap.properties"**

Пустые строки между строками данных вставлены не случайно – именно такой файл необходимо создать для конфигурирования. Этот файл проверен для ОС Windows 2003 Server Standard, для других ОС файл может отличаться.

Также как и в предыдущем случае, обычно для вступления изменений в силу нет необходимости перегружать машину, но все равно рекомендую это сделать – для того, чтобы быть уверенным в правильности своих действий…

*Способ* ***c****. Конфигурирование с помощью файла свойств.*

Есть и еще один способ конфигурирования isapi редиректора (также заменяющий шаги по созданию ключей реестра). Этот способ позволяет вообще не использовать реестр для хранения конфигурации. Редиректор умеет читать конфигурационный файл свойств (офигеть!), используя значения из него вместо параметров из реестра (причем конфигурационный файл используется вместо реестра при наличии параметров и там, и там – т.е. конфиг-файл имеет приоритет перед реестром). Эта возможность позволяет нам использовать несколько редиректоров с независимой конфигурацией на одном сервере. При инициализации редиректор проверяет наличие файла свойств (properties file) и использует его, если находит. Файл свойств создается в том же каталоге, где находится библиотека редиректора и его имя совпадает с именем библиотеки (расширение файла свойств **.properties**), например если библиотека называется **isapi\_redirect-1.2.30.dll**, то файл свойств должен называться **isapi\_redirect-1.2.30.properties**. Пример простого файла свойств (он также есть в пакете с исходниками редиректора – исходники качать с сайта tomcat.apache.org):

*# Configuration file for the Jakarta ISAPI Redirector*

*# The path to the ISAPI Redirector Extension, relative to the website*

*# This must be in a virtual directory with execute privileges*

*extension\_uri=/jakarta/isapi\_redirect.dll*

*# Full path to the log file for the ISAPI Redirector*

*log\_file=c:\tomcat\logs\isapi\_redirect.log*

*# Log level (debug, info, warn, error or trace)*

*log\_level=info*

*# Full path to the workers.properties file*

*worker\_file=c:\tomcat\conf\workers.properties*

*# Full path to the uriworkermap.properties file*

*worker\_mount\_file=c:\tomcat\conf\uriworkermap.properties*

Заметки к файлу свойств.

* символы ‘\’ – не эскейп-символы (экранирующие), а обычные символы
* строки комментариев начинаются с символа ‘#’
* начиная с версии редиректора 1.2.27 в окружение, используемое в файлах свойств, добавлены две переменные:
* JKISAPI\_PATH – полный путь к isapi-редиректору
* JKISAPI\_NAME – имя библиотеки редиректора БЕЗ расширения

Таким образом, можно использовать следующие конструкции

# Use the logs in the installation path of ISAPI Redirector

log\_file=$(JKISAPI\_PATH)\$(JKISAPI\_NAME).log

Гы-гы-гы:

*А в доке на сайте ошибка: предыдущая строка приведена так:*

*log\_file=$(ISAPI\_PATH)\$(ISAPI\_NAME).log*

*так не работает – проверял… Замечание актуально на: 11.02.2011*

Я считаю, что конфигурирование с помощью файла свойств – самый оптимальный метод, т.к. не требует возни с реестром и правки файла \*.reg для разных версий ОС… Но выбор всегда за тобой, мой юный друг. ☺

Важное замечание. Значения соответствующих конфигурационных параметров необходимо настраивать в соответствии с необходимостью – указанные значения носят характер примера.

Итак, продолжаем конфигурирование нашего редиректора (мы закончили настройку самого редиректора, теперь необходимо сказать серверу IIS, что у нас есть такая штучка - редиректор).

1. Теперь нам необходимо убедиться, что наш сервер IIS также работоспособен. В списке служб системы необходимо найти службу «Служба веб-публикации» и проверить, что она запущена. Далее, с помощью веб-браузера идем по адресу <http://localhost> – должна отобразиться любая страничка (в зависимости от настроек нашего ИИСа), кроме странички с ошибкой (типа 404 – не найдено). Все прошло успешно? Тогда продолжим! Если же ничего не получилось – необходимо исправить ошибки и запустить-таки ИИС – без этого дальше что-либо делать бессмысленно.
2. С помощью консоли управления IIS добавить новый виртуальный каталог на сервере, имя каталога обязательно должно быть jakarta. Физический путь (расположение) каталога должен указывать на директорию, в которой находится файл библиотеки редиректора (например, в нашем случае файл **isapi\_redirect-1.2.30.dll**). При создании виртуального каталога необходимо разрешить «выполнение приложений ISAPI или CGI».

**ВАЖНОЕ ЗАМЕЧАНИЕ.** Для тех, кто разбирается. Имя виртуального каталога может быть и другим (не **jakarta**), но тогда значение конфигурационного параметра **extension\_uri** (в реестре или в файле пропертей – в зависимости от используемого способа конфигурирования)должно ему соответствовать, например:

**extension\_uri=/[имя созданного виртуального каталога]/[имя файла редиректора]**

Это касается всех способов конфигурирования – через реестр, через конфиг-файл с пропертями. Эта фишка помогает иметь на машине несколько редиректоров с разными конфигами.

1. Используя консоль управления IIS необходимо добавить **isapi\_redirect-1.2.30.dll** как фильтр на сайте. Для ОС Windows 2003 server standard делается это так: в оснастке управления «Диспетчер служб IIS» жмем правой кнопкой крысы на «Веб-узлы» (установка фильтра ISAPI для всех веб-узлов) или «Веб узел по умолчанию» (установка фильтра ISAPI для конкретного веб-узла), выбираем «Свойства», затем вкладку «Фильтры ISAPI» и добавляем фильтр (непосредственно библиотеку). Наименование фильтра должно указывать на его задачу (необязательно, но желательно, например tomcat), исполняемый (необходимый для работы фильтра) файл – наш файл **isapi\_redirect-1.2.30.dll**
2. При использовании IIS 6 (для ОС Windows 2003 server и новее) необходимо выполнить дополнительные шаги:

используя консоль управления IIS необходимо добавить Jakarta Isapi Redirector в Web Service Extensions (расширения веб-сервисов)

* правой кнопкой мыши нажимаем на «Web Service Extensions» (в оснастке «Диспетчер служб IIS» ветка с наименованием «Расширения веб-служб») и выбираем «Add a new Web Service Extension» («Добавить новые расширения веб-служб…»)
* вводим tomcat как имя расширения (совпадение имен расширения веб-сервиса и фильтра ISAPI необязательно!). Что с русским языком? Не пробовал, но теоретически новое имя расширения веб-служб можно вводить и на русском.
* добавляем файл **isapi\_redirect-1.2.30.dll** как необходимый файл (required)
* установить статус расширения «разрешено» («allowed») – ставим такую галочку
* жмем ОК (вроде все…)

1. Перезагружаем IIS, убеждаемся, что сервис стартовал нормально и наш фильтр tomcat помечен зеленой стрелочкой, направленной вверх. На этом шаге рекомендую перегрузить машину – на всякий случай… ☺

**Еще пара важных замечаний.**

* фильтр после установки и перезагрузке сервисов IIS может быть не загружен сервером сразу, а только после первого обращения к серверу или к ресурсу Томката (столкнулся с этой странностью на Win2003server standard, вдруг тебе встретится…)
* обязательно должны существовать все используемые ресурсы: каталог для логов, файлы свойств workers.properties и uriworkermap.properites, значения конфиг-параметров редиректора в реестре или в файле .properties. при отсутствии каких-либо ресурсов возможно появление непредсказуемых и трудно выявляемых ошибок.
* если настройки вашего Томката и/или редиректора отличаются от начальных (установленных изначально в только что скачанном Томкате и файлах workers.properties и uriworkermap.properties, прилагающихся к нему), то может потребоваться более тонкая/сложная настройка. Удачи!

Вот и все. Теперь можно стартовать сервер Tomcat и запросить у IIS страницу /examples (**http://localhost/examples/jsp/index.html**). Если мы получили эту страницу – все в порядке, конфигурирование удалось! Если же нет – обломс… и надо искать ошибки… (как искать ошибки – напишу потом как-нить…)

**ВАЖНО!**

1. При конфигурировании редиректора любым из описанных способов необходимо учесть следующее: параметр **extension\_uri**, который указывает на URI библиотеки ISAPI расширения (isapi\_redirect.dll), должен указывать на файл, который реально существует на диске, т.е. если файл библиотеки называется isapi\_redirect-1.2.30.dll, то значение параметра должно быть /jakarta/isapi\_redirect-1.2.30.dll. А указанный каталог jakarta – это наименовение созданного в IIS виртуального каталога (см. выше). Резюмируем. Параметр **extension\_uri** указывает на относительный путь к реальному файлу библиотеки ISAPI, путь указывается относительно созданного на сервере IIS виртуального каталога, с указанием имени самого каталога. Как-то так… Если непонятно – сочувствую. ☺
2. Редиректор может не стартовать сразу. Не торопитесь, ищите ошибку. Частая ошибка – каталог для логгера должен реально существовать на диске (файл лога - необязательно), сам редиректор его (каталог) создавать не будет, а без лога редиректор не стартует. Ошибка банальна, но трудноуловима.

**Расширенное конфигурирование редиректора**

**Настройка работы расширения веб-сервера.**

ISAPI редиректор читает свою конфигурацию из реестра, из ветки с наименованием:

**"HKEY\_LOCAL\_MACHINE\SOFTWARE\Apache Software Foundation\Jakarta Isapi Redirector\1.0"**

или из конфигурационного файла – см. инфу о конфигурировании выше. Ниже приводится список основных конфигурационных параметров редиректора (параметры для реестра и для файла конфигурации, наименования параметров одинаковы). Если параметр является обязательным – это указано (в тексте и **жирным** шрифтом).

Оригинальная документация находится тут:

<http://tomcat.apache.org/connectors-doc/reference/iis.html>

В оригинальной документации есть полный список параметров конфигурации модуля редиректора для связи IIS + Tomcat.

|  |  |
| --- | --- |
| **ПАРАМЕТР** | **ОПИСАНИЕ** |
| **extension\_uri** | Строковое значение, указывающее на isapi расширение веб-сервера (см. выше более детально), обычно имеет значение типа: /jakarta/isapi\_redirect.dll Параметр обязательный. |
| **log\_file** | Создаваемый/используемый файл лога. Путь абсолютный (например: c:\tomcat\logs\isapi\_redirect.log). Параметр обязательный. |
| log\_level | Уровень журналирования (возможные значения: debug, info, warn, error, trace). |
| log\_rotationtime | Время смены (ротации) лог-файла в секундах. Установка значения в 0 (значение по умолчанию) запрещает смену (ротацию) лог-файла, основанную на времени. |
| log\_filesize | Макс. размер лог-файла в мегабайтах, после достижения которого лог-файл будет сменен. Установка значения в 0 (значение по умолчанию) запрещает ротацию лог-файла, основанную на размере файла. Значение может иметь опциональный суффикс M (напр., 5=5M). Если указано ненулевое значение параметра log\_rotationtime, то значение данного параметра будет проигнорировано. |
| **worker\_file** | Строковое значение, указывающее полный абсолютный путь к файлу конфигурирования воркеров (workers.properties). Пример: c:\tomcat\conf\workers.properties. Параметр обязательный. |
| **worker\_mount\_file** | Строковое значение, указывающее полный абсолютный путь к файлу маппинга ресурсов и воркеров (uriworkermap.properties). Пример: c:\tomcat\conf\uriworkermap.properties. Параметр обязательный. |
| worker\_mount\_reloaded | Целочисленное значение, указывающее время в секундах, через которое периодически будет перечитан конфигурационный файл **worker\_mount\_file.** |

**Конфигурирование воркеров (файл workers.properties)**

Оригинальная дока находится тут: <http://tomcat.apache.org/connectors-doc/reference/workers.html>

Воркер (worker) Кошака (Tomcat) – это экземпляр Томката, который ожидает запроса на выполнение сервлета или другого контента на стороне веб-сервера. Например, у нас может быть веб-сервер типа АПАЧ, перенаправляющий запросы к сервелетам на процесс Томката (воркер – worker), который запущен как бы «за спиной» апача. Это – самый простой случай, можно также сконфигурировать множество воркеров Томката для обслуживания (выполнения) сервлетов «за спиной» какого-либо веб-сервера.

Воркеры Томката конфигурируются в файле свойств **workers.properties** и ниже представлено описание для работы с этим файлом.

Строки файла определяют свойства, формат следующий <имя>=<значение> (<name>=<value>). Точки используются для представления иерархии свойств, неверные директивы заносятся в лог при старте сервера и не позволят серверу правильно работать. Некоторые директивы запрещены или устарели, но они по-прежнему действуют, но их необходимо заменить соответствующими «преемниками» (обычно указывается в доке). Некоторые директивы разрешено использовать в конфиге несколько раз, для каждой такой директивы это указано в ее описании. Пробелы в начале и конце <имени> или <значения> игнорируются, комментарии начинаются с символа ‘#’.

Глобальные директивы.

|  |  |
| --- | --- |
| **ДИРЕКТИВА** | **ОПИСАНИЕ** |
| worker.list | Разделенный запятой список имен воркеров, используемых компонентом (редиректором). При старте, плагин веб-сервера создает экземпляры (instance) воркеров, чьи имена указаны в данном списке. Имена воркеров в списке используются для маппинга на них запросов. Директива может быть использована множество раз. Значение по умолчанию: ajp13. |
| worker.maintain | Интервал обслуживания пула соединений воркеров (в секундах). При установке целого положительного значения редиректор сканирует все соединения всех воркеров (указанных в списке workers.list) и проверяет валидность соединений.  Кроме того, любой балансировщик (load balancer, см. ниже в доке) выполняет глобальное обслуживание каждые worker.maintain секунд. При глобальном обслуживании счетчики загрузки разрушаются и воркеры, находящиеся в состоянии ошибки проверяются (обслуживаются) в течении времени recover\_time (директива балансировщика нагрузки, см. ниже). |

Свойства и директивы воркеров

Каждая директива конфигурации (свойство) для воркера состоит из трех слов, разделенных точкой:

*worker.<worker name>.<directive>=<value>*

Первое слово – всегда worker, второе слово – имя воркера, выбираемое нами (в случае балансировки нагрузки имя воркера приобретает дополнительное значение, см. соответствующий раздел), третье слово – наименование директивы воркера. Имя воркера может содержать ТОЛЬКО буквенно-цифровые символы **[a-z][A-Z][0-9][\_\-]** и является чувствительным к регистру символов.

Переменные.

В файле свойств также можно использовать переменные <variable\_name>=<value>. Точки в именах разрешены, но нельзя использовать имена переменных, совпадающие с именами стандартных директив. Поэтому имена переменных никогда не должны начинаться со слова worker. Использовать переменную очень просто – просто вставляем $(variable\_name) в любое место на стороне переменной в строке свойства (в правой части оператора присваивания значения). Если переменная не была определена до использования, будет предпринят поиск ее значения в окружении процесса.

Наследование свойств.

Часто возникает необходимость использовать одно и тоже значение свойства для нескольких воркеров. Для уменьшения дублирования строк конфига (так реально проще работать – с маленьким файлом конфига) существует возможность наследования свойств от одного воркера к другому, или от шаблона воркера к реальным воркерам.

Директива “reference” позволяет копировать конфигурацию между воркерами по пути иерархии. Если воркер **hren** установит директиву **worker.hren.reference=worker.zopa**, то он унаследует все свойства воркера **zopa**, за исключением тех, которые были установлены исключительно для воркера **hren**. ВАЖНО! Значение данной директивы не только имя ссылаемого воркера, а полный префикс, включая слово worker.

Для использования шаблона просто объявите его как обычный воркер, только не добавляйте в список worker.list и как балансируемый воркер в любой балансировщик. Шаблон может не содержать обязательных директив.

Ссылки (наследование свойств) могут быть использованы во множестве участков конфига в иерархическом порядке.

Список основных директив воркеров (полный список директив см. в онлайновой доке).

1. **Директивы соединений**

|  |  |
| --- | --- |
| **ДИРЕКТИВА** | **ОПИСАНИЕ** |
| **type**  (default: ajp13) | **Это основная и обязательная директива каждого воркера. Без нее воркер не работоспособен.** Тип воркера, может быть ajp13, ajp14, jni, lb, status. Тип воркера определяет набор директив, применимых к воркеру.  AJP13 – предпочитаемый тип, используемый редиректором для сообщения между веб-сервером и Томкатом. Этот тип воркера использует сокеты как канал связи.  JNI – запрещен к использованию.  lb – Load Balancer – виртуальный воркер – балансировщик нагрузки (запросов) для одного или нескольких реальных воркеров (описание см. дальше).  status – виртуальный «статусный» воркер, который позволяет просматривать статус работы редиректора и серверов Томкат (см. описание). |
| host  (default: localhost) | Имя хоста или айпишнег прикладного (backend) экземпляра Томката. Удаленный Кошак должен поддерживать стек протокола ajp13. Имя хоста может включать номер порта с помощью разделителя : (например: localhost:8009). |
| port  (default: 8009) | Номер порта удаленного экземпляра Кошака, слушающего запросы. Значение по умолчанию зависит от типа воркера, для ajp13 порт по умолчанию 8009, для ajp14 – 8011 (конфигурирование номера порта для ajp-коннектора – см. доку по конфигурированию Томката, директива <Connector>). |
| connection\_pool\_size | Параметр определяет количество соединений к AJP-backend серверу, которое обслуживается как единый пул соединений. Также параметр лимитирует количество таких соединений, которые может создать дочерний процесс веб-сервера.  Используется только для многопоточных веб-серверов (типа Апач, ИИС, Нетшкаф/Солнце).  Данный параметр должен отражать количество параллельных запросов, которые может послать один процесс (не поток, а процесс!) веб-сервера к бэкенд-Томкату. Обычно это число равно числу потоков, генерируемых (используемых) одним процессом веб-сервера. Для Апача редиректор автоматически получает это число (число потоков на один процесс веб-сервера) и устанавливает размер пула автоматом. Для ИИСа значение по умолчанию равно 250, для Нетшкафа = 1.  Тут еще словеса от производителя, которые было лень переводить:  We strongly recommend adjusting this value for IIS and the Netscape/Sun to the number of requests one web server process should be able to send to a backend in parallel. You should measure how many connections you need during peak activity without performance problems, and then add some percentage depending on your growth rate. Finally you should check, whether your web server processes are able to use at least as many threads, as you configured as the pool size.  Do not use connection\_pool\_size with values higher then 1 on **Apache 2.x prefork** or **Apache 1.3.x**! |
| connection\_pool\_min\_size | Минимальный размер обслуживаемого пула соединений. Значение по умолчанию: (connection\_pool\_size + 1)/2  Do not use connection\_pool\_size with values higher then 1 on **Apache 2.x prefork** or **Apache 1.3.x**! |
| lbfactor  (default: 1) | Параметр используется только для воркеров – участников балансировщиков нагрузки (те «реальные» воркеры, которые балансируются через специальный виртуальный воркер балансировщик). Параметр имеет целочисленное значение и означает сколько данный воркер будет работать (обрабатывать запросы) или, говоря проще, рабочую квоту данного воркера. |

1. **Директивы балансировки (load balancing)**

Балансёр (load balancer) – это виртуальный воркер, который реально не соединяется с воркерами Томката. Вместо этого, он ответственен за управление (управляет) несколькими «реальными» воркерами. Воркер становится балансёром, если его тип указан как lb (см. выше о типах воркеров).

