**Jump学习手册**

交通银行软件中心

**文档修改历史记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 版本状态 | 创建人 | 修改日期 | 备注 |
|  | V0.0.0.1 | 刘定朝 | 2018.01.13 | 创建文档 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 目 录

[目 录 III](#_Toc17385403)

[图目录 XXV](#_Toc17385404)

[表目录 XXVI](#_Toc17385405)

[第1章 Webservice 27](#_Toc17385406)

[1.1 什么是Webservice 27](#_Toc17385407)

[1.2 Webservice平台技术 27](#_Toc17385408)

[1.2.1 XSD+XML 27](#_Toc17385409)

[1.2.1.1 XML实体引用 27](#_Toc17385410)

[1.2.1.2 XSD实例 27](#_Toc17385411)

[第2章 HTTP协议 28](#_Toc17385412)

[2.1 URI/URL 28](#_Toc17385413)

[2.2 Http之请求消息Request 28](#_Toc17385414)

[2.3 Http之响应消息Response 29](#_Toc17385415)

[2.4 Http之状态码 29](#_Toc17385416)

[2.5 Http请求方法 29](#_Toc17385417)

[2.6 Http工作原理 30](#_Toc17385418)

[2.7 Post和Get差别 30](#_Toc17385419)

[第3章 XML 32](#_Toc17385420)

[3.1 XML定义 32](#_Toc17385421)

[3.2 XML构成 32](#_Toc17385422)

[3.2.1 注释 32](#_Toc17385423)

[3.2.2 区分大小写 32](#_Toc17385424)

[3.2.3 标记 32](#_Toc17385425)

[3.2.4 属性 32](#_Toc17385426)

[3.2.5 实体引用 32](#_Toc17385427)

[3.2.6 CDATA 32](#_Toc17385428)

[3.2.7 PCDATA 33](#_Toc17385429)

[3.2.8 处理指令 33](#_Toc17385430)

[3.3 应用 33](#_Toc17385431)

[3.3.1 XSD-XML SCHEMA 33](#_Toc17385432)

[3.3.1.1 xsi:schemaLocation属性 33](#_Toc17385433)

[3.3.1.2 xsi:noNamespaceSchemaLocation属性 33](#_Toc17385434)

[第4章 XSD 33](#_Toc17385435)

[4.1 XSD定义 33](#_Toc17385436)

[4.2 元素类型 33](#_Toc17385437)

[4.2.1 简易元素 33](#_Toc17385438)

[4.2.2 数据类型限定 34](#_Toc17385439)

[4.2.3 XSD指示器 34](#_Toc17385440)

[4.3 示例 35](#_Toc17385441)

[第5章 WSDL 35](#_Toc17385442)

[第6章 HTML 35](#_Toc17385443)

[6.1 定义 35](#_Toc17385444)

[6.2 语法 35](#_Toc17385445)

[6.3 常用模板引擎 35](#_Toc17385446)

[6.3.1 Thymeleaf 35](#_Toc17385447)

[第7章 JSON 37](#_Toc17385448)

[7.1 定义 37](#_Toc17385449)

[第8章 MarkDown 37](#_Toc17385450)

[8.1 标题 37](#_Toc17385451)

[8.2 列表 37](#_Toc17385452)

[8.2.1 无序列表 37](#_Toc17385453)

[8.2.2 有序列表 37](#_Toc17385454)

[8.3 引用 37](#_Toc17385455)

[8.4 粗体和斜体 38](#_Toc17385456)

[8.5 链接和图片 38](#_Toc17385457)

[8.6 分割线 38](#_Toc17385458)

[8.7 表格 38](#_Toc17385459)

[8.8 代办任务 38](#_Toc17385460)

[8.9 流程图 38](#_Toc17385461)

[第9章 Spring 38](#_Toc17385462)

[9.1 IOC-Inversion Of Control 38](#_Toc17385463)

[9.2 DI-Dependency Injection 39](#_Toc17385464)

[9.3 Bean 39](#_Toc17385465)

[9.4 Spring注入 39](#_Toc17385466)

[9.4.1 SET方法注入 39](#_Toc17385467)

[9.4.2 构造方法注入 39](#_Toc17385468)

[9.4.3 接口注入 39](#_Toc17385469)

[9.5 Spring 核心思想 39](#_Toc17385470)

[9.5.1 使用应用上下文 39](#_Toc17385471)

[9.5.2 Bean生命周期 40](#_Toc17385472)

[9.5.3 Spring框架 40](#_Toc17385473)

[9.6 装配Bean 40](#_Toc17385474)

[9.6.1 自动化装配 41](#_Toc17385475)

[9.7 搭建Spring环境 41](#_Toc17385476)

[9.7.1 IntellJ搭建Spring 5源码 41](#_Toc17385477)

[9.7.2 Junit4配置 41](#_Toc17385478)

[第10章 Spring-Boot实战 41](#_Toc17385479)

[10.1 快速构建 41](#_Toc17385480)

[第11章 测试 42](#_Toc17385481)

[11.1 Mock 42](#_Toc17385482)

[11.1.1 创建Mock 42](#_Toc17385483)

[11.1.2 设置对象期望和返回值 42](#_Toc17385484)

[11.1.3 结果验证 43](#_Toc17385485)

[11.1.4 Spy对象 43](#_Toc17385486)

[11.1.5 安装 43](#_Toc17385487)

[11.2 EasyMock 43](#_Toc17385488)

[11.2.1 主题步骤 43](#_Toc17385489)

[11.2.2 Mock对象的重用 44](#_Toc17385490)

[11.2.3 参数匹配器 44](#_Toc17385491)

[11.2.4 特殊Mock对象类型 44](#_Toc17385492)

[11.3 Junit 45](#_Toc17385493)

[11.3.1 基本用法 45](#_Toc17385494)

[11.3.2 API 45](#_Toc17385495)

[11.3.3 注释 46](#_Toc17385496)

[11.3.4 套件测试 46](#_Toc17385497)

[11.3.5 时间测试 46](#_Toc17385498)

[11.3.6 异常测试 46](#_Toc17385499)

[11.3.7 参数化测试 46](#_Toc17385500)

[第12章 JDBC 46](#_Toc17385501)

[12.1 常用接口 47](#_Toc17385502)

[12.1.1 Driver 47](#_Toc17385503)

[12.1.2 DriverManager 47](#_Toc17385504)

[12.1.3 Connection 47](#_Toc17385505)

[12.1.4 Statement 47](#_Toc17385506)

[12.1.5 ResultSet 47](#_Toc17385507)

[12.2 JDBC编程步骤 48](#_Toc17385508)

[12.2.1 加载数据库驱动 48](#_Toc17385509)

[12.2.2 获取数据库连接 48](#_Toc17385510)

[12.2.3 创建Statement对象 48](#_Toc17385511)

[12.2.4 执行SQL 48](#_Toc17385512)

[12.2.5 操作结果集 48](#_Toc17385513)

[12.2.6 回收数据库资源 48](#_Toc17385514)

[12.3 Statement与PreparedStatement对比 48](#_Toc17385515)

[12.4 结果集 48](#_Toc17385516)

[12.5 Blob类型数据 49](#_Toc17385517)

[12.6 示例 49](#_Toc17385518)

[12.6.1 连接MySql 49](#_Toc17385519)

[12.7 Batch Insert 49](#_Toc17385520)

[12.8 事务处理 50](#_Toc17385521)

[12.8.1 MySql对事务的支持 50](#_Toc17385522)

[12.9 使用元数据分析数据库 50](#_Toc17385523)

[12.9.1 DatebaseMetaData 51](#_Toc17385524)

[12.9.2 ResultSetMetaData 51](#_Toc17385525)

[第13章 Eclipse 51](#_Toc17385526)

[13.1 环境配置 51](#_Toc17385527)

[13.1.1 本地需要Groovy环境 51](#_Toc17385528)

[13.1.2 Groovy安装 51](#_Toc17385529)

[13.1.3 集成GIT 51](#_Toc17385530)

[13.1.4 Aptana3插件 54](#_Toc17385531)

[13.2 Clean 54](#_Toc17385532)

[13.2.1 功效 54](#_Toc17385533)

[13.2.2 clean后无法正常运行程序 54](#_Toc17385534)

[13.3 快捷键 54](#_Toc17385535)

[13.4 Debug模式 55](#_Toc17385536)

[13.4.1 NullPointerException 55](#_Toc17385537)

[13.4.2 SlowQueryReport 56](#_Toc17385538)

[13.5 构建Spring-Boot 56](#_Toc17385539)