Директивы балансёра определяют параметры, необходимые для конфигурирования и создания воркеров, которые будут соединяться с кластером серверов Томкат. Каждый узел кластера (сервер Томкат) должен иметь своего воркера.

Работа балансера заключается в следующем:

* создание экземпляров воркеров для веб-сервера
* используя load-balancing factor осуществлять балансировку воркеров. Более высокое значение lbfactor означает более мощную машину, которая может обработать больше запросов.
* запускать запросы одной сессии на одном Кошаке (привязка сессии, session stickiness)
* находить обломавшиеся (failed) воркеры, завершать на них запросы и перенаправлять их (запросы) на другой воркер из списка балансировки данного балансёра.

Общий результат работы балансера – воркеры под управлением одного балансера сбалансированы по нагрузке (слово баланс уже бесит!) и резервированы (fall-backed) – падение одного из процессов Кошака (одного сервера) не убивает все приложение.

ВАЖНО! Если необходимо использовать т.н. «приклеивание сессий» (session stickiness), то… Стоп. Что такое «приклеивание сессий» (session stickiness)? Все очень просто. Sticky session это такая фича большинства решений (программных) балансировки нагрузки для набора веб-серверов, которая позволяет для отдельной сессии направить запросы на один физический узел (машину), который обработал первый запрос данной сессии – т.е. «приклеить» сессию пользователя к одному физическому узлу (машине). Использование данной фичи позволяет избежать потери данных выполняющейся сессии между запросами (если запросы одной сессии будут направлены на разные сервера (узлы, машины). Однако это решение имеет и недостаток, о котором важно помнить. Т.к. все запросы пользователя будут направлены на узел, обработавший первый запрос сессии пользователя, то это может послужить причиной неравномерной балансировки нагрузки между серверами (узлами, машинами).

Такс. Продолжим с нашими липкими сессиями (sticky session). Так вот, мой маленький друг, если ты хочешь пользовать такие сессии (это хорошо и правильно!), то необходимо устанавливать разные значения атрибута jvmRoute в элементах <Engine> конфигурации Томкатов (в файле server.xml) – значения должны быть разными для всех используемых экземпляров Томката. Помимо этого, имена воркеров, которые управляются балансёром, должны быть эквивалентны (равны) значениям атрибутов jvmRoute соответствующих экземпляров Томката (с которыми воркеры соединены).

Также рекомендую посмотреть директиву воркера (обычного реального воркера) route.

Ниже приводится таблица основных (не всех) свойств воркера lb (load balancer). Все свойства – смотри доку! ☺

|  |  |
| --- | --- |
| **ДИРЕКТИВА** | **ОПИСАНИЕ** |
| balance\_workers | Разделенный запятой список реальных воркеров, которыми управляет данный балансёр. Директива может быть использоване несколько раз для одного балансёра.  ВАЖНО! Т.к. указанные в директиве воркеры будут доступны через балансировщик, нет необходимости включать их в свойство **workers.list** |

1. **Расширенные директивы для воркеров**

Для воркеров существует еще большой набор директив, но в нашем случае это излишне. Если необходимо – эти директивы можно найти сайте **tomcat.apache.org**

**Пример файла workers.properties**

Данный пример иллюстрирует файл workers.properties, используемый мной. Сконфигурировано 3 воркера для 3-х разных серверов Томката, висящих на разных портах машины.

*# Список воркеров Томката. На каждый Томкат - свой воркер.*

*worker.list=first, second, third, jkstatus*

*# --- Воркер первого Томката*

*worker.first.type=ajp13*

*worker.first.host=localhost*

*worker.first.port=8010*

*# ---Воркер второго Томката*

*worker.second.type=ajp13*

*worker.second.host=localhost*

*worker.second.port=8011*

*# ---Воркер третьего Томката*

*worker.third.type=ajp13*

*worker.third.host=localhost*

*worker.third.port=8012*

*# ---Статус-воркер*

*worker.jkstatus.type=status*

**Конфигурирование маппинга ресурсов (файл uriworkermap.properties)**

Оригинальная документация тут: <http://tomcat.apache.org/connectors-doc/reference/uriworkermap.html>

Перенаправление запросов от веб-сервера к Томкату настраивается с помощью задания маппингов (карты правил отображения). Один маппинг – связывает тип запросов с воркером. Запрос описывается с помощью URI шаблона, воркер – с помощью имени воркера.

Файл **uriworkermap.properties** – это механизм задания маппингов, который работает для всех серверов. Есть и другие механизмы для соединения Томката и веб-серверов (механизмы, специфичные для этих серверов) , но они обсуждаются отдельно – в описании конфигурирования коннекторов Томката для этих серверов.

Имя файла – обычно **uriworkermap.properties** (мы с тобой, дружок, будем использовать именно это имя!).

Формат строки файла, комменты, URI-шаблоны

Файл имеет строковую структуру, одна строка – одно правило (маппинг). Один маппинг – это пара, состоящая из URI-шаблона и имени воркера, соединенная знаком =

/myapp=myworker

Строка коммента начинается с символа “#”. Лидирующие и конечные пробел в строке, вокруг URI-шаблона и имени воркера обрезаются.

# This is fucking comment

/myasshole=assholeworker

В URI-шаблоне можно использовать три специальных символа – “\*”, “?”, “|”. Поясним их. Символ “\*” – любое количество любых символов в URI, “?” – один любой символ. Каждый URI-шаблон начинается с символов “/”, “\*” или “?” и, опционально, может иметь префиксом любую комбинацию символов “!” и “-” (см. чуть дальше). Символ “|” означает «композицию» маппингов, т.е. задание маппинга “X|Y” эквивалентно двум маппингам “X” и “XY” (см. в примерах). Символ “!” означает правило исключения из правил маппинга URI, если запрос совпадает с правилом исключения, то такой запрос не перенаправляется на Томкат (обычно это используется для обработки статического контента веб-сервером).

Правило является правилом исключения, если оно предваряется символом “!” (также см. примеры). Правило исключения перекрывает обычное правило маппинга, если имена воркеров в обычном правиле и в правиле исключении совпадают. Также возможно задать правило исключения для всех воркеров, используя символ “\*” в качестве имени воркера для правила (я уже надоел, да? но – см. примеры, там все есть…), более сложные шаблоны в правилах исключениях запрещены.

Еще одна возможность – запрещение правил. Это может пригодиться, если наш веб-сервер объединяет правила из разных источников и мы хотим запретить правила, определенные раньше. Также, т.к. файл **uriworkermap.properties** динамически перечитывается, мы можем использовать запрещение для временного запрета на перенаправление запросов. Правило является запрещенным, если оно предваряется символом “-” (опять же – см. примеры). Правило исключения также может быть запрещено, если оно предваряется символами “-!”.

О приоритетах правил – см. доку по данному коннектору (редиректору), мне разбираться неохота… Также правила имеют некоторые расширения, которые могут пригодиться – о них также рекомендую почитать доку.

Примеры задания маппингов:

# Маппинг URI /myapp1 и всего контента под URI /myapp1/:

/myapp1=myworker-a

/myapp1/\*=myworker-a

# Более удобная запись вышеприведенного маппинга с использованием символа “|”

# (маппинг URI /myapp1 и всего контента под URI /myapp1/)

/myapp1|/\*=myworker-a

# Маппинг всех URI, оканчивающихся на соотв. суффикс:

\*.jsp=myworker

\*.do=myworker

# Правило исключения подкаталога static:

!/myapp/static|/\*=myworker

# Правило исключения некоторых суффиксов:

!\*.html=myworker

# Правило исключения всех подкаталогов static для всех воркеров:

!/\*/static|/\*=\*

# Правило исключения некоторых суффиксов для всех воркеров:

!\*.html=\*

# Мы не на обслуживании. Запрещаем это правило (оно может быть объявлено

# где-то ранее – а на данном этапе мы его выключаем!)

-/\*=maintenance

**ВАЖНЫЕ ПРАКТИЧЕСКИЕ ЗАМЕЧАНИЯ.**

1. Если файл маппинга ресурсов и воркеров (uriworkermap.properties) перечитывается сервером периодически, то файл конфигурации воркеров (workers.properties) не перечитывается автоматически, а только при рестарте служб IIS (возможно и при перезагрузке isapi расширения на сервере).

Настройка таймаута сессий в Кошаке.

Для этого находим файл [CATALINA\_HOME]/conf/web.xml и в этом файле строчки:

<session-config>

<session-timeout>30</session-timeout>

</session-config>

Тут и указываем таймаут для ВСЕГО сервера, таймаут указывается в МИНУТАХ! Если же необходимо настроить таймаут для конкретного приложения – необходимо использовать тот же тег (см. выше), но уже в файле web.xml указанного приложения. Также возможно программно поменять таймаут сессии, для этого необходимо в web-приложении реализовать интерфейс Listener (создать класс):

public class MyHttpSessionListener implements HttpSessionListener

{  
  public void sessionCreated(HttpSessionEvent event)

{  
    event.getSession().setMaxInactiveInterval(15\*60); //in seconds  
  }  
  public void sessionDestroyed(HttpSessionEvent event) {}  
 }

А в файле web.xml приложения не забыть зарегистрировать указанного слушателя:

<webapp>  
...  
  <listeners>  
    <listener-class>com.example.MyHttpSessionListener</listener-class>  
  </listeners>  
</webapp>

**Разные тонкости работы сервера Tomcat**

1. Когда сервер Томката останавливается, то по умолчанию все активные сессии сохраняются на диск. Потом, при следующем запуске, они восстанавливаются. В большинстве случаев этот механизм не нужен. Чтобы его отключить, надо переопределить настройку **<Manager pathname=”” />** в файле server.xml (для конкретного контекста) или в файле context.xml (для всего сервера) – имеется в виду тот файл context.xml, который находится в каталоге [CATALINA\_HOME/conf].

**Технология JNDI (серьезное слово ☺).**

Что же такое JNDI – Java Naming and Directory Interface? Попытаемся объяснить просто. Это объединенное дерево ресурсов, ствол которого это непосредственно сервер (его служба), а каждая ветка дерева – один ресурс. Тип ресурса – любой: БД, файл, сетевой диск и т.п. В чем преимущество? Преимущество в таком использовании ресурсов то, что управляет ими (ресурсами) и доступом к ним сервер, а не мы, т.е. если наш ресурс – БД, то в своем приложении мы только указываем конфигурацию, а все остальное делает за нас сервер – установка соединение, разделение доступа, балансирование нагрузки. Также мы можем разделять ресурсы с другими приложениями – например, совместное использование БД несколькими приложениями на сервере. Вот она – польза контейнера воочию! ☺ А теперь приведем пример такого конфигурационного файла:

<?xml version="1.0" encoding="UTF-8"?>

<Context path="/">

<Resource name="jdbc/RoyalEstateDS"

auth="Container"

type="javax.sql.DataSource"

username="root"

password="mysql"

driverClassName="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost:3306/RoyalEstate"

maxActive="20"

maxIdle="5"/>

</Context>

[описание конфига ресурса для одного приложения]

[описание и пример конфига ресурса общего для нескольких приложений]

[детальное описание конфигурирования ресурса]

**Соединение с Базой данных. Соединение, источники данных, пул соединений. Как написать меньше кода, или все уже написано до нас. ☺**

**Шаблон МОДЕЛЬ-ПРЕДСТАВЛЕНИЕ-КОНТРОЛЛЕР (Model-View-Controller).**

**Безопасность нашего каталога. Кто на новенького?**

**Вкусняшка. Кто облегчит нам жизнь? Фреймворк STRUTS – реализация шаблона MVC.**

[архитектура Struts]

[описание Action]

1. Форма (ActionForm). Эта форма очень сильно напоминает DTO компонент (см. далее), у этих двух классов обычно совпадают основные поля – это те поля, которые соответствуют полям таблицы. Однако набор дополнительных полей может сильно отличаться, т.к. у этих компонентов разное назначение – DTO компонент предназначен для виртуального представления реальных данных, а форма фреймворка Struts предназначена для редактирования данных и их отображения пользователю. Таким образом, DTO компонент это часть модели данных (Model), а форма Struts – это часть контроллера (Controller).
2. Вернемся к нашей форме. Как уже было сказано, основные поля класса формы совпадают с полями таблицы, для взаимодействия с данными которой предназначена форма. Дополнительные поля формы определяются самим разработчиком в зависимости от конкретной задачи и назначения формы (форма для редактирования данных может сильно отличаться от формы для простого отображения данных).
3. Также как и у DTO компонента (см. ниже) все поля формы должны быть инициализированы (либо сразу при объявлении, либо с помощью конструкторов). Значения по умолчанию для полей – такие же, как и для полей DTO компонента.
4. Для всех полей должны быть предусмотрены методы getXXX() и setXXX(), наименования методов – в соответствии с соглашением об именовании методов bean-компонентов.
5. При необходимости реализуется метод reset(), который сбрасывает нужные поля формы после работы с данными.

[пример формы Struts с методом reset()]

1. Действие (Action). В этом классе обязательно реализуется метод execute().

[детальное описание метода]

1. Маппинг (конфигурирование) фреймворка Struts. Основным конфигурационным файлом Struts является файл struts-config.xml, который располагается в каталоге WEB-INF. В этом файле описываются все формы (ActionForm), все URL, которые будут обрабатываться нашим приложением и все действия, выполняемые при вызове того или другого URL (адреса).
2. Описание бинов форм.
3. Описание маппингов (привязок) действий.

[описание конфигурирования, файлы конфигурации, примеры кода]

1. [константы для действий add-edit-delete-undelete]

Добавление страницы/действия в проект с фреймворком STRUTS.

1. Определяем действие и необходимые для него данные.
2. Создаем класс-форму, наследующий от класса Struts ActionForm. В созданном классе определяем поля, которые будут хранить необходимые на странице данные. Для всех полей определяем методы доступа getXXX() и setXXX(), названные в соответствии с соглашением об именовании для bean-классов Java.
3. Если необходимо, то определяем методы reset() и validate() для данного класса. Метод reset() необходим для сброса данных формы, а метод validate() – для проверки этих данных. Более подробное описание см. в документации Struts.
4. Создаем класс-действие, наследующий от класса Struts Action. Данный класс будет осуществлять действия, выполняемые пере переходом на нужную страницу jsp, а также обрабатывать данные, введенные/измененные на этой странице (это стандартное поведение Action'a, в каждом конкретном случае поведение этого класса зависит от основного потока приложения и конфигурируется отдельно).
5. В классе-действии (Action) определяем метод execute(), который будет основным выполняющимся методом класса. А что будет внутри этого метода? См. далее.

[примеры кода компонентов] [описание доделать в деталях]

**Маленький стукач. Модуль log4j – журналирование приложения.**

Описать выбор имени логгера – почему статическая переменная, а не имя пакета верхнего уровня.

Мы опишем сейчас два способа конфигурирования логгера приложения. Первый способ – конфигурирование логгера с помощью файла свойств (вида имя=значение) и второй способ – конфигурирование в инициализационном сервлете программно. Возможно, существуют еще способы конфигурирования логгера log4j, но за ними отправляю тебя, читатель, к специальной литературе («безумству храбрых поем мы песню…» ☺).

Конфигурирование log4j с помощью файла свойств.

Конфигурироване log4j программно.

Тонкости использования логгера log4j

1. Файловый аппендер DailyRollingFileAppender иногда не может переименовать используемый файл лога (для осуществления его ротации), т.к. несколько экземпляров модуля log4j пишут в один файл (на системе с ОС Windows запрещено переименовывать (просто невозможно) открытый файл, для систем на базе семейства ОС Unix вроде работает – не проверял!). Т.е необходимо использовать другой файловый аппендер или искать другое решение. Я использую следующее решение – аппендер DatedFileAppebder стороннего производителя. Полную инфу и сам аппендер см. тут: <http://minaret.biz/tips/datedFileAppender.html>

**Модель данных (доменная модель), часть 1 - данные. Создание DTO компонентов.**

[рисунки и примеры кода]

DTO компоненты - это первая часть модели данных, виртуальное представление реальных данных, находящихся в постоянном (persistence) хранилище – СУБД, файлы на диске и т.п.

1. Определяем количество таблиц БД – скорее всего столько же будет наших DTO компонентов. DTO – Data Transfer Object – эти объекты (компоненты) необходимы для представления данных из таблиц БД, один объект представляет собой одну запись из таблицы, поэтому количество полей в объекте будет больше или равно количеству полей в таблице. В объект заносятся все публикуемые/изменяемые поля таблицы. Также в объекте могут участвовать виртуальные поля – им нет соответствия в исходной таблице, эти поля вычисляются в процессе выборки данных из одной или нескольких таблиц.
2. В некоторых случаях один DTO компонент может соответствовать более чем одной таблице – архитектура и состав набора DTO компонентов определяются в зависимости от конкретной задачи. Таким образом, наш набор DTO компонентов является частью модели данных приложения – самими данными, поэтому состав набора компонентов и структура самих компонентов (поля) определяются разработчиком.
3. Соответствие типов данных полей DTO компонента и типов данных полей таблицы также может быть определено в зависимости от задачи, но обычно соответствие носит прямой характер:

* Все символьные поля таблицы имеют соответствующие эквивалентные поля компонента типа String.
* Цифровые поля таблицы (дробные/целые) отображаются соответствующими типами данных – int, double и т.п. При этом рекомендуется использовать простые (примитивные) типы данных, вместо соответствующих им объектных: int вместо Integer и т.п.
* Булевские (логические) поля таблицы могут отображаться в типы boolean или byte(самый короткий целочисленный). Также с данным типом проще работать.
* Бинарные даные – отображаем в байтовые массивы (все зависит от задачи, разработчик решает сам!) – byte[].
* Поля таблицы, содержащие дату, время или и дату, и время отображаются в соответствующие им типы данных (см. доку по языку). Также часто применяется отображение таких полей таблицы в строковые поля компонентов.