[13.5.1 安装Spring Tool Suite 56](#_Toc17385540)

[13.5.2 全量构造 56](#_Toc17385541)

[13.5.3 使用spring boot initialize快速构造 56](#_Toc17385542)

[13.5.4 使用STS工具快速构建 57](#_Toc17385543)

[13.5.5 新建Maven配置 57](#_Toc17385544)

[第14章 IntelliJ 60](#_Toc17385545)

[14.1 快捷键 60](#_Toc17385546)

[14.2 打包 60](#_Toc17385547)

[14.2.1 本地仓库导出所有依赖的jar包 60](#_Toc17385548)

[第15章 Maven 60](#_Toc17385549)

[15.1 Maven简介 60](#_Toc17385550)

[15.2 配置 61](#_Toc17385551)

[15.2.1 下载 61](#_Toc17385552)

[15.2.2 设置环境变量 61](#_Toc17385553)

[15.2.3 检查是否安装成功 61](#_Toc17385554)

[15.2.4 使用阿里云仓库 61](#_Toc17385555)

[15.2.5 更新Maven本地库 62](#_Toc17385556)

[15.2.6 Maven依赖机制 62](#_Toc17385557)

[15.2.7 定制库到Maven本地资源库 62](#_Toc17385558)

[15.2.8 从Maven模板创建项目 63](#_Toc17385559)

[15.2.9 Maven目录布局 63](#_Toc17385560)

[15.2.10 Eclipse IDE 63](#_Toc17385561)

[15.2.11 更新POM 64](#_Toc17385562)

[15.2.12 将Maven变量加入Eclipse 65](#_Toc17385563)

[15.2.13 项目打包 67](#_Toc17385564)

[15.2.14 运行Jar文件 67](#_Toc17385565)

[15.3 创建WEB项目（WAR） 67](#_Toc17385566)

[15.4 使用模板来创建项目 69](#_Toc17385567)

[15.4.1 mvn archetype:generate 69](#_Toc17385568)

[15.4.2 默认模式 69](#_Toc17385569)

[15.4.3 定义 70](#_Toc17385570)

[15.5 清理项目 70](#_Toc17385571)

[15.5.1 获取最新部署 70](#_Toc17385572)

[15.5.2 部署进行生产 70](#_Toc17385573)

[15.6 单元测试 70](#_Toc17385574)

[15.7 项目资源安装到本地资源库 70](#_Toc17385575)

[15.8 生成基于Maven的项目文档站点 71](#_Toc17385576)

[15.9 部署站点 71](#_Toc17385577)

[15.9.1 Apache配置WebDAV 71](#_Toc17385578)

[15.9.2 配置WebDAV身份验证 71](#_Toc17385579)

[15.9.3 部署 72](#_Toc17385580)

[15.10 部署基于Maven的war文件到Tomcat 72](#_Toc17385581)

[15.10.1 指令 72](#_Toc17385582)

[15.10.2 Tomcat认证 72](#_Toc17385583)

[15.10.3 发布到Tomcat 73](#_Toc17385584)

[15.11 Eclipse创建一个MavenWeb项目 73](#_Toc17385585)

[15.12 其他指令 75](#_Toc17385586)

[第16章 Apache服务器 75](#_Toc17385587)

[16.1 简介 75](#_Toc17385588)

[16.2 搭建 75](#_Toc17385589)

[16.2.1 下载 75](#_Toc17385590)

[16.2.2 安装Apache服务器 76](#_Toc17385591)

[16.3 操作 77](#_Toc17385592)

[16.3.1 启动 77](#_Toc17385593)

[16.3.2 测试Apache服务器 77](#_Toc17385594)

[16.4 常用指令 78](#_Toc17385595)

[16.5 配置WebDAV 78](#_Toc17385596)

[16.5.1 修改httpd.conf 78](#_Toc17385597)

[16.5.2 创建用户密码 79](#_Toc17385598)

[16.5.3 配置WebDAV文件夹 79](#_Toc17385599)

[16.5.4 使用WebDAV客户端验证 81](#_Toc17385600)

[16.6 Tomcat服务器 81](#_Toc17385601)

[16.6.1 安装 81](#_Toc17385602)

[第17章 JAVA 82](#_Toc17385603)

[17.1 注释 82](#_Toc17385604)

[17.2 数据类型 82](#_Toc17385605)

[17.2.1 直接量的类型 82](#_Toc17385606)

[17.2.2 算术运算符 82](#_Toc17385607)

[17.2.3 赋值运算符 82](#_Toc17385608)

[17.2.4 位运算符 83](#_Toc17385609)

[17.2.5 比较运算符 83](#_Toc17385610)

[17.2.6 逻辑运算符 83](#_Toc17385611)

[17.2.7 三目运算符 83](#_Toc17385612)

[17.3 流程控制 83](#_Toc17385613)

[17.4 数组 83](#_Toc17385614)

[17.4.1 静态初始化 83](#_Toc17385615)

[17.4.2 动态初始化 83](#_Toc17385616)

[17.4.3 Foreach 83](#_Toc17385617)

[17.4.4 初始化 83](#_Toc17385618)

[17.4.5 Array 83](#_Toc17385619)

[17.5 方法 84](#_Toc17385620)

[17.6 对象和内存控制 84](#_Toc17385621)

[17.6.1 实例 85](#_Toc17385622)

[17.6.2 类变量 85](#_Toc17385623)

[17.6.3 父类构造器 85](#_Toc17385624)

[17.6.4 父子实例的内存控制 85](#_Toc17385625)

[17.6.5 Final修饰符 85](#_Toc17385626)

[17.6.6 字符串缓冲池 86](#_Toc17385627)

[17.7 代码块 86](#_Toc17385628)

[17.7.1 This 86](#_Toc17385629)

[17.8 封装Encapsulation 86](#_Toc17385630)

[17.9 Java的常用包 86](#_Toc17385631)

[17.10 多态Polymorphism 86](#_Toc17385632)

[17.11 Java增强包 87](#_Toc17385633)

[17.12 Equals 87](#_Toc17385634)

[17.13 单例类（Singleton） 87](#_Toc17385635)

[17.14 不可变类 87](#_Toc17385636)

[17.14.1 缓存实例的不可变类 87](#_Toc17385637)

[17.15 抽象类 87](#_Toc17385638)

[17.16 内部类 88](#_Toc17385639)

[17.16.1 静态内部类 88](#_Toc17385640)

[17.16.2 局部内部类 88](#_Toc17385641)

[17.16.3 Java8改进的匿名内部类 88](#_Toc17385642)

[17.16.4 Java8新增的Lambda表达式 88](#_Toc17385643)

[17.17 Enum枚举类 89](#_Toc17385644)

[17.18 对象与垃圾回收 89](#_Toc17385645)

[17.19 对象的软、弱和虚引用 90](#_Toc17385646)

[17.20 JAR包 90](#_Toc17385647)

[17.21 常用类 90](#_Toc17385648)

[17.21.1 String 90](#_Toc17385649)

[17.21.2 System 90](#_Toc17385650)

[17.21.3 Runtime 90](#_Toc17385651)

[17.21.4 Object 91](#_Toc17385652)

[17.21.5 String\StringBuffer\StringBuilder 91](#_Toc17385653)

[17.21.6 Random/ThreadLocalRandom 91](#_Toc17385654)

[17.21.7 BigDecimal 91](#_Toc17385655)

[17.21.8 Date 91](#_Toc17385656)

[17.21.9 Calendar 92](#_Toc17385657)

[17.21.10 DateFormat 92](#_Toc17385658)

[17.21.11 SimpleDateFormat 92](#_Toc17385659)

[17.21.12 Math 93](#_Toc17385660)

[17.21.13 DecimalFormat 93](#_Toc17385661)

[17.21.14 BegInteger 93](#_Toc17385662)

[17.21.15 BigDecimal 93](#_Toc17385663)

[17.21.16 NumberFormat 94](#_Toc17385664)

[17.21.17 对象克隆技术 94](#_Toc17385665)

[17.21.18 Arrays类 94](#_Toc17385666)

[17.21.19 Comparable接口 94](#_Toc17385667)

[17.21.20 Comparator 95](#_Toc17385668)

[17.22 获取系统时间 95](#_Toc17385669)

[17.23 定时调度 95](#_Toc17385670)

[17.23.1 Timer类 95](#_Toc17385671)

[17.23.2 TimerTask类 96](#_Toc17385672)

[17.24 正则表达式 96](#_Toc17385673)

[17.24.1 Pattern类和Matcher类 96](#_Toc17385674)

[17.24.2 String对正则表达式的支持 97](#_Toc17385675)

[17.25 集合 97](#_Toc17385676)

[17.25.1 Collection 97](#_Toc17385677)

[17.25.2 List 98](#_Toc17385678)

[17.25.3 Set 98](#_Toc17385679)

[17.25.4 Map 98](#_Toc17385680)

[17.25.5 Java8新增的Predicate操作集合 98](#_Toc17385681)

[17.25.6 Collections 98](#_Toc17385682)

[17.25.7 Stack 98](#_Toc17385683)

[17.25.8 Properties 98](#_Toc17385684)

[17.25.9 运算 98](#_Toc17385685)

[17.26 StringBuilder 98](#_Toc17385686)

[17.27 泛型 98](#_Toc17385687)

[17.27.1 泛型方法与通配符区别 99](#_Toc17385688)

[17.27.2 泛型接口 99](#_Toc17385689)

[17.27.3 泛型数组 99](#_Toc17385690)

[17.28 国际化程序 99](#_Toc17385691)

[17.28.1 Locale 99](#_Toc17385692)

[17.28.2 ResourceBundle 99](#_Toc17385693)

[17.28.3 国际化程序实现 100](#_Toc17385694)

[17.28.4 MessageFormat处理动态文本 100](#_Toc17385695)

[17.28.5 使用类文件代替资源文件 100](#_Toc17385696)

[17.29 异常处理 100](#_Toc17385697)

[17.29.1 常见的异常类之间的继承关系 100](#_Toc17385698)

[17.29.2 访问异常信息 101](#_Toc17385699)

[17.29.3 5 finally 101](#_Toc17385700)

[17.29.4 Java7的自动关闭资源的try语句 101](#_Toc17385701)

[17.30 断言 101](#_Toc17385702)

[17.31 AWT(Abstract Window Tookit) 102](#_Toc17385703)

[17.31.1 AWT容器 102](#_Toc17385704)

[17.32 Annotation 102](#_Toc17385705)

[17.32.1 @Override 102](#_Toc17385706)

[17.32.2 @Deprecated 102](#_Toc17385707)

[17.32.3 @SuppressWarnings 102](#_Toc17385708)

[17.32.4 @SafeVarags 102](#_Toc17385709)

[17.32.5 @FunctionalInterface 102](#_Toc17385710)

[17.32.6 自定义 102](#_Toc17385711)

[17.32.7 Retention和RetentionPolicy 103](#_Toc17385712)

[17.32.8 通过反射取得Annotation 103](#_Toc17385713)

[17.32.9 @Target 103](#_Toc17385714)

[17.32.10 @Documented 103](#_Toc17385715)

[17.32.11 @Inherited 103](#_Toc17385716)

[17.33 多线程 103](#_Toc17385717)

[17.33.1 Thread 104](#_Toc17385718)

[17.33.2 Runnable 104](#_Toc17385719)

[17.33.3 Attribute 105](#_Toc17385720)

[17.33.4 Control 105](#_Toc17385721)

[17.33.5 Sychronized 105](#_Toc17385722)

[17.33.6 Callable 105](#_Toc17385723)

[17.33.7 Priority 105](#_Toc17385724)

[17.33.8 Sychronized 105](#_Toc17385725)

[17.34 类加载机制与反射 106](#_Toc17385726)

[17.34.1 Class实例化 106](#_Toc17385727)

[17.34.2 调用类中的方法 106](#_Toc17385728)

[17.34.3 操作属性 106](#_Toc17385729)

[17.34.4 使用反射操作应用类 106](#_Toc17385730)

[17.34.5 类的生命周期 106](#_Toc17385731)

[17.34.6 动态代理 107](#_Toc17385732)

[17.35 JVM 107](#_Toc17385733)

[17.36 文件 107](#_Toc17385734)

[17.36.1 取得文件信息 108](#_Toc17385735)

[17.36.2 常用方法 108](#_Toc17385736)

[17.36.3 InputStream 108](#_Toc17385737)

[17.36.4 OutputStream 109](#_Toc17385738)

[17.36.5 文件输入流 109](#_Toc17385739)

[17.36.6 文件输出流 110](#_Toc17385740)

[17.36.7 缓存输入流 110](#_Toc17385741)

[17.36.8 缓存输出流 110](#_Toc17385742)

[17.36.9 数据输入流 110](#_Toc17385743)

[17.36.10 数据输出流 110](#_Toc17385744)

[17.36.11 字符输入流 110](#_Toc17385745)

[17.36.12 字符输出流 111](#_Toc17385746)

[17.36.13 管道流 111](#_Toc17385747)

[17.36.14 Scanner 111](#_Toc17385748)

[17.36.15 System中常量 111](#_Toc17385749)

[17.37 编码 111](#_Toc17385750)

[17.37.1 ISO8859-1 111](#_Toc17385751)

[17.37.2 GBK/GBK2312 112](#_Toc17385752)

[17.37.3 Unicode 112](#_Toc17385753)

[17.37.4 UTF 112](#_Toc17385754)

[17.37.5 本机编码 112](#_Toc17385755)

[17.38 序列化 112](#_Toc17385756)

[17.38.1 Serializable 112](#_Toc17385757)

[17.38.2 Externalizable 112](#_Toc17385758)

[17.38.3 Transient 112](#_Toc17385759)

[17.39 网络编程 112](#_Toc17385760)

[17.39.1 InetAddress 112](#_Toc17385761)

[17.39.2 ServerSocket 113](#_Toc17385762)

[17.39.3 Socket 113](#_Toc17385763)

[17.39.4 UDP 113](#_Toc17385764)

[17.39.5 URL 113](#_Toc17385765)

[17.39.6 Encode\Decode 113](#_Toc17385766)

[17.40 设计模式 113](#_Toc17385767)

[17.40.1 工厂模式 113](#_Toc17385768)

[17.40.2 代理模式 113](#_Toc17385769)

[17.40.3 适配器模式 114](#_Toc17385770)

[17.40.4 观察者设计模式 114](#_Toc17385771)

[17.41 Logger 114](#_Toc17385772)

[17.41.1 创建Logger对象 114](#_Toc17385773)

[17.41.2 Logger级别 114](#_Toc17385774)

[17.41.3 Handle 115](#_Toc17385775)

[17.41.4 Formatter 115](#_Toc17385776)

[17.42 实例 115](#_Toc17385777)

[17.42.1 进销存管理系统 115](#_Toc17385778)

[第18章 数据库 115](#_Toc17385779)

[18.1 SQL语句结构 115](#_Toc17385780)

[18.1.1 数据查询语句（Data Query Language） 115](#_Toc17385781)

[18.1.2 数据操作语言（Data Manipulation Language） 115](#_Toc17385782)

[18.1.3 事务处理语言（TPL） 115](#_Toc17385783)

[18.1.4 数据控制语言 （DCL） 115](#_Toc17385784)

[18.1.5 数据定义语言（DDL） 116](#_Toc17385785)

[18.1.6 指针控制语言（CCL） 116](#_Toc17385786)

[18.2 SQL数据类型 116](#_Toc17385787)

[18.2.1 字符型（VARCHARVS CHAR） 117](#_Toc17385788)

[18.2.2 文本型（TEXT） 117](#_Toc17385789)

[18.2.3 数值型（INT、NUMERIC、MONEY） 117](#_Toc17385790)

[18.2.3.1 INT\SAMLLINT\TINYINT 117](#_Toc17385791)

[18.2.3.2 NUMERIC 117](#_Toc17385792)

[18.2.3.3 MONEY\SMALLMONEY 117](#_Toc17385793)

[18.2.4 逻辑型（BIT） 117](#_Toc17385794)

[18.2.5 日期型 117](#_Toc17385795)

[18.2.6 SQL MS ACCESS\MYSQL\SQL SERVER数据类型 117](#_Toc17385796)

[18.2.6.1 Microsoft Access 数据类型 118](#_Toc17385797)

[18.2.6.2 MySQL 数据类型 118](#_Toc17385798)

[18.2.6.3 SQL Server 数据类型 120](#_Toc17385799)

[18.3 一些重要的SQL语句 123](#_Toc17385800)

[18.3.1 SELECT DISTINCT 126](#_Toc17385801)

[18.3.2 INSERT INTO 126](#_Toc17385802)

[18.3.3 检测NULL 126](#_Toc17385803)

[18.3.4 SELECT TOP 127](#_Toc17385804)

[18.3.5 SQL LIKE 127](#_Toc17385805)