1. Наименование полей DTO компонентов должно соответствовать наименованиям полей таблицы (или таблиц) БД, записи которой (которых) отображает данный компонент (так удобнее с ними работать). Для виртуальных (вычисляемых) полей компонентов имена подбираются произвольные – лучше, соответствующие данным, хранящимся в этих полях.
2. Поля в DTO компоненте должны быть обязательно проинициализированы – при их объявлении или в конструкторе по умолчанию (конструкторе без параметров – если нет инициализации полей при объявлении, такой конструктор обязательно должен быть, если же поля проинициализированы при объявлении, такой конструктор не нужен).
3. При необходимости объявляется один или несколько конструкторов с параметрами – для создания уже инициализированного экземпляра класса.
4. Значения полей по умолчанию задаются следующим образом:

* Для полей-идентификаторов (первичных и внешних ключей) значение по умолчанию -1 (минус единица).
* Для всех символьных полей – значение по умолчанию null.
* Для числовых полей (целых, дробных) – значение 0 (или выбирается в зависимости от задачи).
* Для булевских (логических) полей – ложь(false) (или также выбирается в зависимости от задачи).
* Дата, время, дата+время – для этих полей значением по умолчанию является null.
* Для полей с бинарными данными - значением по умолчанию является null.

1. В DTO компоненте должны быть методы доступа к данным (полям) – методы getXXX() и setXXX(). Для полей «только для чтения» методы setXXX() не нужны. Для наименования этих методов обязательно соблюдать соглашения об именовании для bean-классов java (см. доку по языку).
2. В каждом DTO компоненте должен быть реализован метод isEmpty(), который возвращает значения ЛОЖЬ/ИСТИНА в зависимости от наличия данных в данном экземпляре класса DTO (заполнение/пустота соответствующих полей). Если данные есть – метод возвращает ЛОЖЬ, если же нет – ИСТИНА. Алгоритм проверки пустоты экземпляра определяется в зависимости от задачи, но обычно этот алгоритм похож на ограничения, наложенные на данные в самой таблице – это ускорит работу системы, т.к. пустой объект даже не будет отправлен в БД, он будет отсечен на этапе проверки на пустоту (методом isEmpty()). В проверке экземпляра класса на пустоту не участвует поле id (идентификатор), т.к. если объект добавляется в таблицу, он будет заполнен данными, а его идентификатор пуст (< 0 – меньше нуля) и при такой проверке объект добавлен в БД не будет.
3. При разработке любого класса DTO следует реализовывать (переопределять) метод *toString() –* данный метод определен в классе самого верхнего уровня – *Object –* и позволяет получить визуальное (строковое) представление состояния экземпляра объекта – значение полей экземпляра. Если данный DTO компонент является наследником не класса *Object*, то целесообразно включить вызов данного метода класса-родителя (если таковой метод у родителя есть) перед указанием значений полей данного экземпляра класса. Данный метод будет полезен при диагностике ошибок и отладке приложения – при различных ошибках можно легко и просто получить состояние любого DTO компонента – это существенно ускорит и упростит процесс поиска ошибок. Стиль метода (стиль формирования строкового представления) должен быть унифицированным для всего приложения в целом – это позволит быстро идентифицировать в логах приложения сообщения о состоянии какого-либо класса (ну удобнее так, поверьте!).

**Модель данных (доменная модель), часть 2 – действия с данными. Создание DAO компонентов.**

DAO компоненты это вторая часть модели данных – манипуляции с данными. Набор DAO компонентов обеспечивает нам возможность манипулирования данными, представленными набором DTO компонентов – создавать виртуальные выборки из хранилища данных, обработав ввод пользователя, добавить или изменить данные в постоянном хранилище.

1. DAO компоненты должны общаться с постоянным хранилищем данных (в нашем случае – СУБД), поэтому каждый компонент из набора должен иметь возможность независимого соединения с СУБД (часто возникает ситуация, когда один компонент вызывает методы другого и у каждого из них должно быть свое соединение с СУБД) . Т.к. хранилище данных общее, то и метод доступа для всех компонентов будет один и тот же, поэтому разумно будет вынести метод получения соединения с СУБД в отдельный компонент модели данных нашего приложения и использовать этот компонент во всех DAO компонентах. Самое простое решение – написать общий код получения соединения с СУБД (javax.sql.Connection) в классе, от которого затем унаследовать все наши DAO компоненты.

[пример кода]

Также возможен вариант создания класса-утилиты, который будет выдавать по запросу соединение с СУБД. Этот класс-утилиту можно сделать статическим.

[пример кода]

1. Почему не написать в этом общем компоненте (назовем его CommonDAO) и код получения экземпляра класса Statement? Ответ прост – каждое открытое соединение с СУБД необходимо закрывать, а если CommonDAO будет выдавать Statement, то можно просто запутаться в соединениях и какое-либо из них не закрыть, и, плюс к этому, необходимо будет писать в CommonDAO код, закрывающий соединение и хранящий на него ссылку (по этой ссылке и закрываем соединение), причем, CommonDAO должен будет хранить все ссылки на соединения, выданные для работы и необходим будет механизм закрытия именно нужного соединения, чтобы не закрыть соседское рабочее соединение - все это сильно усложнит наш компонент – мы ведь стремимся к простоте и написанию меньшего количества кода, правда, лентяи? Если же CommonDAO будет выдавать соединения (java.sql.Connection), то хранить в нем необходимо будет только ссылку на источник данных (javax.sql.DataSource) – об источниках данных см. выше. И тогда следить за закрытием соединения будет тот, кто его использует – метод DAO компонента. Этот способ далеко не единственный, но мне он кажется самым простым. Дерзайте, совершенству нет предела!

[более развернуто на несколько пунктов]

1. Еще немного об источнике данных в классе CommonDAO. Поле, хранящее ссылку на этот источник данных, можно сделать статическим (это даст единственную ссылку на источник данных для всех экземпляров данного класса – и для всех экземпляров его потомков) и его инициализацию внести в конструктор. При инициализации поля со ссылкой на источник данных проверять – если поле еще не инициализировано (хранит значение null), то только тогда его инициализировать. Поле должно быть закрытым (private) и методов доступа к нему быть не должно. Кроме конструктора, в нашем классе CommonDAO будет еще один метод – метод получения соединения с СУБД. Метод должен проверять – не пуста ли ссылка на источник данных – и только после этого получать у него (источника данных) соединение и отдавать его (соединение).

[пример кода компонента CommonDAO]

1. Также как и DTO компоненты, DAO компоненты в большинстве случаев соответствуют таблицам БД, т.е. для каждой таблицы будет свой DAO компонент для работы с ее записями (и этот DAO компонент будет соответствовать DTO компоненту данной таблицы). В сложных случаях, когда один DTO компонент объединяет две и более таблиц, соответствующий ему DAO компонент также будет объединять в себе манипуляции с данными двух и более таблиц. Не следует перегружать компоненты – т.е. такой случай лучше разбить на несколько: одна пара DTO+DAO соответствует пяти или шести таблицам.
2. В каждом DAO компоненте необходимо реализовать метод(ы) чтения и манипуляции с данными. Какие это методы? Обычный набор методов DAO компонента:

* Метод поиска всех записей findAll(). Метод возвращает список соответствующих данной таблице DTO компонентов, обычно это типизированный класс ArrayList<DTO>. Если в результате работы метода будет найдена одна или более записей, метод возвращает их список, если же ни одной записи найдено не будет, то метод должен возвращать значение null.

[о возвращаемых значениях для всех методов]

* Метод(ы) поиска записей по параметрам findByXXX() – например, поиск по идентификаторам, по наименованиям, по различным признакам. Если предполагается, что данный метод будет возвращать более одной записи, то метод должен возвращать список DTO компонентов, соответствующих данной таблице, обычно это типизированный класс ArrayList<DTO>. Если же метод будет возвращать одну запись, то это будет соответствующий класс DTO.
* Метод для добавления одной записи в таблицу (или таблицы) и метод изменения записи в одной или нескольких таблицах: create(DTO) и update(DTO). Эти методы можно совместить в одном: change(DTO) – этот метод будет сам выбирать действие – добавление или изменение записи в зависимости от параметров. Принимать решение о том, использовать два метода или один комбинированный, необходимо в зависимости от задачи.
* Метод(ы), которые будут необходимы для конкретной задачи – наличие и количество этих методов зависят от разработчика, но, в большинстве случаев, стандартного набора методов бывает достаточно.

[в любом случае архитектура и набор методов зависит от разработчика]

1. В каждом DAO компоненте необходим логгер. Логгер в режиме отладки (DEBUG) должен обозначать точки входа во все методы DAO компонента.

[пример кода получения логгера и его использования]

1. В каждом методе DAO компонента, который конструирует sql-запрос будет правильным выводить (в режиме отладки – DEBUG) сконструированный запрос в лог – так проще всего будет проверить, что сконструированный запрос правилен – его можно выполнить в какой-либо среде работы с СУБД.
2. В каждом методе DAO компонента обязательно использование конструкции try-catch-finally. Описание этих блоков:

* TRY. Данный блок должен начинаться с открытия соединения с СУБД, также в него должны быть включены все операции, которые могут возбуждать ИС.
* CATCH.
* FINALLY. В этом блоке всегда необходимо освобождать ресурсы, которые были получены в блоке TRY (соединения, курсоры и т.п.)

[пример кода использования try-catch-finally]

[пример кода DAO компонента]

1. Обработка исключений. Есть много способов использования обработки ИС в java, для данного случая рассмотрим лишь два (кто хочет знать больше – читайте другую литературу по java, там все уже разобрано и подано).

[обработка ИС внутри DAO компонентов]

[передача ИС наверх для обработки и вывода соответствующих сообщений пользователю]

**Инициализация приложения.**

Для автоматической инициализации нашего приложения необходим сервлет, который будет загружаться при старте самого приложения и выполнять необходимые действия. Что это за действия? Перечислим лишь некоторые из них: проверка/создание необходимых каталогов, проверка соединения с БД, инициализация модуля журналирования приложения, проверка доступности различных ресурсов. Это далеко не весь список – инициализационный сервлет может выполнять любые действия, все зависит от конкретной задачи и фантазии разработчика. Теперь к делу! Для создания инициализационного сервлета выполняем следующие действия:

* *при использовании среды разработки Intellij IDEA:*

1. создать сервлет с названием InitServlet и поместить его в пакет system иерархии классов нашего приложения. Имя сервлета может быть произвольным, но рекомендуется именно «говорящее» имя для того, чтобы можно было сходу разобраться в проекте (не вспоминая долго и мучительно для чего же нужен сервлет с именем MySuperPuper).
2. в конфигурации сервлета указать «Load on startup» (загружать при старте приложения) и порядковый номер загружаемого сервлета – сервлеты загружаются последовательно, в соответствии с присвоенными им номерами. Наш сервлет должен иметь самый высший приоритет, т.к. он должен загружаться ПЕРЕД стартом приложения, соответственно номер нашего сервлета будет 0. Остальные сервлеты приложения загружаются после инициализации приложения – т.е. после данного сервлета.
3. проверить файл web.xml нашего приложения, в нем должна появиться запись следующего вида:

**<servlet>**

**<servlet-name>InitServlet</servlet-name>**

**<servlet-class>my.application.system.InitServlet</servlet-class>**

**<load-on-startup>0</load-on-startup>**

**</servlet>**

1. переопределить в исходном коде сервлета метод **public void init()** – этот метод будет автоматически выполнен при загрузке сервлета. Наш сервлет будет наследником класса HttpServlet и переопределять его метод **init()**, но в нашем методе **init()** нет необходимости вызывать метод суперкласса (родителя – HttpServlet) – super.init(), однако для каждой задачи это решение должно приниматься отдельно!

**Трудолюбивый муравей. Использование средства автоматизации сборки ANT.**

В процессе создания приложения публикации БД у нас будет много повторяющихся и однотипных операций. В конце концов постоянное их повторение надоедает. Что делать? Как всегда – за нас уже подумали и все сделали, нам остается только освоить и использовать соответствующий инструмент автоматизации сборки проекта.

**Java и веб-сервисы. Что и как?**

**Move it to java ee part/chapter**

Веб сервисы набирают свою популярность. Не будем отставать от всего мира…

Для технологии Java существует много спецификаций и фреймворков для построения веб-сервисов и клиентов для них. Основные (ну или те, что знаю я):

* Фреймворк Apache AXIS. Основная особенность – конфигурирование веб-сервиса и клиента веб-сервиса выполняется с помощью xml-файлов.
* JAX-WS (Java API for XML Web Services) – Java API для создания веб-сервисов на Java. Является частью платформы Java EE компании Sun. Основная особенность – как и другие Java EE API's от Sun использует аннотации для конфигурирования. Эталонная (стандартная) реализация (или эталонный пример реализации) JAX-WS (Reference Implementation of JAX-WS) – разработана как проект с открытым исходным кодом и является частью проекта GlassFish (сервер Java EE с открытым исходным кодом). Называется данная реализация – JAX-WS RI (Reference Implementation).

На заметку. Java 6 update 4 (JDK) включает в себя данный фреймворк версии 2.1!

Для того, чтобы начать работать с фреймворком JAX-WS в Java 6 (update 4 и выше) нет необходимости фреймворк устанавливать – он и так уже входит в поставку (версия 2.1). Однако, если необходима более новая версия – все же придется установить фреймворк самостоятельно.

**JAX-WS (версия 2.2.1). Начало работы – установка/удаление.**

Данный материал является вольным переводом официальной доки, поставляемой вместе с JAX-WS RI 2.2.1 – в пианиста не стрелять!

Используемое ПО.

Нам понадобятся:

* Установленная Жаба JDK 5.0 Update 2 и выше. Есть. Мы как раз будем использовать JDK 6.0 Update 14.
* Установленный Ant 1.6.2 и выше. Тоже есть. Будем использовать Ant 1.8.1.
* Tomcat 5.5.x. В доке написано, что фреймворк тестировался на данном Томкате – думаю, что и на нашем Томкате версии 6.0.x все прекрасно заработает.

(добавлено позже: на Tomcat 6.0.32 все прекрасно заработало!)

Инсталляция фреймворка.

Все очень просто и никаких сюрпризов нет. Вот шаги инсталляции:

* Распаковать скачанный архив (неожиданно!)
* Для инсталляции на сервер Tomcat:
* Установить переменную окружения CATALINA\_HOME в значение, указывающее на расположение сервера Кошака
* В каталоге фреймворка запустить команду “ant install”. Это просто копирование необходимых для работы фреймворка на Томкате файлов \*.jar из каталога lib в каталог CATALINA\_HOME\shared\lib (lib/\*.jar -> CATALINA\_HOME\shared\lib). Для того, чтобы каманда выполнилось необходимо наличие файла build.xml в корневом каталоге фреймворка!
* Если мы работаем на JDK6 (наш Кошак под ним), то необходимо также запустить команду “ant install-api” (или воспользоваться командой “ant help” для дополнительной справки – некое подобие справки есть в самом скрипте build.xml).

Если планируется только вызов (использование) уже существующего веб-сервиса, то в качестве альтернативы можно использовать скрипты wsimport и wsgen из каталога bin поставки фреймворка.

Работа на JDK6. Особенности.

Выделено в отдельный пункт, т.к. JDK 6u4 (1.6.0 update 4) уже включает в себя часть данного фреймворка: JAX-WS 2.1 и JAXB 2.1 API и RI (Recent Implementation – эталонная реализация). Для того чтобы использовать JAX-WS RI 2.2.1 на JDK6 необходимо, воспользовавшись механизмом Java Endorsed Standards Override Mechanism (ESOM) переопределить библиотеки jaxws-api.jar и jaxb-api.jar (положить их в каталог JDK\_HOME\jre\lib\endorsed). Подробнее о механизме Java ESOM читай выше в данном документе.

Удаление фреймворка (а вдруг понадобится?).

Если фреймворк нам надоел – его можно удалить и забыть как страшный сон! Для этого необходимо выполнить два простых шага:

* В каталоге фреймворка (где находится файл build.xml) выполнить команду “ant uninstall”. Это удалит установленный фреймворк с сервера Tomcat. Если фреймворк на сервер Кошака не устанавливался – делать ничего и не надо.
* Если мы используем JDK6 (наш случай), то необходимо вручную удалить файлы jaxws-api.jar и jaxb-api.jar из каталога JDK\_HOME\jre\lib\endorsed (подробнее о Java ESOM читай выше). Сам каталог endorsed удалять не надо! Данные действия необходимы, только если мы переопределяли указанные библиотеки используя Java ESOM.

WSDL-файл.

Данный файл является универсальным описателем веб-сервиса (как и следует из его формата – Web Services Definition Language). С помощью данного файла строится клиент любого веб-сервиса (с применением утилиты wsimport). Опубликованный веб-сервис автоматически генерирует данный файл при обращении на специализированный URL.

Структура wsdl файла

<definitions>

<types></types>

<message></message>

………………………………

<message></message>

<portType></portType>

<binding></binding>

<service></service>

</definitions>

По разделам:

**types**

Описание типов данных определенной системы данных – например XML-схемы. Документ, определяющий типы данных – XSD (XML Schema Definition). Данная секция содеожит, указывает или импортирует XSD. Если данная секция пуста – сервис использует только простые типы данных, такие как xsd:string и xsd:long

**message**

Определение сообщений, реализуемых сервисом.

**portType**

Данная секция представляет сервис как именованные операции, каждая из которых из одного или более сообщений (секция message).