[18.3.6 JOIN 127](#_Toc17385806)

[18.3.6.1 INNER JOIN/JOIN 127](#_Toc17385807)

[18.3.6.2 LEFT JOIN 127](#_Toc17385808)

[18.3.6.3 RIGHT JOIN 127](#_Toc17385809)

[18.3.6.4 FULL JOIN 128](#_Toc17385810)

[18.3.7 UNION 128](#_Toc17385811)

[18.3.8 SELECT INTO 128](#_Toc17385812)

[18.3.9 INSERT INTO SELECT 128](#_Toc17385813)

[18.3.10 ALTER TABLE 128](#_Toc17385814)

[18.3.11 AUTO INCREMENT 128](#_Toc17385815)

[18.3.12 VIEW 128](#_Toc17385816)

[18.3.13 Date 129](#_Toc17385817)

[18.3.14 SQL约束 129](#_Toc17385818)

[18.3.14.1 SQL NOT NULL约束 130](#_Toc17385819)

[18.3.14.2 SQL UNIQUE 130](#_Toc17385820)

[18.3.14.3 SQL PRIMARY KEY约束 130](#_Toc17385821)

[18.3.14.4 SQL FOREIGN KEY约束 130](#_Toc17385822)

[18.3.14.5 SQL DEFAULT约束 130](#_Toc17385823)

[18.3.14.6 SQL CHECK约束 130](#_Toc17385824)

[18.3.15 SQL INTERSECT 130](#_Toc17385825)

[18.3.16 SQL EXPECT 130](#_Toc17385826)

[18.3.17 SQL克隆表格 130](#_Toc17385827)

[18.3.18 SQL INDEX 130](#_Toc17385828)

[18.3.19 SQL子查询 130](#_Toc17385829)

[18.3.20 SQL注入 131](#_Toc17385830)

[18.3.21 SQL HAVING子句 131](#_Toc17385831)

[18.4 SQL事务 131](#_Toc17385832)

[18.4.1 属性 131](#_Toc17385833)

[18.4.1.1 原子性 131](#_Toc17385834)

[18.4.1.2 一致性 131](#_Toc17385835)

[18.4.1.3 隔离性 131](#_Toc17385836)

[18.4.1.4 持久性 131](#_Toc17385837)

[18.4.2 事务控制 131](#_Toc17385838)

[18.4.2.1 COMMIT 131](#_Toc17385839)

[18.4.2.2 ROLLBACK 132](#_Toc17385840)

[18.4.2.3 SAVEPOINT 132](#_Toc17385841)

[18.4.2.4 SET TRANSACTION 132](#_Toc17385842)

[18.4.3 创建临时表 132](#_Toc17385843)

[18.5 SQL函数 132](#_Toc17385844)

[18.5.1 SQL AGGREGATE函数 132](#_Toc17385845)

[18.5.2 SQL SCALAR函数 132](#_Toc17385846)

[18.5.3 SQL日期函数 132](#_Toc17385847)

[18.5.4 SQL FIELD() 134](#_Toc17385848)

[18.5.5 SQL SQRT() 134](#_Toc17385849)

[18.5.6 SQL RAND() 135](#_Toc17385850)

[18.5.7 SQL CONTACT() 135](#_Toc17385851)

[18.5.8 SQL ISNULL()\NVL()\IFNULL()\COALESCE() 135](#_Toc17385852)

[18.5.9 SQL REPLACE() 135](#_Toc17385853)

[18.5.10 SQL TRIM() 135](#_Toc17385854)

[18.5.11 SQL语句大全 135](#_Toc17385855)

[18.6 SQL反模式—BILL Karwin 135](#_Toc17385856)

[18.6.1 解决树形结构的查询和存储 135](#_Toc17385857)

[18.6.1.1 路径枚举 135](#_Toc17385858)

[18.6.1.2 嵌套集 135](#_Toc17385859)

[18.6.1.3 闭包表 135](#_Toc17385860)

[18.7 配置MySql数据库 135](#_Toc17385861)

[18.7.1 安装MySql 136](#_Toc17385862)

[18.7.2 使用sqldbx连接本机数据库 136](#_Toc17385863)

[18.7.3 MySql存储机制 138](#_Toc17385864)

[18.7.4 常用指令 138](#_Toc17385865)

[第19章 LintCode实例 139](#_Toc17385866)

[19.1 A+B问题 139](#_Toc17385867)

[第20章 Github 139](#_Toc17385868)

[20.1 Config 139](#_Toc17385869)

[20.1.1 设置姓名和邮箱地址 139](#_Toc17385870)

[20.1.2 设置私密文件 139](#_Toc17385871)

[20.1.3 简写 139](#_Toc17385872)

[20.1.4 提高命令输出的可读性 140](#_Toc17385873)

[20.1.5 设置SSH Key 140](#_Toc17385874)

[20.1.6 修改本地主目录路径 140](#_Toc17385875)

[20.2 Clone 141](#_Toc17385876)

[20.3 Git基础操作 141](#_Toc17385877)

[20.3.1 Git init—初始化仓库 141](#_Toc17385878)

[20.3.2 Git status—查看仓库的状态 141](#_Toc17385879)

[20.3.3 Git add—向暂存区中添加文件 141](#_Toc17385880)

[20.3.4 Git commit—保存仓库的历史记录 141](#_Toc17385881)

[20.3.4.1 记述一行提交信息 142](#_Toc17385882)

[20.3.4.2 记述详细提交信息 142](#_Toc17385883)

[20.3.5 Git log—查看提交日志 142](#_Toc17385884)

[20.3.6 Git diff—查看更改前后的差别 142](#_Toc17385885)

[20.3.7 Git branch—显示分支一览表 142](#_Toc17385886)

[20.3.8 Git checkout –b创建、切换分支 142](#_Toc17385887)

[20.3.9 Git checkout –切回上一个分支 142](#_Toc17385888)

[20.3.10 Git merge—合并分支 142](#_Toc17385889)

[20.3.11 Git log—graph—以图表形式查看分支 142](#_Toc17385890)

[20.3.12 Git reset—回溯历史版本 142](#_Toc17385891)

[20.3.13 Git reflog—查看当前仓库的操作历史 142](#_Toc17385892)

[20.3.14 Git commit –amend—修改提交信息 142](#_Toc17385893)

[20.3.15 Git rebase –i—压缩历史 143](#_Toc17385894)

[20.3.16 Git commit –am “” 143](#_Toc17385895)

[20.3.17 Git remote add—添加远程仓库 143](#_Toc17385896)

[20.3.18 Git push—推送至远程仓库 143](#_Toc17385897)

[20.3.19 Git clone—获取远程仓库 143](#_Toc17385898)

[20.3.20 Git pull—获取最新的远程仓库的分支 143](#_Toc17385899)

[20.3.21 Git remote –v—查看远端地址 143](#_Toc17385900)

[20.3.22 Git config –list—查看配置 143](#_Toc17385901)

[20.3.23 Git stash 144](#_Toc17385902)

[20.3.24 Git fetch--all 144](#_Toc17385903)

[20.4 多人协作 144](#_Toc17385904)

[20.5 标签管理 144](#_Toc17385905)

[20.6 建立git服务器 144](#_Toc17385906)

[20.7 快捷键 145](#_Toc17385907)

[20.8 Tips 145](#_Toc17385908)

[20.8.1 LF will be replaced by CRLF in hello\_world.php. 145](#_Toc17385909)

[20.8.2 Enter passphrase for key 145](#_Toc17385910)

[第21章 Linux 145](#_Toc17385911)

[21.1 安装 145](#_Toc17385912)

[21.1.1 VMware安装 145](#_Toc17385913)

[21.1.2 Linux安装 145](#_Toc17385914)

[21.2 改变linux配色方案 146](#_Toc17385915)

[21.3 Linux目录名 146](#_Toc17385916)

[21.4 基础命令 147](#_Toc17385917)

[21.5 内建命令 149](#_Toc17385918)

[21.5.1 外部命令 149](#_Toc17385919)

[21.5.2 内建命令 149](#_Toc17385920)

[21.6 环境变量 149](#_Toc17385921)

[21.6.1 全局环境变量 150](#_Toc17385922)

[21.6.2 局部环境变量 150](#_Toc17385923)

[21.6.3 删除环境变量 150](#_Toc17385924)

[21.6.4 设置PATH环境变量 150](#_Toc17385925)

[21.6.5 永久改变环境变量 150](#_Toc17385926)

[21.6.6 数组变量 150](#_Toc17385927)