**binding**

В данной секции определения переходят от абстракций к конкретике. Эта секция сродни реализации интерфейса в Java (сам интерфейс – это секция portType). В секции binding определяются следующие параметры сервиса:

* транспортный протокол для отправки и приема soap-сообщений (обычно http или (реже) smtp)
* стиль сервиса (rpc/document, по умолчанию - document)
* формат данных, используемый в soap-сообщениях (literal/encoded, обычно используется формат literal, т.к. формат encoded не совместим с руководящими принципами (нормативами) WS-I – Web Services – Interoperability, подробнее см. http://www.ws-i.org).

**service**

Данная секция описывает одну или более конечных точек (endpoint), в которых доступна функциональность сервиса (сумма всех операций сервиса).

**Утилита wsgen.**

Данная утилита предназначена для генерации классов поддержки для серверной части веб-сервиса, благодаря классам, генерируемым данной утилитой, возможна динамическая генерация wsdl-файла работающим сервисом. Строго говоря, данная утилита генерирует артефакты JAX-B (классы), которые позволяют отображать типы данных Java на XML. JAX-B = Java Api for XML Binding.

**Утилита wsimport.**

Данная утилита предназначена для генерации классов поддержки клиента веб-сервиса по wsdl-файлу. wsdl-файл может располагаться локально или быть доступен через специализированный URL.

**Немного о самих веб сервисах.**

Веб сервисы можно разделить на две большие группы: SOAP-based и REST-style. Различия между ними не очень большие, т.к. SOAP-based сервисы, доставляемые по протоколу HTTP – это специальный случай REST-style сервисов.

SOAP.

Изначально аббревиатура SOAP расшифровывалась как Simple Object Access Protocol, но сейчас интуитивно эта аббревиатура ближе к Service Oriented Architecture (SOA) Protocol. Сейчас SOAP это диалект XML, в котором документы являются сообщениями. Для SOAP веб сервисов существуют стандарты (под управлением W3C), обширный инструментарий и большой набор библиотек.

RESTful.

REST означает REpresentational State Transfer. Для RESTful веб сервисов нет стандартов, бедный инструментарий, очень небольшой набор библиотек. Отличие от SOAP веб сервисов (ключевое, но не единственное) в том, что запросы клиента должны быть не обязательно XML документами (в SOAP и запрос клиента и ответ сервиса это XML документ), а могут быть обычными HTTP запросами.

Обычно клиентами веб сервисов являются приложения без графического интерфейса, иногда, обычно при тестировании, клиентом может быть веб браузер.

Главная идея веб сервисов – «языковая прозрачность», т.е. клиент и сервер могут быть написаны с применением разных технологий (языков программирования).

**Создание веб-сервиса. Серверная часть.**

1. Непосредственно написание кода самого сервиса – методы сервиса. При этом обычно используются аннотации:

* для SOAP веб сервисов: @WebService и @WebMethod
* для RESTful веб сервисов: @WebServiceProvider

1. Для написанного веб-сервиса необходимо сгенерировать классы поддержки – это выполняется с помощью утилиты wsgen. Классы поддержки необходимы для динамической генерации wsdl-файла для данного сервиса.
2. Созданный сервис публикуется. Для этого можно использовать различные варианты: для теста подойдет класс javax.xml.ws.Endpoint, для выпуска конечного продукта рекомендую использовать J2EE контейнеры – подойдет Tomcat, Gassfish и т.п.
3. С помощью опубликованного сервиса генерируем wsdl-файл. Для тестовой генерации можно воспользоваться публикацией сервиса с помощью javax.xml.ws.Endpoint. Важно помнить, что при изменении исходного кода веб-сервиса меняется и генерируемый данным сервисом wsdl-файл.

**Создание веб-сервиса. Клиентская часть.**

1. Первое, что необходимо сделать для написания клиента веб-сервиса – это получить wsdl-файл. Получить его возможно с помощью опубликованного сервиса или непосредственно (в виде файла) у поставщика/разработчика данного сервиса.
2. После получения wsdl-файла нам необходима утилита wsimport, которая генерирует классы поддержки для клиента веб сервиса. Используя данную утилиту непосредственно генерируем классы поддержки клиента, при этом рекомендуется сохранить исходники сгенерированных файлов – это поможет при написании клиента сервиса (необходимо проанализировать исходники сгенерированных классов).
3. В случае простейшего клиента (самая простая реализация клиента):

* находим среди классов исходников, сгенерированных утилитой wsimport, класс с наименованием XXXService – это наш сервис. Создаем экземпляр сервиса:

*XXXService service = new XXXService();*

* затем находим среди тех же классов исходников (сгенерированных wsimport), интерфейс сервиса – это SEI – Service Endpoint Interface – интерфейс сервиса. С помощью данного интерфейса нам необходимо получить «порт» сервиса – конечную точку для вызова сервиса. Делаем это так:

*ServiceInterface port = service.getPort(ServiceInterface.class);*

при этом service – экземпляр сервиса, полученный нами в предыдущем пункте.

* вуаля! Используя ссылку port мы можем вызывать наш сервис из созданного клиента. Напоминаю – это самый простой способ быстрого создания клиента для веб сервиса, который позволит потестировать сервис быстро. В случае production системы скорее всего потребуется написание более сложного клиента.

## Глава 33. Тестирование программ. JUnit 4.x.

Для того, чтобы наши программы были написаны хорошо, одного желания программиста недостаточно. Необходимы дополнительные усилия и дополнительные инструменты. Одним из таких дополнительных инструментов является фреймворк JUnit – фреймворк для автоматического тестирования приложения и его модулей. Тесты с использованием данного фреймворка пишет сам программист и выполняются они вручную или при автоматизированной сборке приложения – Maven, Ant и т.п.

Данный фреймворк предназначен для юнит-тестирования и интеграционного тестирования (существуют и другие виды тестирования, но данные – основные, ИМХО).

**Юнит-тестирование.**

Каждый юнит-тест должен тестировать только один юнит (тестируемый объект). При использовании нескольких объектов приложения в одном юнит-тесте трудно будет разобраться, в каком именно объекте возникает ошибка (если она возникает). Если тестируемый объект взаимодействует или использует другие объекты, то для тестирования необходимо создавать специальные тестовые объекты (имитировать окружение тестируемого объекта) – в этом вся суть юнит-тестирования.

**Интеграционное тестирование.**

Другая форма тестирования - интеграционное тестирование – тестирование взаимодействия нескольких объектов приложения между собой. В случае интеграционного тестирования используются реальные объекты приложения. Однако наличие интеграционных тестов для объектов не отменяет создание юнит-тестов для данных объектов.

Версия JUnit 4.x в данный момент является последней. В этой версии, в отличие от предыдущей (3.8.x), методы класса-теста помечаются специальными аннотациями, нет необходимости реализации какого-либо специального интерфейса или наследования от какого-либо класса. Т.к. JUnit 4.x использует аннотации, то для его использования необходима Java 1.5 и выше.

### Добавление JUnit 4 в проект.

Как добавить JUnit в свое приложение и как его использовать? Об этом я и расскажу ниже, вкратце.

1. Для использования JUnit фреймворка необходимо добавить его библиотеку в classpath приложения или в список зависимостей проекта Maven:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

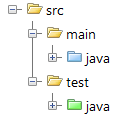
<version>4.8.2</version>

<scope>test</scope>

</dependency>

Scope лучше всего указать test, т.к. данный фреймворк нам будет нужен только на момент тестирования приложения – не при его запуске.

1. Классы тестов приложения лучше всего расположить в отдельном от основного приложения каталоге, при этом сохранив такую же структуру пакетов – для того, чтобы классы тестов могли получать доступ к полям и методам тестируемого класса, объявленным как protected (не private). При этом мы получаем структуру каталогов, похожую на эту (указана часть структуры каталогов проекта, относящаяся к исходному коду, скриншот взят из среды Intellij IDEA 11):



Т.е. в каталоге **src/main/java** находятся пакеты и классы с исходным кодом приложения, а в каталоге **src/test/java** находятся пакеты и классы с исходным кодом тестов приложения, причем структура пакетов в обоих каталогах совпадает.

*Для чего это стоит делать? Хранить исходный код приложения и код юнит-тестов вместе – плохая практика при наличии большого количества классов – такой код очень сложно будет читать/поддерживать. Рекомендуется разделять исходный код и код тестов по разным папкам, но сохранять для них один пакет (package) – это необходимо для того, чтобы тестовые классы имели доступ к внутренностям тестируемых классов (например, к методам объявленным как protected).*

1. В папке для тестовых классов приложения создаем необходимые классы-тесты, каждый класс-тест создается в том пакете, в котором лежит тестируемый класс в основном каталоге. Методы класса-теста помечаются специальными аннотациями. Пример класса-теста:

public class FactorialTasksTest {

@Test

public void testMethod() {

[some test code]

}

}

1. Если в проекте много классов-тестов, то их можно объединить в тестовую сюиту – один тест класс, который может запустить много других классов тестов. Тест сюита также помечается специальными аннотациями. Пример класса тест-сюиты:

@RunWith(value=Suite.class)

@Suite.SuiteClasses(value={Test1.class, Test2.class, Test3.class})

public class TasksTestSuite {

}

В аннотации **@Suite.SuiteClasses** перечисляются (через запятую) все классы тестов, которые должна запускать данная сюита.

Теперь все готово для написания кода тестов нашего приложения.

### Небольшой FAQ/рекомендации.

* В фреймворке JUnit есть дополнительные аннотации, которые довольно широко используются: @After, @AfterClass, @Before, @BeforeClass. Давайте рассмотрим их подробнее.

**@BeforeClass.** Если необходимо однократно выполнить какие-либо действия (например, создание окружения тестирования) ПЕРЕД выполнением всех методов-тестов данного тестового класса (тестирующего), то необходимо создать public static void метод и аннотировать его @BeforeClass.

**@AfterClass.** Если необходимо однократно выполнить какие-либо действия (например, освобождение ресурсов окружения тестирования) ПОСЛЕ выполнения всех методов-тестов данного тестового класса (тестирующего), то необходимо создать public static void метод и аннотировать его @AfterClass.

**@Before.** Иногда необходимо перед выполнением метода-теста создать какие-либо объекты или выполнить какие-либо действия. Когда мы аннотируем public void метод данной аннотацией (@Before), это означает, что аннотированный метод будет выполнен ПЕРЕД каждым тестовым методом (методом, аннотированным @Test).

**@After.** Также бывает необходимо освободить какие-либо ресурсы или выполнить другие действия после выполнения меода-теста. Когда мы аанотируем public void метод данной аннотацией (@After), это означает, что аннотированный метод будет выполнен ПОСЛЕ каждого тестового метода (метода, аннотированного @Test).

Важно помнить, что количество методов с вышеуказанными аннотациями (@Before/@BeforeClass/@After/@AfterClass) неограниченно, но при наличии более одного метода с любой из аннотаций порядок их выполнения неизвестен.

**@Ignore(value=”ignore reason”).** Данная аннотация указывает, что данный тестовый метод необходимо пропустить (не выполнять). Значение value можно опустить, но делать этого не следует – всегда указывайте причину игнорирования (пропуска) тестового метода.

* Имя тестового метода обычно соответствует шаблону testXXX, где XXX – наименование тестируемого метода тестируемого класса. Если один тестируемый метод имеет несколько тестирующих методов, то рекомендуется следующая схема наименования тестовых методов – testXXXYYY, где XXX – наименование тестируемого метода, YYY – аспект тестирования. Также рекомендуется комментировать (минимально) тестовые методы.
* При использовании методов assert\* используйте те методы, которые принимают в качестве первого параметра значение String – описание, которое будет выведено на экран если данный метод assert\* провалится (fail). Это позволит точно идентифицировать место возникновения ошибки.
* Один юнит-тест должен соответствовать одному тестовому методу (@Test). Нет необходимости смешивать в одном методе множество тестов или использовать другие тестовые методы – это сильно исказит картину тестирования и закончится очень сложными тестовыми методами и придется писать тесты для тестовых методов.
* При тестировании методов, которые могут генерировать ИС, можно воспользоваться двумя готовыми путями (путей больше, расскажу о двух):

1. К аннотации @Test тестового метода можно добавить параметр expected и указать класс ИС, которую мы ожидаем в данном тесте. Если ИС не произойдет или произойдет, но класс ИС не совпадет с указанным тест, будет провален. Пример такой аннотации: @Test(expected=NumberFormatException.class).
2. Никаких параметров к аннотации @Test мы не добавляем, просто используем конструкцию try-catch и выполняем необходимый анализ – класс ИС, возникновение ИС, параметры возникшей ИС и т.п.
3. Придумайте свой способ ☺

## Глава 34. Фреймворк Spring 3/4.

### Spring для веб-разработки.

Для того, чтобы добавить в свой веб-проект Spring необходимо в файл web.xml проекта добавить сервлет-диспетчер данного фреймворка (сервлет с маппингом):

<!-- Spring dispatcher servlet -->

<servlet>

<servlet-name>[**spring\_servlet\_name\_without\_spaces**]</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>[**spring\_servlet\_name\_without\_spaces**]</servlet-name>

<url-pattern>[**url\_pattern\_for\_servlet**]</url-pattern>

<!--<url-pattern>/home.do</url-pattern>-->

</servlet-mapping>

Необходимые пояснения:

1. [**spring\_servlet\_name\_without\_spaces**] – имя сервлета без пробелов, обычно указывает на имя приложения.
2. [**url\_pattern\_for\_servlet**] – шаблон для URL, которые будут обрабатываться фреймворком Spring (например: **\*.do** или **/zzz/\*.myext**)

Также необходимо создать файл конфигурации Spring с именем *[****spring\_servlet\_name\_without\_spaces****]-servlet.xml* и расположенный в каталоге WEB-INF приложения. Это имя файла и его расположение можно изменить – см. документацию к фреймворку.

## Глава 35. Spring Security 3.

Использование Spring Security (далее SS) в приложении позволяет создавать защищенные веб-приложения, получив много различных функциональных решений **out of the box** – функцию remember me, логин/логаут с использованием форм и т.п.

SS включается в приложение с помощью специальных фильтров (ServletFilter), через которые пропускаются запросы/ответы между клиентом и сервером.

Для того, чтобы использовать Spring Security 3 в своем приложении необходимо выполнить три простых шага:

1. Надо создать xml конфигурационный файл, который описывал бы компоненты SS (Spring Security). Пример такого файла (пример очень простой, базовый):

*<?xml version="1.0" encoding="UTF-8"?>*

*<beans:beans xmlns="http://www.springframework.org/schema/security"*

*xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"*

*xmlns:beans="http://www.springframework.org/schema/beans"*

*xsi:schemaLocation="*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/security*

*http://www.springframework.org/schema/security/spring-security-3.0.xsd">*

*<http auto-config="true">*

*<intercept-url pattern="/\*" access="ROLE\_USER"/>*

*</http>*

*<authentication-manager alias="authenticationManager">*

*<authentication-provider>*

*<user-service>*

*<user authorities="ROLE\_USER" name="guest" password="guest"/>*

*</user-service>*

*</authentication-provider>*

*</authentication-manager>*

*</beans:beans>*

Обычно данный файл создается в каталоге WEB-INF приложения.

1. В файл web.xml приложения надо добавить фильтр SS **DelegatingFilterProxy**:

*<filter>*

*<filter-name>springSecurityFilterChain</filter-name>*

*<filterclass>*

*org.springframework.web.filter.DelegatingFilterProxy*

*</filter-class>*

*</filter>*

*<filter-mapping>*

*<filter-name>springSecurityFilterChain</filter-name>*

*<url-pattern>/\*</url-pattern>*

*</filter-mapping>*

Этот фильтр будет перехватывать все запросы и пропускать их через механизм SS.

1. Теперь нам необходимо соединить первые два шага вместе: в файл web.xml необходимо добавить ссылку на созданный нами в п.1 xml конфигурационный файл

## Глава 36. Мониторинг приложений/контейнеров Java.

Для правильного представления, что же происходит внутри наших приложений, нам необходимо как бы «заглянуть им в душу», а точнее – в JVM, на которой наше приложение выполняется. Для этого и существуют утилиты мониторинга. Эти утилиты позволяют заглянуть внутрь как JVM для standalone приложения, так и внутрь JVM, работающей внутри контейнера J2EE. Здесь будет рассказано только о немногих из них – бесплатных, входящих в комплект поставки JDK.

### VisualVM.

### JConsole.

## Полезные ссылки и ресурсы.

1. <http://download.oracle.com/javase/tutorial/java/index.html> - сборник Java Tutorials на сайте производителя Java (Oracle).
2. <http://docs.oracle.com/javase/tutorial/uiswing/> - описание разработки пользовательского интерфейса с использованием Swing.

# Часть V. Мир \*nix

## Глава 37. Работа с UNIX-like системами. Основные принципы (на примере Ubuntu Server 12.04/14.04/16.04 LTS) и практики.

В этой главе описаны основные команды/действия/практики для работы с ОС семейства Unix. Описывается настройка только что установленного сервера для нужд разработки – настройка и установка необходимого ПО. В работе использовалась ОС Ubuntu Server 12.04 (семейство ОС Debian). Я не претендую на абсолютную полноту изложения информации и на ее 100% достоверность – все ошибаются, я не исключение ☺. В данном описании изложены мои шаги по настройке удаленной машины для работы с ней в качестве сервера. Предполагается, что доступ к серверу осуществляется по протоколу ssh (удаленный сервер) и мы не имеем доступа к визуальным возможностям (X-window). Если же сервер ssh не установлен на удаленной машине, то его необходимо установить вручную, для этого потребуется физический доступ к серверу – другого способа я не знаю ☺. Для подключения к удаленному серверу я использую свой комп с ОС Windows 7, что не вызывает никаких затруднений. Все программы/утилиты для комфортной работы с удаленным Линукс-сервером есть под ОС Виндовс, большинство из них бесплатны. Основная используемая утилита – ssh-клиет Putty. Также предполагается, что удаленный сервер имеет выход в Инет, т.к. все ПО устанавливается из онлайн репозиториев. Если же сервер не имеет выхода в инет – необходимо вручную скачивать (с другой машины) инсталляционные пакеты из репозиториев и копировать их на наш сервер для установки локально. Но это – отдельная тема.