[21.7 文件权限 151](#_Toc17385928)

[21.7.1 /etc/passwd 151](#_Toc17385929)

[21.7.2 /etc/shadow 151](#_Toc17385930)

[21.7.3 添加新用户 151](#_Toc17385931)

[21.7.4 删除用户 152](#_Toc17385932)

[21.7.5 修改用户 152](#_Toc17385933)

[21.7.6 使用Linux组 152](#_Toc17385934)

[21.7.7 理解文件权限 153](#_Toc17385935)

[21.7.8 改变权限 153](#_Toc17385936)

[21.8 VI 154](#_Toc17385937)

[21.9 Touch命令 154](#_Toc17385938)

[21.10 Mkdir命令 155](#_Toc17385939)

[21.11 Rm命令 155](#_Toc17385940)

[21.12 构建基本脚本 155](#_Toc17385941)

[21.12.1 创建shell脚本文件 155](#_Toc17385942)

[21.12.2 退出脚本 155](#_Toc17385943)

[21.13 结构化命令 156](#_Toc17385944)

[21.13.1 if-then 156](#_Toc17385945)

[21.13.2 if-then-else 156](#_Toc17385946)

[21.13.3 test 156](#_Toc17385947)

[21.13.4 复合条件测试 157](#_Toc17385948)

[21.13.5 if-then高级特性 157](#_Toc17385949)

[21.13.6 case 157](#_Toc17385950)

[21.13.7 for 157](#_Toc17385951)

[21.13.8 字符串拼接 157](#_Toc17385952)

[21.13.9 更改字符串分隔符 158](#_Toc17385953)

[21.13.10 通配符读取目录 158](#_Toc17385954)

[21.13.11 C语言风格的for命令 158](#_Toc17385955)

[21.13.12 while命令 158](#_Toc17385956)

[21.13.13 until命令 158](#_Toc17385957)

[21.13.14 break 158](#_Toc17385958)

[21.13.15 continue 158](#_Toc17385959)

[21.13.16 read 158](#_Toc17385960)

[21.14 处理用户输入 159](#_Toc17385961)

[21.14.1 命令行参数 159](#_Toc17385962)

[21.14.2 读取脚本名 159](#_Toc17385963)

[21.14.3 参数统计 159](#_Toc17385964)

[21.15 Debug 159](#_Toc17385965)

[21.15.1 Segmentation fault(core dumped) 160](#_Toc17385966)

[21.16 Nmon监控工具 160](#_Toc17385967)

[21.16.1 非交互模式 160](#_Toc17385968)

[第22章 DB2 160](#_Toc17385969)

[22.1 错误纠察 160](#_Toc17385970)

[22.1.1 SQL6048N 160](#_Toc17385971)

[第23章 Spark 161](#_Toc17385972)

[23.1 Spark组成 161](#_Toc17385973)

[23.2 RDD 161](#_Toc17385974)

[23.2.1 RDD属性 161](#_Toc17385975)

[23.2.2 RDD创建方式 161](#_Toc17385976)

[23.2.3 API 161](#_Toc17385977)

[23.3 基本概念 164](#_Toc17385978)

[23.4 运行流程 164](#_Toc17385979)

[23.5 运行特点 165](#_Toc17385980)

[23.6 DAG 165](#_Toc17385981)

[23.6.1 DAGScheduler 165](#_Toc17385982)

[23.6.2 TaskScheduler 166](#_Toc17385983)

[23.6.3 划分stage 166](#_Toc17385984)

[23.6.4 SchedulerBackend 167](#_Toc17385985)

[23.6.5 宽依赖(Wide Dependencies,Shuffle Dependencies) 167](#_Toc17385986)

[23.6.6 窄依赖（Narrow Dependencies） 168](#_Toc17385987)

[23.7 广播变量 168](#_Toc17385988)

[23.7.1 注意事项 168](#_Toc17385989)

[23.7.2 累加器 168](#_Toc17385990)

[23.8 RDD容错 168](#_Toc17385991)

[23.8.1 lineage 168](#_Toc17385992)

[23.8.2 checkpoint 168](#_Toc17385993)

[23.9 数据倾斜 169](#_Toc17385994)

[23.9.1 原因分析 169](#_Toc17385995)

[23.9.2 现象 169](#_Toc17385996)

[23.9.3 定位 169](#_Toc17385997)

[23.10 调优 169](#_Toc17385998)

[23.10.1 开发调优 169](#_Toc17385999)

[第24章 码制 170](#_Toc17386000)

[第25章 Python 170](#_Toc17386001)

[25.1 Python基础 170](#_Toc17386002)

[25.1.1 数据类型 170](#_Toc17386003)

[25.1.2 数学操作符 170](#_Toc17386004)

[25.1.3 比较操作符 170](#_Toc17386005)

[25.1.4 布尔操作符 170](#_Toc17386006)

[25.1.5 控制流 171](#_Toc17386007)

[25.1.6 异常处理 171](#_Toc17386008)

[25.1.7 列表 171](#_Toc17386009)

[25.1.8 函数 172](#_Toc17386010)

[25.1.9 元组数据类型 172](#_Toc17386011)

[25.1.10 字典和结构化数据 172](#_Toc17386012)

[25.1.11 字符串 173](#_Toc17386013)

[25.1.12 正则表达式 174](#_Toc17386014)

[25.1.13 读写文件 174](#_Toc17386015)

[25.1.14 shelve模块 175](#_Toc17386016)

[25.1.15 用pprint.pformat()函数保存变量 175](#_Toc17386017)

[25.1.16 组织文件 175](#_Toc17386018)

[25.1.17 异常 175](#_Toc17386019)

[25.1.18 日志 176](#_Toc17386020)

[25.2 从Web抓取信息 176](#_Toc17386021)

[25.2.1 webbrowser 176](#_Toc17386022)

[25.2.2 requests 176](#_Toc17386023)

[25.2.3 Beautiful Soup 176](#_Toc17386024)

[25.2.4 selenium 177](#_Toc17386025)

[25.3 处理Excel 178](#_Toc17386026)

[25.3.1 读取Excel 178](#_Toc17386027)

[25.3.2 写入Excel 178](#_Toc17386028)

[25.3.3 设置单元格的字体风格 179](#_Toc17386029)

[25.3.4 Font对象 179](#_Toc17386030)

[25.3.5 公式 179](#_Toc17386031)

[25.3.6 调整行和列 179](#_Toc17386032)

[25.3.7 合并单元格 179](#_Toc17386033)

[25.3.8 冻结窗格 179](#_Toc17386034)

[25.3.9 图表 179](#_Toc17386035)

[25.4 保持时间、计划任务和启动程序 180](#_Toc17386036)

[25.4.1 time模块 180](#_Toc17386037)

[25.4.2 四舍五入 180](#_Toc17386038)

[25.4.3 datetime模块 180](#_Toc17386039)

[25.5 多线程 181](#_Toc17386040)

[25.6 从Python启动其他程序 181](#_Toc17386041)

[25.7 发送电子邮件和短信 181](#_Toc17386042)

[25.7.1 SMTP 181](#_Toc17386043)

[25.7.2 IMAP 182](#_Toc17386044)

[25.8 BeautifulSoup 183](#_Toc17386045)

[25.8.1 Definition 183](#_Toc17386046)

[25.8.2 安装 183](#_Toc17386047)

[第26章 CMD相关 184](#_Toc17386048)

[26.1 网络相关 184](#_Toc17386049)

[26.1.1 查看被占用的端口 184](#_Toc17386050)

[26.2 文件操作 185](#_Toc17386051)

[26.2.1 创建文件 185](#_Toc17386052)

[26.3 显示更改 185](#_Toc17386053)

[第27章 八仙过海 185](#_Toc17386054)

[27.1 设计树形数据结构 185](#_Toc17386055)

[第28章 Virtuosity 185](#_Toc17386056)

[28.1 Windows 185](#_Toc17386057)

[28.1.1 Cmd创建文件 186](#_Toc17386058)

[第29章 书单 186](#_Toc17386059)

# 图目录

[图 2‑1 Http请求消息结构 7](#_Toc503816831)

# 表目录

[**表 3.1** WDMA模块软件接口 11](#_Toc503770710)