Иногда, при указании команд, будут использованы символы **$** - это означает ввод команды в командном процессоре Линукса (терминале). С помощью этого символа мы будем отличать команды для командного процессора Линукса от команд/операторов для других приложений. Например, sql-операторы для работы с СУБД Mysql вводятся в клиенте командной строки и мы будет обозначать это символами **mysql>** При вводе команд в терминале ***вводить указанные символы не надо*** – они там уже будут нарисованы.

Если перед символом **$** указан путь – это означает нахождение в соответствующем каталоге.

## Устройство файловой системы

Где и что находится на сервере

## Работа с командной строкой

Работа с командной строкой в \*никс системах отличается от вин систем. Я попытаюсь пояснить некоторые основные моменты, ключевые для никсов.

* Запуск скрипта ./<script> - точка означает, что после выполнения скрипта в текущем сеансе переменные окружения, установленные скриптом, должны остаться, если же скрипт выполнить как <script> то переменные окружения пропадут (останутся как до запуска скрипта).

## Копирование файлов по протоколу ssh.

По протоколу ssh возможно работать с удаленным сервером не только через терминал, но и передавать файлы. Для этого используется SCP – Secure Copy или SFTP – Secure FTP. Т.к. мы не устанавливали FTP сервер, то воспользуемся наиболее простым способом – используем SCP.

Я подключаюсь к удаленной Linux-машине со своего компа, который оборудован ОС Windows 7, поэтому для копирования файлов на Линукс-машину мне необходима программа под Виндофс для работы с SCP. Я использую вот эту – WinSCP, сайт программы - <http://winscp.net>. Разобраться с работой этой программы несложно – предоставляю это чичтателю.

Для подключения к удаленной машине используем те же логин/пароль, что и для терминала.

Интерфейс программы прост и понятен – анаглог Norton Commander/Volcov Commander/Windows Commander. Думаю, разобраться будет просто.

## Настройка сети

* [**ifconfig**] – просмотр текущей конфигурации сети. Если добавить ключ **-a**, то мы увидим конфигурацию сети вместе с неактивными сетевыми интерфейсами (полная конфигурация).

Конфигурация сети (сетевые интерфейсы) описана в файле /etc/network/interfaces, при его редактировании необходимо повышать права до рута (при запуске редактора использовать sudo):

**sudo vi /etc/network/interfaces**

После редактирования рекомендуется перезагрузить систему (для надежности). Однако, можно поробовать и просто перезапустить службы сети (см. ниже).

* [**sudo /etc/init.d/networking restart**] – перезагрузка сетевых служб (необходимо выполнять, если менялись параметры сети/интерфейсов в файле /etc/network/interfaces).

Пример описания сетевого интерфейса (в файле interfaces):

*auto eth0*

*iface eth0 inet dhcp*

Если необходимо посмотреть список DNS серверов, то это можно сделать, заглянув в файл /etc/resolv.conf. Данный файл генерируется специальной службой системы – его не надо править вручную.

### Настройка сети для сервера Ubuntu установленного на Virtual Box.

Для тестирования я часто использую виртуализацию серверов с применением Oracle Virtual Box – бесплатное решение для виртуализации. В нашей локальной сети получить для виртуального сервера Ubuntu выход в интернет и, одновременно, внутренний IP адрес для подключения по ssh, получилось только с использованием двух сетевых интерфейсов (в конфигурации Virtual Box – интерфейс NAT и интерфейс «сетевой мост»). Никакой специальной конфигурации этих интерфейсов не выполнялось. Ниже приводится пример (мой рабочий конфиг) конфигурации сетевых интерфейсов для сервера Ubuntu (файл /etc/network/interfaces):

*# First network interface (inet access)*

*auto eth0*

*iface eth0 inet dhcp*

*# Second network interface (LAN access)*

*auto eth1*

*iface eth1 inet static*

*address 172.18.53.162*

*netmask 255.255.255.0*

## Работа с текстом

### Редактор vi/vim

Для работы с текстовыми конфигами есть множество редакторов/утилит, но самый популярный из них – **vi**. Это текстовый редактор для консоли. Есть и «улучшенная» его версия – **vim** (Vi Improved), немного модифицированная версия этого редактора. Функционал обеих версии оченб похож.

* [**vi**] – запуск редактора vi. Запуск редактора vim осуществляется командой **vim**. В качестве параметра указывается имя редактируемого файла. Если такого файла нет – будет создан новый, если имя файла не указано – также будет создан новый.

Редактор имеет два режима – режим «редактирование» и режим «команды». При старте редактор находится в режиме команд. Вот основные команды для редактора (вводятся напрямую с клавиатуры, редактор должен находиться в режиме команд):

* i – перевод редактора в режим редактирования (если редактор находится в режиме команд, в режиме редактирования – просто буква i).
* <esc> - перевод редактора в режим команд (из режима редактирования).
* :x или :wq – выход с сохранением
* :q – выход, если не было изменений
* :q! – форсированный выход (отмена изменений)
* i, I – вставка перед курсором, перед строкой
* a, A – добавление после курсора, после строки
* o, O – новая строка после, перед
* x, X – удаление символа справа, слева
* dd или :d – удаление строки
* D – удаление до конца строки

Это основные команды, которых мне хватает для правки конфигурационных файлов, остальное есть в нете или в справке ☺.

### Редактор nano

Для работы с текстом из консоли можно также использовать редактор nano. Он удобный и интуитивно понятный (по командам).

## Установка сервера ssh.

Если на машине, которую мы хотим использовать удаленно, нет сервера ssh, то мы не сможем к ней подключиться. Поэтому – установим сервер ssh.

* [**sudo apt-get install ssh**] - установка ssh-сервера из командной строки.
* [**sudo service ssh stop|start|restart**] – управление сервисом (службой) ssh.

Основным файлом конфигурации SSH-сервера является файл */etc/ssh/sshd\_config*, который должен быть доступным для чтения/редактирования только [суперпользователю](http://help.ubuntu.ru/wiki/%D1%81%D1%83%D0%BF%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_%D0%B2_ubuntu) (root). После каждого изменения этого файла необходимо перезапустить ssh-сервер (сервис ssh) для применения изменений.

Установленный нами сервер работоспособен по умолчанию, но его настройки не оптимальны с точки зрения безопасности (для тестирования – подойдут). Более подробно о настройке ssh сервера можно почитать тут: <http://help.ubuntu.ru/wiki/ssh> Подключиться с помощью терминала по протоколу ssh можно с теми же данными (логин/пароль), что использовались для авторизации на сервере (при физическом доступе). Для подключения я использую популярную программку под Виндофс – putty (<http://www.putty.org>). В работе она очень проста и удобна.

## Работа с пользователями и группами

* [**who**] - покажет админу, кто из пользователей сейчас в системе (logged in)
* [**cat /etc/passwd | cut -d":" -f1**] – список всех пользователей системы
* [**cat /etc/passwd | wc -l**] – количество пользователей, включая системные аккаунты (mail etc)
* [**cat /etc/group | cut -d":" -f1**] – список всех групп системы (по аналогии со списком пользователей)
* [**cat /etc/group | wc -l**] – количество групп (также по аналогии с кол-вом пользователей).
* [**groups**] – список групп, членом которых является текущий пользователь

Команды добавления/редактирования/удаления пользователей/групп выполняются с помощью команды **sudo** (повышение прав до уровня рута).

* [**sudo adduser <username>**] – добавление пользователя
* [**sudo useradd <username> -m -s /bin/bash**] – команда useradd дает дополнительные возможности по сравнению с командой adduser. Указанная команда добавляет пользователя, указывает ему стандартный домашний каталог в системном /home и указывает командную оболочку по умолчанию (bash).
* [**sudo passwd username**] – смена пароля для пользователя
* [**sudo addgroup <groupname>**] – добавление группы
* [**sudo adduser <username> <groupname>**] – добавление нового/существующего пользователя в существующую группу. Для выполнения этого действия также можно использовать команду [**sudo usermod -aG <groupname> <username>**]. После добавления пользователя в группу этому пользователю надо перелогиниться – logout->login.

## Системные команды

Большинство системных команд нуждаются в привилегиях рута для своего запуска или корректной работы. Будьте с этим осторожны, это не Винды – Линукс два раза переспрашивать не будет ☺.

* [**sudo <команда с параметрами>**] – повышение прав текущего пользователя до прав «суперпользователя» - root – для выполнения указанной команды. После выполнения команды права пользователя возвращаюися в исходное состояние. Многие системные команды Unix можно выполнить только от имени пользователя root, но работать под этим пользователем опасно. Поэтому для однократного повышения прав текущего пользователя до рута используется команда sudo, поэтому же большинство системных команд/команд установки ПО/открытие конфигов для редактирования и т.п. выполняется с указанием sudo (при описании разных команд это указано). Подробнее о sudo можно почитать тут <https://help.ubuntu.com/community/RootSudo>.
* [**sudo su**] – перманентное повышение прав текущего пользователя до прав рута (для текущего сеанса). Будьте очень осторожны с этим и дважды подумайте – оно вам надо? Работать постоянно от имени рута – чревато!
* [**man <command name>**] – вывод на экран страниц мануала (доки) по указанной команде. Очень полезная штука, позволяет узнать много нового и интересного по используемым инструментам.
* [**sudo netstat -tap**] – позволяет узнать список активных сетевых соединений (если запускать данную команду не от имени рута, может показать не всю инфу).
* [**sudo tar -xvf <archive.tar>**] – распаковка tar-архива в текущий каталог. tar - формат архива (для Unix) без сжатия, такое же наименование носит и утилита для упаковки/распаковки файлов.
* [**cp**] – копирование файлов/каталогов.
* [**mv**] – перенос (move) файлов/каталогов.
* [**ln**] – создание символической ссылки (symbolic link).
* [**reboot**] – перезагрузка системы. Лучше выполнять данную команду от имени рута: **sudo reboot**.
* [**shutdown now**] – немедленный останов сервера. Вместо now можно поставить желаемое время отключения сервера (см. man). Данную команду лучше выполнять от имени рута (с префиксом sudo).
* [**ls**] – список каталогов и файлов в текущем каталоге. При добавлении ключа **-l** – показывает простой список.
* [**clear**] – очистка экрана. Удобно для чтения логов или вывода какой-либо команды.
* [**more**] – вывод на экран содержания файла.
* [**grep**] – поиск подстроки в указанном выводе.
* [**hostname**] – показывает имя данного хоста. С ключом **-f** команда показывает полное доменное имя хоста. Если же запустить данную команду от имени рута и указать через пробел имя – будет установлено новое имя хоста (данной машины).
* [**ps**] – команда для просмотра списка запущенных процессов. Имеет много различных ключей, о которых можно прочитать в мануале. Некоторые примеры:
* **ps -A** - список всех запущенных процессов
* **ps -ef | grep mysql** – поиск среди запущенных процессов службы mysq – СУБД.
* [**rm**] – удаление файла.
* [**chmod**] – смена атрибутов файлов. Стандартные атрибуты: R – read, W – write, X – execute.
* [**gzip**], [**gunzip**] – упаковка/распаковка архива формата zip.
* [**tail**]

## Работа с переменными окружения

Хороший материал на данную тему можно найти тут:

<https://help.ubuntu.com/community/EnvironmentVariables>

Переменные окружения бывают двух видов:

* **session-wide** – пользовательские переменные, которые действуют для текущего пользователя на время его сессии в системе. Самое лучшее место для таких переменных – файл **~/.pam\_environment** (это не скрипт, данный файл содержит выражения присваивания вида name=value, одно на строку). Есть и другие места – см. статью по указанной ссылке.
* **system-wide** – системные переменные окружения, которые действуют для всех пользователей в системе. Лучшее место для этих переменных – файл **/etc/environment**, файл не является скриптом, он содержит выражения присваивания вида name=value, одно на строку. Есть и другие места – см. статью по указанной ссылке.

### Некоторые команды для работы с переменными окружения

* [**printenv**] – вывод всех установленных переменных окружения (для данного пользователя/сеанса)
* [**printenv <VAR\_NAME>**] – вывод значения указанной переменной окружения. Для этого также можно использовать команду [**echo $<VAR\_NAME>**].

## Установка ПО (общие сведения).

Для инсталляции ПО можно воспользоваться графическими утилитами (Ubuntu Server 12.04 идет без графической системы X-Window), но т.к. у нас есть доступ по протоколу ssh (и все!) – мы можем использовать только командную строку (если мы используем серверную версию).

Хорошие ресурсы по установке ПО на Ubuntu (список будет пополняться):

<http://www.dedoimedo.com/computers/aptitude.html>

<http://www.thegeekstuff.com/2009/10/debian-ubuntu-install-upgrade-remove-packages-using-apt-get-apt-cache-apt-file-dpkg>

<https://help.ubuntu.com/community/AptGet/Howto>

Для установки ПО необходимо повышать свои права до рута (предварять команды установки софта командой sudo или выполнить команду sudo su – для того, чтобы стать рутом до конца данного сеанса).

Установка/удаление программ в Убунту (Дебиан) обычно производятся с помощью утилиты **apt** и специальных команд, часть которых будет описана ниже. Софт устанавливается из онлайн-репозиториев, список которых указан в конфигах утилиты **apt**.

### Обновление имеющегося ПО

Установленное в системе ПО можно обновлять, для этого используются команды [**sudo apt-get update**], [**sudo apt-get dist-upgrade**], [**sudo apt-get upgrade**] – проверка того, что утилиты управления пакетами обновлены и в системе установлены последние версии всех пакетов (проверка обновлений и обновление системы). Будьте внимательны – система должна иметь выход в инет (для скачивания обновлений из репозиториев). Выполнение данной команды может занять длительное время. Порядок выполнения команд также важен – вначале необходимо обновить список пакетов и их версий (apt-get update), а уже затем проводить обновление системы (dist-upgrade, upgrade).

### Установка нового ПО, удаление установленного ПО

Для выполнения указанных действий используются команды:

* [**sudo apt-cache search <package name>**] – поиск нужного пакета по некоторым ключевым словам.
* [**sudo apt-get install <package name>**] – установка указанного пакета. Имя пакета должно быть указано точно.
* [**sudo apt-get remove <package name>**] – удаление указанного пакета, конфиг файл пакета остается в системе. В имени пакета можно использовать wildcards (см. следующую команду).
* [**sudo apt-get purge <package name>**] – удаление указанного пакета и связанных с этим пакетом конфигурационных файлов. В имени пакета можно использовать wildcards: **sudo apt-get purge openjdk\*** - удаление всех пакетов, связанных с Open JDK.
* [**sudo apt-get autoremove**] – удаление пакетов, установленных другими пакетами (зависимости), которые больше не нужны.
* [**sudo apt-get check**] – диагностическая команда, проверяет/обновляет список пакетов и выявляет нарушенные зависимости.
* [**sudo apt-get autoclean**] – удаление файлов, связанных с удаленными пакетами.

Как я уже сказал, установка софта происходит из онлайн репозиториев, а это означает, что нам нужен доступ к инету. Если доступ открыт – все ок, можно спокойно устанавливать софт. Если доступа нет вообще – надо скачивать софт вручную (это отдельная тема). Наиболее вероятная ситуация – доступ есть, но через прокси-сервер с авторизацией или без – см. ниже о конфигурировании доступа утилиты apt-get в инет через прокси сервер.

### Настройка apt для использования прокси-сервера

Есть много способов достичь желаемого, я опишу свой – этот способ сработал у меня (он наиболее простой из всех альтернатив – мое мнение). Этот способ позволяет настроить прокси для apt на постоянной основе (не только для текущей сессии).

Выполняем команду [**sudo vi /etc/apt/apt.conf**] – открываем файл конфига утилиты apt в текстовом редакторе vi. В этот файл конфига надо добавить такую строчку:

*Acquire::http::Proxy “http://[username:password@]proxyaddress[:proxyport]”*

Файл сохраняем – все должно работать (на всякий случай можно перегрузить машину – но это необязательно). Для Windows-сетей, использующих доменную организацию, имя пользователя часто указывается в формате: *domain\username*

## Автозагрузка сервисов (процессов) при старте системы

Автоматически загружаемые системой сервисы находятся в каталоге /etc/init.d (точнее говоря, там находятся скрипты для запуска сервисов). Любой скрипт из этого каталога также может быть запущен вручную – для старта/останова/перезапуска соответствующего сервиса.

### Помещение сервиса в автозагрузку

Для этого необходимо выполнить следующее:

* в папке /etc/init.d создать файл скрипта для запуска нашего сервиса
* сделать скрипт выполняемым с помощью команды **sudo chmod +x <script name>**
* добавить скрипт в автозагрузку. Для этого надо перейти в каталог /etc/init.d и выполнить там команду:

**sudo update-rc.d <script name> defaults**

В консольном выводе данной команды мы увидим, что ссылки на данную команду (скрипт) добавлены в автозагрузку.

### Удаление сервиса из автозагрузки

Удаление сервиса из автозагрузки проще, чем добавление:

* выполняем в каталоге /etc/init.d команду:

**sudo update-rc.d -f <script name> remove**

ключ [-f] – «форсированный режим».

## Файловый менеджер MC (Midnight Commander)

Для более комфортной работы с файлами в системе можно установить «визуальный» файловый менеджер. «Визуальный» в кавычках, т.к. визуализация текстовая. Однако достоинство – данный менеджер работает из консоли и по ssh.

Устанавливается командой

**sudo apt-get install mc**

Запуск

**mc**

## Установка и настройка сервера subversion (с утилитами командной строки).

Опишем кратко процесс установки и начальной конфигурации данного сервиса. Все указанные настройки и умолчания можно поменять на свои. Более детальное описание инсталляции и настройки можно найти тут:

<https://help.ubuntu.com/community/Subversion>

<http://odyniec.net/articles/ubuntu-subversion-server>

<http://benrobb.com/2007/01/15/howto-start-subversion-at-boot-on-ubuntu/>

<https://help.ubuntu.com/community/InstallingSoftware>

* [**sudo apt-get install subversion libsvn1 apache2-utils libapache2-svn**] – команда для установки сервера subversion и необходимых дополнительных библиотек. Установка выполняется из онлайн репозитория, поэтому если комп не подключен к инету – установить не получится, будет необходимо вручную скачать из нета нужные пакеты и установить (проблема с зависимостями!).
* Теперь нам необходимо создать группу пользователей **subversion** и добавить в нее себя (я работаю от имени пользователя **vinnypuhh**) и пользователя **www-data** (пользователь Apache). Это делаем командами:

**sudo addgroup subversion**

**sudo adduser www-data subversion**

**sudo adduser vinnypuhh subversion**

После добавления себя в группу subversion необходимо перелогиниться в системе, иначе в текщем сеансе мы в эту группу не попадем.