# Webservice

## 什么是Webservice

Webservice是一种跨编程语言和跨操作系统平台的远程调用技术

## Webservice平台技术

Websercice平台的三大技术：XSD+XML，SOAP，WSDL

### XSD+XML

Webservice采用HTTP协议传输数据，采用XML格式封装数据。XML是Webservice平台中表示税局的格式，与平台、产商无关。

XML Schema(XSD)，定义了一套标准的数据类型，解决XML的问题。

#### XML实体引用

#### XSD实例

# HTTP协议

HyperText Transfer Protocol，应用层的面向对象的协议，主要特点包括：

1. 支持客户/服务模式；
2. 简单快速：客户向服务器请求服务时，只需要传送请求方法和路径，常用请求方法有GET、HEAD、POST
3. 灵活：HTTP允许传输任意类型的数据对象
4. 无连接：每次连接处理一个请求，服务器处理完客户请求，并且收到客户应答后，即断开连接
5. 无状态：协议对于事物处理没有记忆能力

## URI/URL

URI(Uniform resource identifier)，统一资源标识符

URL(Uniform resource locater)，统一资源定位符

Http使用URI来传输数据和建立连接，URL是一种特殊的URI

[http://www.aspxfans.com:8080/news/index.asp?boardID=5&ID=24618&page=1#name](http://www.aspxfans.com:8080/news/index.asp?boardID=5&ID=24618&page=1%23name)

1. 协议部分：URL协议部分http:
2. 域名部分：IP或者地址（可使用nslookup + [www.baidu.com](http://www.baidu.com)查看，查看接口占用netstat -ano）
3. 端口部分：跟在域名后面为端口，用“:”分隔，默认使用80端口
4. 虚拟目录部分：从域名第一个/到最后一个/之间
5. 文件名部分：从域名最后一个/到？或者#或者结束
6. 锚部分：从#开始到最后
7. 参数部分：从？到#都是参数部分，又称搜索部分、查询部分，允许多个参数，之间用哪个&作为分隔符

## Http之请求消息Request

客户发送Http请求由是四个部分组成，请求行、请求头部、空行和请求数据



图 2‑1 Http请求消息结构

第一部分：请求行，用来说明请求类型,要访问的资源以及所使用的HTTP版本.

第二部分：请求头部，紧接着请求行（即第一行）之后的部分，用来说明服务器用的附加信息

第三部分：空行，请求头部后面的空行是必须的

即使第四部分的请求数据为空，也必须有空行。

第四部分：请求数据也叫主体，可以添加任意的其他数据。

## Http之响应消息Response

Http响应也由四部分组成，状态行、消息头部、空行和响应正文

第一部分：状态行，由HTTP协议版本号， 状态码， 状态消息 三部分组成。

第一行为状态行，（HTTP/1.1）表明HTTP版本为1.1版本，状态码为200，状态消息为（ok）

第二部分：消息报头，用来说明客户端要使用的一些附加信息

第二行和第三行为消息报头，

Date:生成响应的日期和时间；Content-Type:指定了MIME类型的HTML(text/html),编码类型是UTF-8

第三部分：空行，消息报头后面的空行是必须的

第四部分：响应正文，服务器返回给客户端的文本信息。

空行后面的html部分为响应正文。

## Http之状态码

状态代码有三位数字组成，第一个数字定义了响应的类别，共分五种类别:

1xx：指示信息--表示请求已接收，继续处理

2xx：成功--表示请求已被成功接收、理解、接受

3xx：重定向--要完成请求必须进行更进一步的操作

4xx：客户端错误--请求有语法错误或请求无法实现

5xx：服务器端错误--服务器未能实现合法的请求

常见状态码：

200 OK //客户端请求成功

400 Bad Request //客户端请求有语法错误，不能被服务器所理解

401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用

403 Forbidden //服务器收到请求，但是拒绝提供服务

404 Not Found //请求资源不存在，eg：输入了错误的URL

500 Internal Server Error //服务器发生不可预期的错误

503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

## Http请求方法

根据HTTP标准，HTTP请求可以使用多种请求方法。

HTTP1.0定义了三种请求方法： GET, POST 和 HEAD方法。

HTTP1.1新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

GET 请求指定的页面信息，并返回实体主体。

HEAD 类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头

POST 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。

PUT 从客户端向服务器传送的数据取代指定的文档的内容。

DELETE 请求服务器删除指定的页面。

CONNECT HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。

OPTIONS 允许客户端查看服务器的性能。

TRACE 回显服务器收到的请求，主要用于测试或诊断。

## Http工作原理

HTTP协议定义Web客户端如何从Web服务器请求Web页面，以及服务器如何把Web页面传送给客户端。HTTP协议采用了请求/响应模型。客户端向服务器发送一个请求报文，请求报文包含请求的方法、URL、协议版本、请求头部和请求数据。服务器以一个状态行作为响应，响应的内容包括协议的版本、成功或者错误代码、服务器信息、响应头部和响应数据。

以下是 HTTP 请求/响应的步骤：

1、客户端连接到Web服务器

一个HTTP客户端，通常是浏览器，与Web服务器的HTTP端口（默认为80）建立一个TCP套接字连接。

2、发送HTTP请求

通过TCP套接字，客户端向Web服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据4部分组成。

3、服务器接受请求并返回HTTP响应

Web服务器解析请求，定位请求资源。服务器将资源复本写到TCP套接字，由客户端读取。一个响应由状态行、响应头部、空行和响应数据4部分组成。

4、释放连接TCP连接

若connection 模式为close，则服务器主动关闭TCP连接，客户端被动关闭连接，释放TCP连接;若connection 模式为keepalive，则该连接会保持一段时间，在该时间内可以继续接收请求;

5、客户端浏览器解析HTML内容

客户端浏览器首先解析状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的HTML文档和文档的字符集。客户端浏览器读取响应数据HTML，根据HTML的语法对其进行格式化，并在浏览器窗口中显示。

例如：在浏览器地址栏键入URL，按下回车之后会经历以下流程：

1、浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址;

2、解析出 IP 地址后，根据该 IP 地址和默认端口 80，和服务器建立TCP连接;

3、浏览器发出读取文件(URL 中域名后面部分对应的文件)的HTTP 请求，该请求报文作为 TCP 三次握手的第三个报文的数据发送给服务器;

4、服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器;

5、释放 TCP连接;

6、浏览器将该 html 文本并显示内容;

## Post和Get差别

1、GET提交，请求的数据会附在URL之后（就是把数据放置在HTTP协议头中），以?分割URL和传输数据，多个参数用&连接；例 如：login.action?name=hyddd&password=idontknow&verify=%E4%BD%A0 %E5%A5%BD。如果数据是英文字母/数字，原样发送，如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用BASE64加密，得出如： %E4%BD%A0%E5%A5%BD，其中％XX中的XX为该符号以16进制表示的ASCII。

POST提交：把提交的数据放置在是HTTP包的包体中。上文示例中红色字体标明的就是实际的传输数据

因此，GET提交的数据会在地址栏中显示出来，而POST提交，地址栏不会改变

2、传输数据的大小：首先声明：HTTP协议没有对传输的数据大小进行限制，HTTP协议规范也没有对URL长度进行限制。

而在实际开发中存在的限制主要有：

GET:特定浏览器和服务器对URL长度有限制，例如 IE对URL长度的限制是2083字节(2K+35)。对于其他浏览器，如Netscape、FireFox等，理论上没有长度限制，其限制取决于操作系 统的支持。

因此对于GET提交时，传输数据就会受到URL长度的 限制。

POST:由于不是通过URL传值，理论上数据不受 限。但实际各个WEB服务器会规定对post提交数据大小进行限制，Apache、IIS6都有各自的配置。

3、安全性

POST的安全性要比GET的安全性高。比如：通过GET提交数据，用户名和密码将明文出现在URL上，因为(1)登录页面有可能被浏览器缓存；(2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了，除此之外，使用GET提交数据还可能会造成Cross-site request forgery攻击

4、Http get,post,soap协议都是在http上运行的

（1）get：请求参数是作为一个key/value对的序列（查询字符串）附加到URL上的

查询字符串的长度受到web浏览器和web服务器的限制（如IE最多支持2048个字符），不适合传输大型数据集同时，它很不安全

（2）post：请求参数是在http标题的一个不同部分（名为entity body）传输的，这一部分用来传输表单信息，因此必须将Content-type设置为:application/x-www-form- urlencoded。post设计用来支持web窗体上的用户字段，其参数也是作为key/value对传输。

但是：它不支持复杂数据类型，因为post没有定义传输数据结构的语义和规则。

（3）soap：是http post的一个专用版本，遵循一种特殊的xml消息格式

Content-type设置为: text/xml 任何数据都可以xml化。

Http协议定义了很多与服务器交互的方法，最基本的有4种，分别是GET,POST,PUT,DELETE. 一个URL地址用于描述一个网络上的资源，而HTTP中的GET, POST, PUT, DELETE就对应着对这个资源的查，改，增，删4个操作。 我们最常见的就是GET和POST了。GET一般用于获取/查询资源信息，而POST一般用于更新资源信息.