* Репозиторий subversion будет расположен в каталоге /home/svn, наш тестовый проект назовем myproject. Далее создаем необходимые каталоги:

**sudo mkdir /home/svn**

**cd /home/svn**

**sudo mkdir myproject**

После создания каталогов создаем и сам репозиторий subversion:

**sudo svnadmin create /home/svn/myproject**

Назначаем необходимые права доступа к каталогам:

**cd /home/svn**

**sudo chown -R www-data:subversion myproject**

**sudo chmod -R g+rws myproject**

Команда chown – смена владельца каталога, команда chmod – необходимые разрешения для всех новых добавляемых в проект файлов.

* Доступ к репозиторию subversion может быть осуществлен разными способами.

|  |  |
| --- | --- |
| **Схема доступа** | **Метод доступа** |
| file:/// | прямой доступ к репозиторию (локальный диск) |
| http:// | доступ к репозтиторию по протоколу WebDAV (через веб сервер Apache) |
| https:// | то же, что и http://, но с SSL шифрованием |
| svn:// | доступ по специальному протоколу к серверу svnserve |
| svn+ssh:// | то же, что и svn://, но через SSH туннель |

Мы будем рассматривать доступ по протоколу **svn**.

* Предполагаем, что наш проект (и соответствующий репозиторий) находится в каталоге /home/svn/myproject. Дальнейшие действия опираются на данное предположение.

Для конфигурирования доступа к репозиторию необходимо отредактировать файл /home/svn/myproject/conf/svnserve.conf (svnserve чувствителен к пробелам, не оставляй лишних пробелов в начале строк, иначе файл может быть не прочитан). Надо раскомментировать следующие строчки в этом файле:

*# [general]*

*# password-db = passwd*

После этого, можно создавать список пользователей в файле passwd (находится /home/svn/myproject/conf). Я добавил в этот файл строчку:

*gusevd = <password>*

со своим паролем.

* Сейчас наш сервер не запущен. Его можно запустить из консоли командой:

**sudo svnserve -d --foreground -r /home/svn**

Кратко поясню: [-d] – режим демона, [--foreground] – запустить в foreground, [-r] – корневой каталог для svnserve. После выполнения этой команды будет запущен svnserve, этот процесс будет слушать стандартный порт 3690.

* Довольно неудобно каждый раз при перезапуске сервера запускать svnserve вручную. Поэтому лучшее решение – добавить его в автозагрузку. Верный путь – стартовать svnserve как системный сервис скриптом, который расположен в /etc/init.d. У нас нет готового (встроенного в сервер) скрипта для svnserve, поэтому мы напишем свой (см. ресурсы по установке subversion, там есть примеры скриптов):

*#!/bin/sh*

*#*

*# Dmitry Gusev*

*# subversion daemon start/stop script*

*#*

*test -f /usr/bin/svnserve || exit 0*

*OPTIONS="-d -r /home/svn"*

*case "$1" in*

*start)*

*echo -n "Starting subversion daemon:"*

*echo -n " svnserve"*

*start-stop-daemon --start --quiet --oknodo --user root --exec /usr/bin/svnserve -- $OPTIONS*

*echo "."*

*;;*

*stop)*

*echo -n "Stopping subversion daemon:"*

*echo -n " svnserve"*

*start-stop-daemon --stop --quiet --oknodo --exec /usr/bin/svnserve*

*echo "."*

*;;*

*reload)*

*;;*

*force-reload)*

*$0 restart*

*;;*

*restart)*

*$0 stop*

*$0 start*

*;;*

*\*)*

*echo "Usage: /etc/init.d/svnserve start|stop|reload|restart"*

*exit 1*

*;;*

*esac*

*exit 0*

Все указания каталогов в скрипте надо поменять на свои, также поменяйте пользователя-владельца (здесь использован root).

Сохраняем этот скрипт как /etc/init.d/svnserve и делаем его запускаемым командой

**sudo chmod +x /etc/init.d/svnserve**

*Краткое пояснение: копируем скрипт отсюда, вставляем в текстовый файл svnserve на нашей виндовой машине, с помощью WinSCP копируем его на удаленный сервер в каталог /etc/init.d и делаем его запускаемым командой, указанной выше.*

Инсталлируем скрипт в автозагрузку командой:

**sudo update-rc.d svnserve defaults**

Перезагружаем наш сервер и проверяем доступ к репозиторию.

Нашим сервером svnserve можно управлять и вручную командой:

**sudo /etc/init.d/svnserve start|stop|restart**

### LAMP

Очень известное слово в мире Linux, означает оно набор серверных программ. Компоненты LAMP:

* **Linux** – операционная система, чаще всего используется для серверных решений. Без графической оболочки очень нетребовательна к ресурсам.
* **Apache** – пожалуй, самый популярный веб-сервер. Имеет множество возможностей для настройки.
* **MySql** – простая бесплатная СУБД, одна из нескольких популярных.
* **PHP** – серверный язык сценариев для создания интерактивных веб-страниц и больших порталов.

Эта связка программ является наиболее популярной в сети Интернет для реализации веб-сервера. Ниже мы рассмотрим установку и (немного) конфигурирование компонентов данного набора.

### Установка mysql (используем версию 5)

Описан простой процесс начальной установки и конфигурирования сервера и клиента mysql.

Если сервер mysql может быть уже установлен, то клиентские библиотеки (пакет mysql-client) обычно в поставку сервера не входит – его стоит установить. Также рекомендуется проверить настройки установленного сервера мускула – см. ниже их описание.

Хорошие ресурсы по установке:

<http://www.ithowto.ru/149-ustanovka-mysql-v-ubuntu.html>

<https://ariejan.net/2007/12/12/how-to-install-mysql-on-ubuntudebian>

* [**mysql --version**] – проверка версии установленной mysql. Если нет установленного сервера – мы получим сообщение об этом.
* [**sudo apt-get install mysql-server mysql-client**] – данную команду необходимо выполнить для установки mysql. По умолчанию сервер мускула поставится в каталог /etc/mysql. Во время иснталляции может быть запрошен пароль для пользователя root – его надо задать и запомнить! Не оставляй пароль пустым.
* Для того чтобы к установленному серверу можно было подключиться извне (по сетке, не через инет), надо отредактировать файл /etc/mysql/my.cnf. В этом файле надо найти строчку вида «bind-address = 127.0.0.1» и закомментить ее (в начале строки поставить символ #). После этого необходимо перезапустить сервис mysql.

**sudo nano /etc/mysql/my.cnf**

* [**sudo service mysql start|stop|restart**] – запуск/останов/перезапуск сервиса mysql. После редактирования конфига нам необходимо перезапустить сервис.
* [**sudo apt-get install --reinstall mysql-server**] – если вдруг понадобится переустановить СУБД mysql, то это можно сделать указанной командой. При переустановке базы данных пострадать не должны (но лучше все равно иметь бекап!).
* [**ps -ef | grep mysql**] – после установки СУБД можно проверить, запущена ли соответствующая служба. В выводе данной команды это можно увидеть.

После установки мускула к нему можно подключиться только с локальной машины и пользователем root (мы выполняем все действия через ssh – в терминале и можем подключиться к СУБД). Для подключений извне нам необходимо создать пользователя и дать ему какие-либо права на какой-либо объект СУБД или на весь сервер целиком.

Больше инфы по данному вопросу (пользователи/права доступа) можно найти тут:

<http://dev.mysql.com/doc/refman/5.0/en/create-user.html>

<http://dev.mysql.com/doc/refman/5.0/en/grant.html>

<http://dev.mysql.com/doc/refman/5.0/en/connection-access.html>

<http://dev.mysql.com/doc/refman/5.0/en/privileges-provided.html>

<http://dev.mysql.com/doc/refman/5.0/en/account-names.html>

* [**mysql -u root -p**] – подключиться к серверу мускула с именем пользователя root и с использованием пароля. Пароль будет запрошен через консоль (интерактивно).
* Для создания пользователя используем следующую команду (sql-оператор)

**create user ’<username>’@’<host>’ identified by ’<user\_password>’**

**<username>** - имя пользователя

**<host>** - хост, с которого разрешен доступ данному пользователю (правила наименования хостов см. в документации к мускулу).

**<user\_password>** - пароль для создаваемого пользователя.

Я создал такого пользователя:

**create user ’vinnypuhh’@’%’ identified by ’123456’;**

символ % в наименовании хоста указывает, что данному пользователю разрешен доступ с любого хоста (подробнее – см. доку к мускулу), точка-с-запятой означают конец команды.

* Теперь нам необходимо дать пользователю права доступа к нашему серверу. Мы дадим нашему пользователю vinnypuhh все привилегии – у нас же тестовый сервер, однако на реальном сервере необходимо четко разграничивать права доступа! Назначение прав осуществляется sql-оператором grant (синтаксис и описание оператора grant см. по ссылкам выше в доке к мускулу):

**grant all on \*.\* to ’vinnypuhh’@’%’**

* Если понадобится сменить пароль для пользователя в СУБД MySql, то необходимо использовать следующую команду:

**set password for '<username>'@'<host>' = password('<new password>');**

Вроде все. Наш сервер функционирует и принимает соединения – можно пользоваться.

### Установка веб-сервера apache2

Неплохие материалы на эту тему можно найти тут:

<http://www.linuxspace.org/archives/4285>

<https://help.ubuntu.com/10.04/serverguide/httpd.html>

* [**apache2 -v**] – данной командой можно проверить версию установленного сервера Апач или убедиться, что он не установлен в системе.
* [**sudo apt-get install apache2**] – установка веб-сервера Апач. При установке используются параметры по умолчанию. После успешной установки можно проверить работоспособность установленного сервера, зайдя веб-браузером по адресу **http://<server-ip>** или **http://<server-name>** (также можно использовать локальный адрес http://localhost, если браузер запущен на одной машине с веб сервером)
* [**sudo service apache2 start|stop|restart**] – старт/останов/перезапуск сервера апач. Имя сервиса (apache2) устанавливается по умолчанию. Также для перезапуска/старта/останова сервера можно использовать команду [**sudo /etc/init.d/apache2 restart**].
* [**ps -ef | grep apache2**] – с помощью данной команды можно убедиться, что сервис запущен (или нет).

По умолчанию document root (папка веб-сервера по умолчанию) находится здесь: /var/www, там же лежит HTML страничка по умолчанию, которую мы видим по адресу http://<server-name> или http://localhost.

#### Немного о конфигурировании apache2

**httpd.conf** – вся конфигурация и настройка сервера осуществляется через этот файл. Стоит отметить, что окрыв файл /etc/apache2/httpd.conf вы обнаружете, что он пуст, так и есть, настройки сервера лежат в /etc/apache2/apache2.conf (вот так ☺).

**.htaccess** – файл дополнительной конфигурации. Данный файл является подобием файла httpd.conf с той лишь разницей, что дейтсвует только на каталог, в котором он располагается (и на его дочерние каталоги). Возможность использования .htaccess в том или ином каталоге указывается в httpd.conf – директива AllowOverride.

**.htpasswd** – файл, содержащий пароли для доступа к ресурсу веб сервера Апач. Метод авторизации с использованием такого файла называется базовым (basic authentication).

### Установка PHP5

ПХП - неплохой и довольно удобный язык для разработки веб сайтов, от небольших домашних страничек до достаточно больших порталов. Давайте установим и его. Тем более, что он нам в дальнейшем понадобится.

* [**sudo apt-get install php5 libapache2-mod-php5**] – установка ПХП. Вместе с самим пхп (пакет php5) мы устанавливаем модуль выполнения пхп сценариев для веб сервера Апач (пакет libapache2-mod-php5).

Теперь проверим работоспособность пхп. Стандартная проверка: создаем тестовый php файл в document root веб сервера Апач (/var/www). Для создания этого файла воспользуемся командой [**sudo vi /var/www/info.php**], в создаваемом файле вводим и сохраняем текст:

*<?php*

*phpinfo();*

*?>*

Затем заходим по адресу **http://<server-name>/info.php** или **http://loclahost/info.php** и смотрим на результат. Там должны быть описаны основные (почти все) параметры пхп. Ищем раздел поддержки mysql (он так и назыввется), проверяем данные. Если такого раздела мы не нашли – необходимо установить модули поддержки mysql в php. Используем для этого следующую команду:

* [**sudo apt-get install php5-mysql php5-curl php5-gd php5-idn php-pear php5-imagick php5-imap php5-mcrypt php5-memcache php5-ming php5-ps php5-pspell php5-recode php5-snmp php5-sqlite php5-tidy php5-xmlrpc php5-xsl**] – данной командой устанавливаем модули поддержки Mysql в Php (рекомендую выполнить данную команду, даже если есть раздел mysql на страничке инфы о php – мы можем доустановить те модули, которых еще нет; в любом случае – хуже не будет! ☺).

Снова проверяем – все должно быть в порядке. Если же нет – где-то ошибка, ее надо найти и пофиксить ☺.

После установки всех модулей рекомендую перезагрузить сервер Апач.

### Работа с БД – phpmyadmin

Для визаулизации работы с СУБД MySql существует замечательный инструмент – PhpMyAdmin. Это написанный на пхп веб-интерфейс к СУБД mysql. Предлагаю им воспользоваться. Существуют и другие инструменты, но этот (на мой взгляд) - самый простой и удобный. Кстати – как раз сейчас нам и пригодится установленный ранее PHP.

* [**sudo apt-get install phpmyadmin**] – установка пхпадмина. Во время установки будут заданы пара вопросов, на которые надо ответить так:

*Web server to reconfigure automatically --->apache2*

*Configure database for phpmyadmin with dbconfig-common ---> no*

Результат успешной установки можно глянуть по адресу **http://<server-name>/phpmyadmin** или **http://localhost/phpmyadmin**. Логин и пароль - те же, что и для СУБД MySql.

### Установка Perl

* [**perl -v**] - с помощью данной команды можно проверить установленную версию Перла или то, что Перл не установлен.
* [**sudo apt-get install perl**] – установка Перла.

Перл очень неплохая штука, но местами очень сложная и непонятная. Например, если надо установить дополнительные модули (в частности, установка доп. модулей необходима будет при установке багзиллы – см. ниже). Есть несколько (или больше) способов сделать это. Некоторые способы я опишу тут.

#### Способ №1.

Можно воспользоваться утилитой apt-cache для поиска и, затем, утилитой apt-get для установки нужного модуля.

Например, для установки багзиллы нам надо установить дополнительный модуль perl-ldap (Net::LDAP) – работа по протоколу LDAP из перла. Этот модуль я нашел с помощью apt-cache:

**sudo apt-cache search perl ldap**

или

**sudo apt-cache search Net::LDAP**

или

**sudo apt-cache search perl Net::LDAP**

Если данная команда что-либо найдет, то выведет список найденных библиотек, удовлетворяющих критериям поиска, если же не найдет – никакого вывода (на экран) не будет.

Допустим, что мы что-то нашли, и среди выведенных библиотек видим такую:

**libnet-ldap-perl - <какие-то пояснения/описание на английском>**

ее и будем устанавливать:

**sudo apt-get install libnet-ldap-perl**

Все. Если нужный модуль никак не находится – пробуйте разные ключевые слова для поиска. Чаще модули находятся, если знать имена пакетов – как в данном случае – Net::LDAP.

Как выбрать нужную библиотеку из списка найденных? Я выбирал исходя из названия и описания, пару раз правда ошибся, но все равно – нужные мне модули установил. Как-то так.

#### Способ №2.

Воспользоваться  CPAN - Comprehensive Perl Archive Network – «всеобъемлющая сеть архивов перл». С этой штукой вроде все не очень сложно, хотя, говорят, есть подводные камни.

Если система имеет прямой выход в инет – все будет ок, если же выход в инет осуществляется через прокси-сервер, то могут быть траблы с настройкой.

Для начала надо установить все необходимые пакеты софта для CPAN:

**sudo apt-get install build-essential**

Запускаем

**sudo cpan**

Можно запускать и от имени обычного пользователя, но у меня при таком запуске во время установки ругалось на отсутствие прав – поэтому, думаю, лучше запустить от имени рута. Во время запуска нас будут спрашивать о параметрах – на первый вопрос надо ввести **yes** – все последующие шаги будут сконфигурированы автоматически.

Выполняем следующие команды:

**make install**

**install Bundle::CPAN**

Все необходимые шаги выполнены – мы можем устанавливать необходимые нам модули. Модули устанавливаются по именам пакетов ПО. Например так я установил модуль Email-MIME-Attachment-Stripper (так его имя указано в зависимостях багзиллы), который находится в пакете Email::MIME::Attachment::Stripper (наименование пакета также было указано в зависимостях багзиллы – см. ниже установку багзиллы):

**cpan prompt> install Email::MIME::Attachment::Stripper**

Таким образом можно установить дополнительные модули Перла.

Выбор способа установки зависит от вас, но некоторые модули я смог найти/установить только с помощью способа №2 (например, модуль Email-MIME-Attachment-Stripper), т.к. с помощью первого способа я не смог найти необходимую для инсталляции библиотеку.

### Установка системы трекинга ошибок (BugZilla, версия 4.x.x).

Хорошее описание, как это сделать, можно найти тут:

<http://vibhurishi.blogspot.ru/2009/05/howto-install-bugzilla-on-ubuntu-9.html>

<http://neil.motorcyclemarket.com/how-to-install-bugzilla-4-0-step-by-step-tutorial-that-works>

Этот раздел является переводом-компиляцией нескольких источников, с моими дополнениями и описнаием опыта практического применения.

Оригинальный ресурс, который поможет точно (если не помогло ничего):

<http://www.bugzilla.org>, <http://www.bugzilla.org/docs/4.2/en/html>.

Для установки данной системы необходимы некоторые предварительные условия:

* Perl 5.8.1 или выше (проверка версии командой [**perl -v**])
* MySql 5.0.15 или выше (проверка версии командой [**mysql -V**])
* Apache2 (можно использовать Apache 1.3.x или 2.x)

Если все эти условия выполнены мы можем приступить к установке Багзиллы.

* необходимо создать в системе пользователя bugzilla и дать ему сложный пароль:

**sudo useradd -d /home/bugzilla -m bugzilla**

**sudo passwd bugzilla**

Для тестовых задач я установил пароль **bugzilla** (как имя пользователя).

* теперь создаем БД bugzilla на сервере mysql и даем все привелегии доступа к ней пользователю bugzilla (которого также необходимо создать в СУБД):

**$ mysql -u root -p**

**mysql> create database bugzilla;**

**mysql> show databases;** - эта команда покажет все базы данных на сервере (найдем нашу)

**mysql> create user ’bugzilla’@’localhost’ identified by ’<bugzilla user password>’;**

**mysql> grant all privileges on bugzilla.\* to bugzilla@localhost;**

Этими командами мы подготовили нашу СУБД к установке Багзиллы.

* Проверяем установку Апача2 – командой [**apache2 -v**] и зайдя браузером по адресу http://<server\_name> (мы должны увидеть какую-нибудь страничку). Если апач2 не установлен – надо установить. Об этом см. выше в данном документе.
* Теперь нам необходимо скачать саму Багзиллу – <http://www.bugzilla.org> Нам необходима только стабильная версия (stable release). После скачивания, tar файл с багзиллой кладем в какой-нить каталог на сервере и распаковываем его (я скачал багзилу 4, файл называется bugzilla-4.2.4.tar)

**sudo tar -xvf bugzilla-4.2.4.tar**

переносим распакованный каталог с зиллой в /usr/local (tar файл и распакованный каталог у меня лежат в папке /tmp)

**sudo mv /tmp/bugzilla-4.2.4 /usr/local**

и создаем символическую ссылку

**sudo ln -s /usr/local/bugzilla-4.2.4 /var/www/bugzilla**

Наша багзилла должна лежать в каталоге, обрабатываемом Апачем – для этого и нужна ссылка. Наверное, можно просто положить багзиллу туда, но в одном из мануалов указан именно способ с символической ссылкой.

* Процесс инсталляции багзиллы основан на перловом скрипте checksetup.pl Этот скрипт проверяет наличие и версии всех необходимых модулей Перла. Нам надо пройти проверку.

Проверяем инсталляцию нужных модулей

**/usr/local/bugzilla-4.2.4 $ sudo ./checksetup.pl --check-modules**

Если установлены не все модули (это можно будет понять из сообщений на экране), то необходимо установить недостающие – для этого загляни в раздел «Установка Perl» выше в данном документе. Процесс установки всех недостающих модулей может быть долгим, скучным и местами не очень простым – но установка БагЗиллы того стоит ☺. Стоит обратить внимание, что не все модули являются обязательными для Зиллы – некоторые модули опциональны, если они отсутствуют, то не будут работать некоторые «плюшки», без которых можно обойтись. Главное – установить все обязательные модули, без этого продолжать установку бессмысленно.

Проверить, что установлены все необходимые модули – запустить скрипт checksetup.pl еще раз – см. команду выше.

Вот небольшая помощь для инсталляции модулей perl для багзиллы:

* Основные модули:

**apache2 mysql-server libappconfig-perl libdate-calc-perl libtemplate-perl libmime-perl build-essential libdatetime-timezone-perl libdatetime-perl libemail-send-perl libemail-mime-perl libemail-mime-modifier-perl libdbi-perl libdbd-mysql-perl libcgi-pm-perl libmath-random-isaac-perl**

* Дополнительные модули:

**apache2-mpm-prefork libapache2-mod-perl2 libapache2-mod-perl2-dev libchart-perl libxml-perl libxml-twig-perl perlmagick libgd-graph-perl libtemplate-plugin-gd-perl libsoap-lite-perl libhtml-scrubber-perl libjson-rpc-perl libdaemon-generic-perl libtheschwartz-perl libtest-taint-perl libauthen-radius-perl libfile-slurp-perl libencode-detect-perl libmodule-build-perl**

Установка этих пакетов с помощью apt-get должна помочь – это бОльшая часть модулей, необходимых для установки и корректной работы Багзиллы.

Еще немного информации по поводу инсталляции дополнительных модулей:

<http://www.temme.net/sander/2007/05/12/bugzilla-3-has-landed/>

* После установки всех необходимых модулей Перла, запускаем установочный скрипт checksetup.pl еще раз, но уже без параметров:

**/usr/local/bugzilla-4.2.4 $ sudo ./checksetup.pl**

Когда данный скрипт отработает, в каталоге багзиллы будет создан конфиг-файл нашего сервера (багзиллы) – файл **localconfig**. Этот файл надо открыть текстовым редактором и отредактировать:

$webservergroup = ‘www-data’;  
$db\_pass = ‘your password for bugs@localhost’;

Также необходимо проверить другие настройки – имя созданной для багзиллы БД, имя пользователя, порт СУБД и т.п. – смотрите в localconfig внимательно – все настройки достаточно очевидны.

* После редактирования и сохранения файла localconfig надо снова запустить установочный скрипт без параметров:

**/usr/local/bugzilla-4.2.4 $ sudo ./checksetup.pl**

Если все настройки в файле localconfig были указаны верно, то в БД для багзиллы будут созданы все необходимые таблицы, а нас спросят о email адресе для админа системы (админа Багзиллы!) и о пароле для этого админа. ***Важно! Мейл адрес админа является его логином в систему!***

* Мы почти закончили! Еще чуть-чуть! Теперь нам надо настроить Апач для работы с багзиллой. Открывем конфиг апача (см. выше про настройку Апача, у меня конфиг Апача - это файл /etc/apache2/apache2.conf) и добавляем в него следующее (в самый конец файла):

**<Directory “/var/www/bugzilla”>  
AddHandler cgi-script cgi  
DirectoryIndex index.cgi  
Options +Indexes +ExecCGI -MultiViews +SymLinksIfOwnerMatch +FollowSymLinks  
AllowOverride None  
Order allow,deny  
Allow from all  
</Directory>**

Перезапускаем Апач:

**sudo /etc/init.d/apache2 restart**

Если мы все сделали правильно, то по адресу **http://<server-name>/bugzilla** или **http://<localhost>/bugzilla**  - мы увидим нашу Багзиллу. Ура.

### Установка wiki-движка (MediaWiki)

Хорошее и простое руководство, как установить и настроить Media Wiki находится тут:

<https://help.ubuntu.com/community/MediaWiki>

Нижеследующие шаги я выполнил по данному руководству – и все получилось! ☺

* устанавливаем вики из онлайн репозитория:

**sudo apt-get install mediawiki**

* опционально можно установить дополнительные модули (mediawiki-math весит около 400 метров – надо ли оно нам?):

**sudo apt-get install imagemagick mediawiki-math php5-gd**

* теперь надо немного сконфигурировать нашу вики – открываем файл /etc/mediawiki/apache.conf в текстовом редакторе и удаляем символ ‘#’ из начала третьей сверху строки (раскомментируем строку), она должна выглядеть вот так:

*Alias /mediawiki /var/lib/mediawiki*

Сохраняем отредактированный файл и перезапускаем Апач:

**sudo service apache2 restart**

Теперь наша вики должна быть доступна по адресу **http://<server-name>/mediawiki** или **http://localhost/mediawiki**. Надо зайти и проверить! Но не надо торопиться с настройками – создадим БД для нашей Вики ☺.

* Важный этап – создание БД на нашем сервере mysql для медиавики:

**$ mysql -u root -p**

**mysql> create database wikidb;**

**mysql> show databases;** - эта команда покажет все базы данных на сервере (найдем нашу)

**mysql> create user ’wikiuser’@’localhost’ identified by ’<wiki user password>’;**

**mysql> grant all privileges on wikidb.\* to wikiuser@localhost;**

* Завершаем настройку: заходим на начальную страницу вики (см. выше) - там будут инструкции по первоначальной настройке – надо следовать им, там все просто (настройка пользователя-админа, настройка доступа к СУБД и другие очевидные настройки). **После заполнения формы настроек и появления странички с сообщениями об успешной установке – обрати внимание на необходимость переноса файла настроек вики в другое место – это сообщение будет внизу страницы!**

Дополнительную инфу можно найти в указанной выше статье: информация по поддержке email-уведомлений в вики и по кастомизации вики – рекомендую с ней ознакомиться.

Мы закончили установку и первоначальную настройку MediaWiki-движка. Все.

### Установка java (JDK).

Теперь на нашем сервере не хватает главного – java. Нам необходимо установить не только (и не столько) **jre**, сколько **jdk**. Для чего? Мы планируем установить на наш сервер контейнеры Java EE – Tomcat, GlassFish, другие, а им для работы необходим именно пакет JDK. Также в состав пакета JDK входит компилятор Java.

Начнем. Для начала необходимо убедиться, что Жавы действительно нет на нашем сервере. В этом нам помогут команды **java -version** и **javac -version**. Если Жавы нет – сервер нам об этом сообщит, если же есть – также сообщит ☺

Мы рассмотрим установку двух версий Java SDK:

* Open JDK – свободно распространяемый пакет, есть в онлайн репозиториях Ubuntu и легко ставится с помощью apt-get. Однако данный пакет имеет некоторые проблемы с производительностью и стабильностью – друзья рассказывали, сам не сталкивался. Я не очень хочу его устанавливать ☺.
* Oracle JDK – классическая реализация спецификации java от производителя технологии. Говорят, тоже имеет проблемы на Убунту, но меньше, чем свободный аналог. Предлагаю воспользоваться именно этой реализацией пакета.

#### Open JDK

Этот пакет достаточно легко ставится из онлайн репозиториев следующими командами:

* [**sudo apt-get install openjdk-7-jdk**], [**sudo apt-get install openjdk-7-jre**] – установка jdk и jre 7, вместо 7 можно поставить 6-ю версию. Еще можно установить пакет с документацией.

#### Oracle JDK

**Способ №1. Установка вручную (неудобно, не рекомендую).**

С данным пакетом все несколько сложнее. Его нет в онлайн репозиториях, и нам придется устанавливать его вручную. Вот тут прекрасная статья, как это выполнить:

<http://habrahabr.ru/post/143113>

* для начала, надо скачать дистрибутив JDK для Linux с сайта Oracle – нам нужен **rpm**-пакет установки. Если нет отдельного rpm-пакета – тогда скачиваем jdk-xxxx-rpm.bin (это исполняемый zip-архив, его надо распаковать и достать оттуда rpm-пакет). Копируем установочный пакет rpm на наш Убунту сервер (помнишь WinSCP?). Я скопировал в каталог /tmp
* [**sudo apt-get install alien**] - установка пакета alien на наш сервер. Alien – это конвертер пакетов, который может перепаковать rpm-пакет в формат deb-пакета (пакет для Debian – Убунту создана на основе Debian).
* [**sudo alien jdk-6u38-linux-i586.rpm --scripts**] - натравливаем установленного «чужого» на наш rpm-пакет с Oracle JDK (в команде вместо указанного имени надо поставить имя своего скачанного пакета). Перепаковка может занять какое-то время, у меня заняла около 4-5 минут. Результатом работы Чужого будет сгенерированный в той же директории пакет .deb (native-пакет для Debian).
* [**sudo dpkg -i jdk\_1.6.038\_l\_i386.deb**] – установка перекодированного пакета. Это также занимает некоторое время. После установки обязательно надо проверить правильность – запустить команды **java -version** и **javac -version**.

Куда же установилась наша Жаба (каталог)? На моем сервере установка произошла в каталог **/usr/java/jdk<version>** , также в каталоге /usr/java были созданы две символических ссылки:

**latest -> /usr/java/jdk<version>**

**default -> /usr/java/latest**

После выполнения вышеуказанных действий на нашем Убунту-сервере установлен пакет JDK.

**Способ №2. Достаточно удобно (+ обновления).**

Можно автоматически установить JDK с помощью сторонних скриптов. Подробно см. <http://help.ubuntu.ru/wiki/java>

1. Добавляем репозиторий и обновляемся:

***sudo add-apt-repository ppa:webupd8team/java***

***sudo apt-get update***

1. Устанавливаем нужную версию java:

***sudo apt-get install oracle-java8-installer***

В процессе инсталляции необходимо будет подтвердить (вручную) лицензию Оракла.

1. Кроме того, данный репозиторий изменяет зависимости пакетов, использующих Java, так, что становится возможным безболезненное удаление OpenJDK из системы.
2. После установки можно проверить установленную версию:

**java -version**

**javac -version**

1. Возможна установка нескольких версий Джава на машину:

***sudo apt-get install oracle-java7-installer***

Переключение между используемыми версиями (с обновлением ссылок):

***sudo update-java-alternatives -s java-7-oracle***

***sudo update-java-alternatives -s java-8-oracle***

1. Для автоматической установки переменных среды:

***sudo apt-get install oracle-java7-set-default***

### Установка Apache Ant

Данный инструмент для сборки можно поставить автоматически из репозитория, однако нужную специфическую версию лучше поставить самостоятельно вручную. Для этого:

1. Скачиваем zip/gzip архив и распаковываем его в нужный каталог:

***wget*** [***http://apache.heikorichter.name/ant/binaries/apache-ant-1.8.4-bin.tar.gz***](http://apache.heikorichter.name/ant/binaries/apache-ant-1.8.4-bin.tar.gz)

*(the last version for 02.10.2016 - http://apache-mirror.rbc.ru/pub/apache//ant/binaries/apache-ant-1.9.7-bin.tar.gz)*

***tar xvfz apache-ant-1.8.4-bin.tar.gz***

***sudo mv apache-ant-1.8.4 /opt***

1. Редактируем файл ~/.profile и добавляем следующие строки:

***ANT\_HOME="/opt/apache-ant-1.8.4"***

***PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:$ANT\_HOME/bin"***

1. Загружаем изменения из файла профиля:

***source ~/.profile***

Можно просто перезагрузиться.

1. Необходимо зайти в каталог [ANT\_HOME]\bin и убедиться, что файл ant имеет все права для запуска (атрибут x), если нет, то добавить этот атрибут командой:

***sudo chmod 755 ant***

1. Запускаем Ант (для проверки версии) командой

***ant –version***

данная команда должна показать используемую версию Анта.

### Установка Apache Maven

http://apache-mirror.rbc.ru/pub/apache/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz

### Установка Ruby (+RVM (Ruby Version Manager), +Rails).

1. Для того, чтобы не возникало путаницы с версиями Руби, для начала удалим имеющуюся установку Руби (внимание, возможно имеющаяся установка все-таки нужна – подумайте перед удалением!):

***sudo apt-get remove ruby***

RVM – Ruby Version Manager, программа управления версиями Руби (подробнее см. [http://rvm.io](http://rvm.io/)). Эта программа (набор скриптов) позволяет иметь на ПК более одной версии Руби и переключаться между ними. Более подробно можно посмотреть тут:

<http://habrahabr.ru/post/120504/>

<http://www.8host.com/blog/ustanovka-ruby-on-rails-s-pomoshhyu-rvm-v-ubuntu-14-04/>

1. Непосредственно установка связки Ruby+RVM+Rails. Для начала выполнить команду:

***gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3***

для импорта ключа шифрования, а затем:

***\curl -sSL https://get.rvm.io | sudo bash -s stable --rails***

непосредственно для установки rvm+rails+ruby. Команду запускать не от рута. Символ \ необходимо обязательно указывать. Команда установит Руби для всех пользователей системы, входящих в группу rvm (группа будет создана в процессе установки). Папка установки /usr/local/rvm. После установки обязательно необходимо выполнить:

***source /usr/local/rvm/scripts/rvm***

1. Проверить установленные версии пакетов можно командами:

***ruby -v***

***rails -v***

***rvm -v***

### Установка Redmine (для версии 3.1.1)

Описание системы, тех. информацию, разные FAQ/HowTo можно посмотреть на сайте redmine.org

В процессе установки Редмайн будет установлен в каталог /var/www/vhosts/redmine. Выбрать для установки можно любой каталог.

Хорошая статья по установке тут:

<https://niccomatik.de/2015/03/installing-redmine-3-0-on-clean-ubuntu-14-04>

Непосредственно шаги установки:

1. Рекомендуется для начала обновить систему

***sudo apt-get update && sudo apt-get upgrade && sudo apt-get dist-upgrade***

1. Установить Ruby (см. раздел об установке Руби, рекомендуется одна версия для Редмайна)
2. После установки Руби, добавить текущего пользователя и пользователя www-data в группу rvm:

***sudo usermod –aG rvm <current user>***

***sudo usermod –aG rvm www-data***

После этого рекомендуется перелогиниться.

1. Установить (если еще не установлены) Apache2/MySql:

***sudo apt-get install apache2 mysql-server mysql-client***

Для Апача не следует заранее из репозитория ставить мод Passenger (мод для запуска Руби приложений по Апачем), т.к. он тянет за собой свою (часто старую) версию Руби.

1. Доустановить в систему нужные пакеты:

* libmysqlclient-dev - пакет клиента для mysql, необходим для сборки/работы некоторых модулей Редмайна на Руби
* libmagickwand-dev - пакет для установки gem'a rmagic – работа с графикой в Редмайн, экспорт PDF/PNG, пакет тянет за собой много зависимостей
* libcurl4-openssl-dev, apache2-threaded-dev - пакеты необходимые для интеграции мода passenger под apache2 (их можно установить после установки passengera через gem)

1. Настроить СУБД MySql (создать БД и пользователя, дать пользователю права):

***CREATE DATABASE redmine CHARACTER SET utf8;***

***CREATE USER 'redmine'@'localhost' IDENTIFIED BY 'my\_password';***

***GRANT ALL PRIVILEGES ON redmine.\* TO 'redmine'@'localhost';***

1. Выгрузить из репозитория Subversion или скачать архивом необходимую версию Redmine. Расположить/распаковать ее в нужном каталоге, у меня это каталог /var/www/vhosts/redmine.
2. Все дальнейшие шаги подразумевают, что рабочий каталог – тот каталог, куда установлен Редмайн. В нашем случае: /var/www/vhosts/redmine. Ссылки на файлы и запуск команд подразумеваются относительно данного каталога.