我们看看GET和POST的区别

GET提交的数据会放在URL之后，以?分割URL和传输数据，参数之间以&相连，如EditPosts.aspx?name=test1&id=123456. POST方法是把提交的数据放在HTTP包的Body中.

GET提交的数据大小有限制（因为浏览器对URL的长度有限制），而POST方法提交的数据没有限制.

GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值。

GET方式提交数据，会带来安全问题，比如一个登录页面，通过GET方式提交数据时，用户名和密码将出现在URL上，如果页面可以被缓存或者其他人可以访问这台机器，就可以从历史记录获得该用户的账号和密码.

# XML

## XML定义

XML(Extensible Markup Language)，可扩展标记语言，标准通用标记语言的子集，是一种用于标记电子文件使其具有结构性的标记语言。用来传输数据。

## XML构成

由文件头和文件体组成。

文件头：<?xml version="1.0" encoding="gb2312"?>

文件体：文件体中包含的是XML文件的内容，XML元素是XML文件内容的基本单元。从语法讲，一个元素包含一个起始标记、一个结束标记以及标记之间的数据内容。

### 注释

XML的注释与HTML的注释相同，以“<!--”开始，以“-->”结束。

### 区分大小写

在HTML中是不区分大小写的，而XML区分大小写，包括标记，属性，指令等。

### 标记

XML标记与HTML标记相同，“<”表示一个标记的开始，“>” 表示一个标记的结束。

### 属性

XML属性的使用与HTML属性基本相同，但需要注意的是属性值要加双引号。

### 实体引用

实体引用是指分析文档时会被字符数据取代的元素，实体引用用于XML文档中的特殊字符，否则这些字符会被解释为元素的组成部分

|  |  |  |
| --- | --- | --- |
| &lt; | < | 小于 |
| &gt; | > | 大于 |
| &amp; | & | 和号 |
| &apos; | ' | 单引号 |
| &quot; | " | 引号 |

3‑1实体引用对照表

### CDATA

不由XML解析器进行解析的文本文件，CDATA部分由“<![CDATA[”开始，由”]]>”结束。

CDATA 部分不能包含字符串 "]]>"。也不允许嵌套的 CDATA 部分。

标记 CDATA 部分结尾的 "]]>" 不能包含空格或折行。

### PCDATA

PCDATA(Parsed Character Data):被解析的字符数据。

### 处理指令

处理指令使用来给处理XML文件的应用程序提供信息的，处理指令的格式如下：

<?处理指令名称 处理指令信息?>

例如，XML声明就是一条处理指令：

<?xml version="1.0" encoding="gb2312"?>

其中，“xml”是处理指令名称，version="1.0" encoding="gb2312"是处理指令信息。

## 应用

### XSD-XML SCHEMA

XML Schema提供了两个在实例文档中使用的特殊属性，用于指出模式文档的位置。这两个属性是：xsi:schemaLocation和xsi:noNamespaceSchemaLocation，前者用于声明了目标名称空间的模式文档，后者用于没有目标名称空间的模式文档，它们通常在实例文档中使用。

#### xsi:schemaLocation属性

xsi:schemaLocation属性的值由一个URI引用对组成，两个URI之间以空白符分隔。第一个URI是名称空间的名字，第二个URI给出模式文档的位置，模式处理器将从这个位置读取模式文档，该模式文档的目标名称空间必须与第一个URI相匹配。

Xmlns=”xx”，申明默认名称空间

xsi:schemaLocation属性的值也可以由多个URI引用对组成，每个URI引用对之间使用空白符分隔

#### xsi:noNamespaceSchemaLocation属性

xsi:noNamespaceSchemaLocation属性用于引用没有目标名称空间的模式文档。与xsi:schemaLocation属性不同的是，xsi:noNamespaceSchemaLocation属性的值是单一的值，只是用于指定模式文档的位置。

# XSD

## XSD定义

XSD，XML Schema的作用是定义 XML 文档的合法构建模块，类似DTD。

## 元素类型

### 简易元素

* xs:string
* xs:decimal
* xs:integer
* xs:boolean
* xs:date
* xs:time

### 数据类型限定

restriction

|  |  |
| --- | --- |
| **限定** | **描述** |
| enumeration | 定义可接受值的一个列表 |
| fractionDigits | 定义所允许的最大的小数位数。必须大于等于0。 |
| length | 定义所允许的字符或者列表项目的精确数目。必须大于或等于0。 |
| maxExclusive | 定义数值的上限。所允许的值必须小于此值。 |
| maxInclusive | 定义数值的上限。所允许的值必须小于或等于此值。 |
| maxLength | 定义所允许的字符或者列表项目的最大数目。必须大于或等于0。 |
| minExclusive | 定义数值的下限。所允许的值必需大于此值。 |
| minInclusive | 定义数值的下限。所允许的值必需大于或等于此值。 |
| minLength | 定义所允许的字符或者列表项目的最小数目。必须大于或等于0。 |
| pattern | 定义可接受的字符的精确序列。 |
| totalDigits | 定义所允许的阿拉伯数字的精确位数。必须大于0。 |
| whiteSpace | 定义空白字符（换行、回车、空格以及制表符）的处理方式。 |

### XSD指示器

Order 指示器：

* All
* Choice
* Sequence

Occurrence 指示器：

* maxOccurs
* minOccurs

Group 指示器：

* Group name
* attributeGroup name

## 示例

<xs:simpleType name="stringtype">

<xs:restriction base="xs:string"/>

</xs:simpleType>

<xs:simpleType name="inttype">

<xs:restriction base="xs:positiveInteger"/>

</xs:simpleType>

<xs:simpleType name="dectype">

<xs:restriction base="xs:decimal"/>

</xs:simpleType>

<xs:simpleType name="orderidtype">

<xs:restriction base="xs:string">

<xs:pattern value="[0-9]{6}"/>

</xs:restriction>

</xs:simpleType>

# WSDL

# HTML

## 定义

HTML，超文本标记语言，Hyper Text Makeup Language

## 语法

<h1> - <h6>标题

<p>段落

<a>链接，anchor，<a href="http://www.w3cschool.cn">这是一个链接</a>，Href

<img>自关闭元素，不需要配对结尾，用src属性指向地址

## 常用模板引擎

### Thymeleaf

|- <html xmlns:th="http://www.thymeleaf.org">，引入域空间

|-简单表达式

|-${...}  变量表达式

|-\*{...}  选择变量表达式

|-#{...}  消息表达式

|-@{...}  链接url表达式

|-字面量

|-'one text','another one!',...   文本

|-0,34,3.0,12.3,... 数值

|-true false 布尔类型

|-null 空

|-one,sometext,main 文本字符

|-文本操作

|-+  字符串连接

|-|The name is ${name}|  字符串连接

|-算术运算

|-+ , - , \* , / , %  二元运算符

|--  负号（一元运算符）

|-布尔操作

|-and,or  二元操作符

|-!,not 非（一元操作符）

|-关系操作符

|-> , < , >= , <= (gt , lt , ge , le)

|-== , != (eq, ne)

|-条件判断

|-(if) ? (then)      if-then

|-(if) ? (then) : (else)   if-then-else

|-常用标签

关键字 　　功能介绍 　　　　案例

th:id 　　替换id 　　　　 <input th:id="'xxx' + ${collect.id}"/>

th:text 　文本替换 　　　　<p th:text="${collect.description}">description</p>

th:utext 支持html的文本替换 <p th:utext="${htmlcontent}">conten</p>

th:object 替换对象 　　　　<div th:object="${session.user}">

th:value 属性赋值 　　　　<input th:value="${user.name}" />

th:with 变量赋值运算 　　　　<div th:with="isEven=${prodStat.count}%2==0"></div>

th:style 设置样式 　　　　　　　　th:style="'display:' + @{(${sitrue} ? 'none' : 'inline-block')} + ''"

th:onclick 点击事件 　　　　　　th:onclick="'getCollect()'"

th:each 属性赋值 　　　　　　　　tr th:each="user,userStat:${users}">

th:if 判断条件 　　　　　　　　<a th:if="${userId == collect.userId}" >

th:unless 和th:if判断相反 　　　　<a th:href="@{/login}" th:unless=${session.user != null}>Login</a>

th:href 链接地址 　　　　　　　　　　<a th:href="@{/login}" th:unless=${session.user != null}>Login</a> />