Настроить конфиг соединения Редмайна с БД – скопировать файл config/database.yml.example в config/database.yml (пути относительно каталоги установки Редмайна). Настройка для MySql и Руби версии 1.9 и выше:

***production:***

***adapter: mysql2***

***database: redmine***

***host: localhost***

***username: redmine***

***password: my\_password***

- это прописать непосредственно в файле

1. Установить bundler и необходимые зависимости для Редмайна (команду надо выполнять в каталоге установки Редмайна!):

***gem install bundler***

***bundle install --without development test***

1. Сгенерировать секретный ключ для пользовательских сессий (для Редмайн старше 2.х):

***bundle exec rake generate\_secret\_token***

1. Создание структуры и объектов в БД:

***RAILS\_ENV=production bundle exec rake db:migrate***

1. Загрузка данных по умолчанию (начальных данных):

***RAILS\_ENV=production bundle exec rake redmine:load\_default\_data***

В процессе будет задан вопрос про используемый язык, надо будет указать двухсимвольный код языка (напр. ru или en).

1. Для проверки правильности установки Редмайна можно запустить встроенный сервер webrick (он должен стартовать без ошибок, завершение работы сервера – Ctrl-C):

***bundle exec ruby bin/rails server webrick –e production***

1. Для интеграции Redmine с Апачем, необходимо установить мод passenger для Апача:

***gem install passenger***

Не рекомендуется устанавливать пассенджер из репозитория, т.к. он тащит за собой Руби (свою версию) и некоторые свои зависимости, а под них мы не устанавливали gem'ы.

1. Непосредственно установим мод passenger под apache2:

***sudo -i***

***passenger-install-apache2-module***

установка будет проходить в текстовом интерактивном режиме, надо будет ответить на несколько вопросов. Установка займет какое-то время, т.к. будет собрана непосредственно сама библиотека (.so файл). В завершении установки будет предложено настроить конфиг апача для использования пассенджера (см. далее).

1. В файл апача /etc/apache2/mods-avaiable/passendger.load необходимо записать команду загрузки мода:

*LoadModule passenger\_module /usr/local/rvm/gems/ruby-2.2.1/gems/passenger-5.0.20/buildout/apache2/mod\_passenger.so*

Путь может варьироваться в зависимости от установки.

1. В файл апача /etc/apache2/mods-avaiable/passendger.conf необходимо записать конфиг модуля пассенджер:

*<IfModule mod\_passenger.c>*

*PassengerRoot /usr/local/rvm/gems/ruby-2.2.1/gems/passenger-5.0.20*

*PassengerDefaultRuby /usr/local/rvm/gems/ruby-2.2.1/wrappers/ruby*

*</IfModule>*

Пути могут варьироваться в зависимости от установки.

1. Разрешаем установленный мод под апачем:

***a2enmod passenger***

***sudo service apache2 restart***

1. Конфигурация хоста под апачем. Необходимо создать файл конфигурации для хоста Редмайн под апачем в каталоге /etc/apache2/sites/available, например с именем redmine.conf (расширение .conf обязательно) с примерно следующим содержимым:

*<VirtualHost \*:80>*

*# ServerName redmine.example.com - опционально*

*#ServerAlias www.redmine.example.com - опционально*

*# !!! Be sure to point DocumentRoot to 'public'!*

*DocumentRoot /var/www/vhosts/redmine/public*

*<Directory /var/www/vhosts/redmine/public>*

*# This relaxes Apache security settings.*

*AllowOverride all*

*# MultiViews must be turned off.*

*Options -MultiViews*

*Require all granted*

*</Directory>*

*</VirtualHost>*

Для активации нового сайта выполнить команду:

***a2ensite redmine***

Если используется порт по умолчанию (80), надо отключить сайт по умолчанию под апачем (команда a2dissite). Если используется другой порт (напр. 3000), то необходимо прописать его прослушивание апачем – в файл /etc/apache2/ports.conf необходимо дописать команду:

*Listen 3000*

1. В каталоге Редмайна необходимо переименовать файл /var/www/vhosts/redmine/public/dispatch.fcgi.example в dispatch.fcgi.
2. Перезапустить апач и проверить работоспособность установленного редмайна.
3. Интеграция Редмайн с почтой. Для интеграции необходимо добавить соответствующие параметры в конфиг файл Редмайна config/configuration.yml (см. описание на оф. сайте Редмайна или пример в файле config/configuration.yml.example).

### Установка Tomcat 6

Tomcat - один из самых простых и легковесных (да еще и бесплатный) j2ee контейнер. Для простых задач на java enterprise – подойдет.

Кратко, но емко об установке Томката можно прочесть здесь:

<http://www.howtogeek.com/howto/linux/installing-tomcat-6-on-ubuntu/>

Далее описаны мои действия (завершившиеся успехом ☺).

* для начала, необходимо скачать Томкат с сайта **tomcat.apache.org** – я взял версию Tomcat 6 (также пробовал и версию 7 – все работает). Скачиваем .zip или tar.gz. Скачанный архив кладем на сервер в каталог /tmp (пользуемся для этого WinSCP). Переходим в терминал и далее устанавливаем Кошака на сервере.
* [**sudo tar xvzf apache-tomcat-6.0.36.tar.gz**] – распаковываем архив с Кошаком в текущий каталог (у меня текущий - /tmp). Вместо указанного имени архива надо указать имя своего архива (если отличается). Далее по тексту – если упоминается конкретный каталог Кошака – подставляешь свой.
* [**sudo mv apache-tomcat-6.0.36 /usr/local/tomcat**] – переносим распакованный сервер Tomcat в нужный каталог (каталог может быть любым).
* Для корректной работы Кошаку необходима установленная переменная окружения **JAVA\_HOME**. Ее можно установить для конкретного пользователя или для системы в целом. Я предпочитаю второй вариант, поэтому добавим установку этой переменной в файл **/etc/environment** (воспользуемся редактором vi).

Подробнее о переменных окружения см. выше – в разделе о них. О том, где найти установленный пакет JDK – см. выше в разделе об установке java. После установки переменной окружения – рекомендую перезагрузить машину (или сделать logout/login). Напомню – посмотреть все переменные – **printenv**, одну конкретную – **printenv <variable>** или **echo $<variable>**.

* Теперь надо запустить наш Томкат – проверить установку. Для этого заходим в каталог с установленным Томкатом (у меня это каталог /usr/local/tomcat), переходим в каталог bin и выполняем команду [**sudo ./startup.sh**]. Мы запустили наш сервер. Проверить, что он работает можно по адресу **http://<server-name>:8080** или **http://<localhost>:8080** (8080 – порт по умолчанию для Tomcat). Если мы увидели стартовую страничку Томката – все ок, если же нет – проблема.
* Наш Томкат запускается только вручную, т.е. при перезагрузке сервера он сам не стартует. Надо это исправить. Для этого надо создать скрипт для автозапуска/перезапуска Томката. Создаем файл скрипта командой [**sudo vi /etc/init.d/tomcat**]. Вот необходимое нам содержимое файла (можно поменять все настройки на свои):

*# Tomcat auto-start  
#  
# description: Auto-starts tomcat  
# processname: tomcat  
# pidfile: /var/run/tomcat.pid*

*export JAVA\_HOME=/usr/lib/jvm/java-6-sun*

*case $1 in  
start)  
        sh /usr/local/tomcat/bin/startup.sh  
        ;;   
stop)     
        sh /usr/local/tomcat/bin/shutdown.sh  
        ;;   
restart)  
        sh /usr/local/tomcat/bin/shutdown.sh  
        sh /usr/local/tomcat/bin/startup.sh  
        ;;   
esac      
exit 0*

Строчка *export …* может и не нужна – мы ведь уже установили глобальную системную переменную окружения JAVA\_HOME. По крайней мере, мой скрипт заработал без этой строчки. После создания файла скрипта нам необходимо сделать его запускаемым, для этого выполним команду

**sudo chmod 755 /etc/init.d/tomcat**

Вместо chmod 755 можно указать ключ +x (добавление к атрибутам файла атрибута «запускаемый»):

**sudo chmod +x /etc/init.d/tomcat**

Последний шаг для добавления Томката в автозапуск системы – связать символическими ссылками данный скрипт с папками автозапуска. Для этого выполним команды:

**sudo ln -s /etc/init.d/tomcat /etc/rc1.d/K99tomcat**

**sudo ln -s /etc/init.d/tomcat /etc/rc2.d/S99tomcat**

Однако есть более правильный (простой) способ для этого – воспользуемся утилитой update-rc.d:

**sudo update-rc.d tomcat defaults**

Теперь запуск/останов/перезапуск сервера может быть выполнен командой:

**sudo service tomcat start|stop|restart**

Все! Перезагружаем наш сервер Ubuntu и проверяем, что Томкат стартовал автоматом. После выполнения всех вышеописанных шагов на нашем сервере поселился Кошак ☺.

### Установка GlassFish

GlassFish – полноценная и эталонная реализация контейнера j2ee от производителя технологии - компании Oracle. Данный контейнер бесплатен и свободно распространяется (подробнее см. лицензию).

#### Немного о сервере GlassFish

Где и что, домены, каталоги и т.п.

#### Непосредственно установка

Неплохие ресурсы по установке/обслуживанию/работе с Glassfish:

<http://ivanbochko.com/rabochee-okruzhenie/ustanovka-glassfish-na-ubuntu-server>

<http://www.nabisoft.com/tutorials/glassfish/installing-glassfish-311-on-ubuntu>

Ниже описаны мои действия по установке.

* Скачиваем zip-архив с официального сайта (<http://glassfish.java.net>) и копируем этот архив на наш Ubuntu сервер в каталог /tmp.
* распаковываем на сервере этот архив командой unzip (tckb гтяшз не установлен – надо установить: **sudo apt-get install unzip**):

**unzip glassfish-3.1.2.zip**

архив будет распакован в текущий каталог.

* копируем распакованный сервер в каталог /usr/local, затем выставляем скопированному каталогу необходимые атрибуты:

**sudo cp -fR glassfish3/ /usr/local**

**sudo chmod 0777 /usr/local/glassfish3 -fR**

если еще необходимо сменить владельца, то поможет команда

**sudo chown www-data:www-data /usr/local/glassfish3 -fR**

здесь мы меняем владельца на www-data, можно установить другого.

* по умолчанию Глассфиш использует порт 8080 для веб-соединений, этот же порт использует и Томкат. Поэтому, если Томкат уже установлен – возникнет конфликт портов (наш Глаз не стартует просто). Необходимо изменить файл конфига GlassFish:

**sudo vi /usr/local/glassfish3/glassfish/domains/domain1/config/domain.xml**

и изменить этот порт. Я просто добавил ко всем номерам портов единицу слева: 8080 -> 18080.

* для старта и останова сервера можно использовать команды:

**sudo ./usr/local/glassfish3/bin/asadmin start-domain domain1**

**sudo ./usr/local/glassfish3/bin/asadmin stop-domain domain1**

* остался только один вопрос: нам надо запускать ГлазФиш при старте нашего сервера. Для этого создадим старт-скрипт:

**sudo vi /etc/init.d/glassfish3**

Содержимое скрипта:

*#!/bin/sh*

*case $1 in*

*start)*

*start-stop-daemon -c root --start --exec /usr/local/glassfish3/bin/asadmin start-domain domain1*

*;;*

*stop)*

*start-stop-daemon -c root --start --exec /usr/local/glassfish3/bin/asadmin stop-domain domain1*

*;;*

*restart)*

*start-stop-daemon -c root --start --exec /usr/local/glassfish3/bin/asadmin stop-domain domain1*

*start-stop-daemon -c root --start --exec /usr/local/glassfish3/bin/asadmin start-domain domain1*

*;;*

*\*)*

*echo "Usage: /etc/init.d/glassfish3 {start|stop|restart}"*

*exit 1*

*;;*

*esac*

*exit 0*

Команда **start-stop-daemon** запускает/останавливает процесс от имени указанного пользователя. Для созданного скрипта надо установить права на запуск:

**sudo chmod a+x /etc/init.d/glassfish3**

также надо добавить этот скрипт в автозагрузку:

**sudo update-rc.d glassfish3 defaults**

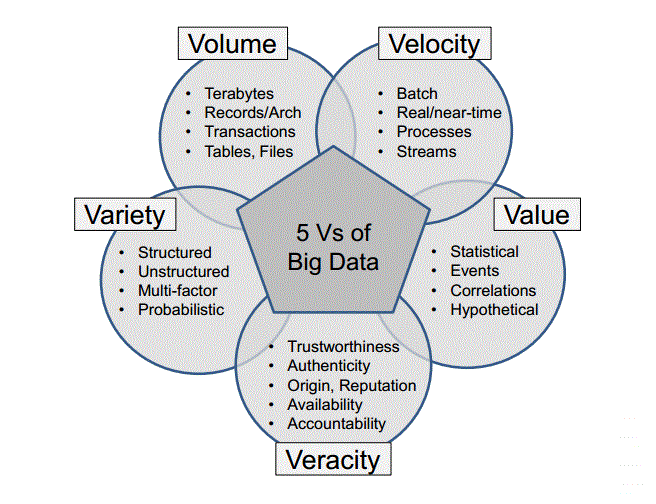
Теперь запуск/останов/перезапуск сервера может быть выполнен командой:

**sudo service glassfish3 start|stop|restart**

После выполнения вышеуказанных шагов будем считать, что на нашем сервере успешно установлен GlassFish 3.

# Часть VI. Big Data.

What is BigData? Usually BigData is 5Vs (sometimes you may hear N Vs):



## Глава 38. Hadoop.

Hadoop core components:

* **Hadoop Commons.** Libraries and utilities used by other Hadoop modules.
* **Hadoop Clients.** Libraries and utilities used to access Hadoop’s components.
* **HDFS.** Scalable system that stores data across multiple machines without prior organization.
* **YARN.** Resource management framework for scheduling and handling resource requests from distributed apps.
* **MapReduce.** Software programming model for processing large sets of data in parallel in fact, distributed application on top of YARN.

Hadoop frameworks:

* Universal (MapReduce, HDFS, Kudu, RDD, Tez)
* Pipelining (Pig, Cascading)
* SQL Like (Hive, Impala, Spark SQL)
* Graph Processing (Giraph, GraphX)
* Machine Learning m(Mahout, MLib)
* Stream Processing (Spark Streaming, Storm)

# Часть VII. Management stuff ☺

## Глава 39. О переговорах.

Из книги Гэвина Кеннеди «Договориться можно обо всем».

1. Существует четыре типа переговорщиков. Ослы пребывают в блаженном неведении относительно существующих возможностей. Овцы считают, что получить что-нибудь — уже хорошо. Лисы понимают, что происходит и твердо убеждены что заслуживают того, чего добиваются. Совы демонстрируют, что у них хватает мудрости оценить долгосрочные выгоды.
2. Ничто не может быть более скучным, чем предсказуемая «безопасность» презентаций идей, которая никого не оскорбляет просто потому, что никого не провоцирует.
3. Все взрослые прекрасно знают, как надо воспитывать детей, — но только тогда, когда речь идет о чужих детях, а не о своих собственных.
4. Сила детей-переговорщиков заключается в ассиметрии ценностей — их и ваших. Они быстро подмечают, что важнее всего для вас — и, угрожая отнять у вас это, добиваются того, что нужно им.
5. Самое худшее, что вы можете сделать в процессе переговоров, это принять первое предложение. Принятое с ходу предложение подрывает веру в совершенную сделку и в самих себя.
6. Люди великолепно владеют техникой выражения недовольства. При этом люди забывают о главном, а именно: каким образом исправить ситуацию, вызвавшую их недовольство. Ваша задача состоит не в том, что возмущаться недоработками другой стороны, а позаботиться о собственных интересах.
7. Предоставляя инициативу другому человеку, можете быть уверены: он будет иметь в виду только собственные интересы. Атакуйте человека — он будет защищаться. Атакуйте его яростно — и защита превратится в контратаку.
8. Никогда не смягчайте предлагаемые вами условия до того, как вы получили предложения от другой стороны.
9. Заключая соглашение, вам нужно задавать массу вопросов, каждый из которых начинается с двух простых слов: «Что если?».
10. Добровольные уступки, сделанные одной стороной, не смягчают другую сторону — они делают её позицию ещё более жёсткой.
11. Метод тетушки Хаббард — вы действительно хотите купить товар, но у вас нет ресурсов, чтобы принять первоначальную цену.
12. Фраза «торг уместен» — ошибка. Это сигнал покупателю, что вы готовы продать товар дешевле еще до того, как покупатель откроет рот.
13. Жесткость в переговорах (определяемая как умение предъявлять жесткие требования и твердо их придерживаться) обычно выигрывает у мягкости.
14. Переговор — это контакты, в которых обе стороны имеют право наложить вето на конечный результат. Переговоры предполагают добровольное согласие сторон на принятое решение.
15. Самое полезное слово в словаре переговорщика — «Если…». Именно тогда, когда вы воспринимаете переговоры как процесс обмена, вы понимаете важность того, чтобы никогда ничего не уступать, не получая ничего взамен.
16. Худшее в ваших уступках — это то, что они ослабляют вашу позицию в гораздо большей степени, чем сам размер уступки.
17. Каким бы ни было поведение другой стороны — тяжелым, компромиссным или любым другим, оно не должно влиять на конечный результат.
18. Смысл грубого поведения, хамских «наездов», забастовок, террористических актов заключается в том, чтобы причинить ущерб противной стороне. Ущерб, достаточный для того, чтобы ослабить её желание настаивать на первоначально избранном ею варианте разрешения ситуации.
19. Тактика «наказа» — действовать в переговорах от лица начальника, брата, друга, и не соглашаться на уступки, так как переговоры якобы вы ведете не от своего лица.
20. Чем больше физическая дистанция между вами и оппонентом, тем легче сказать ему «нет». Поэтому, если вам надо сказать «нет», делайте это по телефону — или в письме — и избегайте личных контактов.
21. Твердо установленных цен нет в природе — в этом заключена сама суть рыночной экономики.
22. Не снижайте цену — меняйте пакет предложения. Для этого пакета предложений — эта цена, для другой цены — другой пакет.
23. Если вы профессионально делаете то, что делаете — это лучший способ «запугать» оппонента.
24. Требование захватчика самолёта выпустить из тюрем содержащихся там террористов — это угроза вынуждения. Принятие закона о обязательном пожизненном заключении для захватчиков — это угроза сдерживания.
25. Никогда не рассказывайте о ваших нуждах — расскажите лучше о выгодах, которые появятся, если вам дадут то, в чём вы нуждаетесь.