th:switch 多路选择 配合th:case 使用 <div th:switch="${user.role}">

th:case th:switch的一个分支 　　　　<p th:case="'admin'">User is an administrator</p>

th:fragment 布局标签，定义一个代码片段，方便其它地方引用 <div th:fragment="alert">

th:include 布局标签，替换内容到引入的文件 <head th:include="layout :: htmlhead" th:with="title='xx'"></head> />

th:replace 布局标签，替换整个标签到引入的文件 <div th:replace="fragments/header :: title"></div>

th:selected selected选择框 选中 th:selected="(${xxx.id} == ${configObj.dd})"

th:src 图片类地址引入 　　　　　　<img class="img-responsive" alt="App Logo" th:src="@{/img/logo.png}" />

th:inline 定义js脚本可以使用变量 <script type="text/javascript" th:inline="javascript">

th:action 表单提交的地址 　　　　<form action="subscribe.html" th:action="@{/subscribe}">

th:remove 删除某个属性 　　　　<tr th:remove="all">

　　　　　　　　　　　　　　　　　　　　1.all:删除包含标签和所有的孩子。

　　　　　　　　　　　　　　　　　　　　2.body:不包含标记删除,但删除其所有的孩子。

　　　　　　　　　　　　　　　　　　　　3.tag:包含标记的删除,但不删除它的孩子。

　　　　　　　　　　　　　　　　　　　　4.all-but-first:删除所有包含标签的孩子,除了第一个。

　　　　　　　　　　　　　　　　　　　　5.none:什么也不做。这个值是有用的动态评估。

th:attr 设置标签属性，多个属性可以用逗号分隔 比如 th:attr="src=@{/image/aa.jpg},title=#{logo}"，此标签不太优雅，一般用的比较少。

# JSON

## 定义

JSON,JavaScript Object Notation,JavaScript对象表示法

JSON是存储和交换文本信息的语法，类似于XML

JSON比XML更小、更快，更易解析

# MarkDown

## 标题

# 一级标题

## 二级标题

## 列表

### 无序列表

* 列表1

### 有序列表

1. 有序列表1

## 引用

>+space+context

## 粗体和斜体

\*\*粗体\*\*

\*斜体\*

## 链接和图片

[显示文本]（links-adress）

[显示文本](graph-adress)

## 分割线

另起一行，\*\*\*

## 表格

Head1|head2

X | y

## 代办任务

* []、- [x]

## 流程图

“`开头，”`结尾

# Spring

Spring是一个开源的轻量级Java SE（Java 标准版本）/Java EE（Java 企业版本）开发应用框架，其目的是用于简化企业级应用程序开发。

Spring能帮我们根据配置文件创建及组装对象之间的依赖关系。

Spring 面向切面编程能帮助我们无耦合的实现日志记录，性能统计，安全控制。

Spring能非常简单的帮我们管理数据库事务。

Spring还提供了与第三方数据访问框架（如Hibernate、JPA）无缝集成，而且自己也提供了一套JDBC访问模板，来方便数据库访问。

Spring还提供与第三方Web（如Struts、JSF）框架无缝集成，而且自己也提供了一套Spring MVC框架，来方便web层搭建。

Spring能方便的与Java EE（如Java Mail、任务调度）整合，与更多技术整合（比如缓存框架）。

## IOC-Inversion Of Control

传统Java SE程序设计，我们直接在对象内部通过new进行创建对象，是程序主动去创建依赖对象；而IoC是有专门一个容器来创建这些对象，即由Ioc容器来控制对象的创建；谁控制谁？当然是IoC 容器控制了对象；控制什么？那就是主要控制了外部资源获取（不只是对象包括比如文件等）。

IoC容器就是具有依赖注入功能的容器，IoC容器负责实例化、定位、配置应用程序中的对象及建立这些对象间的依赖。应用程序无需直接在代码中new相关的对象，应用程序由IoC容器进行组装。在Spring中BeanFactory是IoC容器的实际代表者。

## DI-Dependency Injection

●谁依赖于谁：当然是应用程序依赖于IoC容器；

●为什么需要依赖：应用程序需要IoC容器来提供对象需要的外部资源；

●谁注入谁：很明显是IoC容器注入应用程序某个对象，应用程序依赖的对象；

●注入了什么：就是注入某个对象所需要的外部资源（包括对象、资源、常量数据）。

## Bean

由IoC容器管理的那些组成你应用程序的对象我们就叫它Bean， Bean就是由Spring容器初始化、装配及管理的对象，除此之外，bean就与应用程序中的其他对象没有什么区别了。

配置文件的根元素是beans，每个组件使用bean元素来定义。Bean的两个元素是必须的：id-组件的默认名称，class-类的全名，name-是class属性的一个别名

## Spring注入

### SET方法注入

Property

### 构造方法注入

Constructor-org

### 接口注入

## Spring 核心思想

降低Java开发的复杂性

* 基于POJO(Plain Old Java Object)的轻量级和最小侵入性编程；
* 通过依赖注入和面向接口实现松耦合；
* 基于切面和惯例进行申明式编程；
* 通过切面和模板较少样板式代码

### 使用应用上下文

* AnnotationConfigApplicationContext：从一个或多个基于Java配置类中加载Spring应用上下文
* AnnotaConfigWebApplicationContex：从一个或多个基于Java配置类中加载Spring Web应用上下文
* ClassPathXmlApplictionContext：从类路径下的一个或多个XML配置文件中加载上下文定义，把应用上下文的定义文件作为类资源
* FileSystemXmlapplicationcntext：从文件系统下的一个或多个XML配置文件中加载上下文定义
* xmlWebApllicationContext：从Web应用下的一个或多个XML配置文件中加载文件定义

### Bean生命周期



### Spring框架



## 装配Bean

Spring提供三种装配机制。

* 在XML中进行显示装配；
* 在JAVA中进行显示装配；
* 隐式的bean发现机制和自动装配

### 自动化装配

* 组件扫描
* 自动装配

@Component();

@Component(“bean\_name”);

@Named();

@ComponentScan();

@ComponentScan(“Libname”);

@ComponentScan(basePackages = “Libname”);

@ComponentScan(basePackages = {“Nm1”,”Nm2”});

@ComponentScan(basePackageClases ={CDplayer.class,DVDPlayer.class});

@Autowired();

@Autowired(required=false);若Spring没有找到合适的bean不会报异常，使得该bean处于未装配状态

## 搭建Spring环境

### IntellJ搭建Spring 5源码

* IntellJ通过从git导入https://github.com/spring-projects/spring-framework.git
* 安装gradle，下载<https://gradle.org/install/#with-a-package-manager>，下载完成增加系统path变量
* 查看Spring目录下的import-into-idea.md文件，按照指南开始构建
  + Precompile `spring-oxm` with `./gradlew :spring-oxm:compileTestJava`
  + import into IntelliJ(选择gradle项目)

### Junit4配置

* 下载Junit4，<https://plugins.jetbrains.com/plugin/3064-junitgenerator-v2-0>

# Spring-Boot实战

## 快速构建

1. Spring Boot initializer：<https://start.spring.io/>

# 测试

## Mock

Mock使用虚拟对象来模拟测试的方法。

* Mock对象：模拟对象
* Stub桩：替代具体功能的程序段

添加Mock依赖：

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-all</artifactId>

<version>2.0.2-beta</version>

</dependency>

Import static org.mockito.Mockito.\*;

### 创建Mock

* 通过方法创建  
  class CreateMock {   
   @Before   
   public void setup() { mockUserDao = mock(UserDao.class);   
   userService = new UserServiceImpl();   
   userService.setUserDao(mockUserDao); }  
   }
* 通过注解创建  
  class CreateMock {   
   @Mock UserDao mockUserDao;   
   @InjectMocks private UserServiceImpl userService;   
   @Before public void setUp() { //初始化对象的注  
   MockitoAnnotations.initMocks(this); }   
  }

### 设置对象期望和返回值

以下写法结果相同

1)When(mock.someMethod()).thenReturn(values1).thenReturn(values2);

2)When(mock.someMethod()).thenReturn(values1,vaue2);

3)When(mock.someMethod()).thenReturn(values1);

When(mock.someMethod()).thenReturn(values2);

4)doReturn(value1).doReturn(value2).when(mock).someMethod();

若返回void，设置doNothing

doNothing().when(mock).someMethod;

对方法设定返回异常

When (mock.someMethod()).thenThrow(new RuntimeException());

doThrow(new RuntimeException).when(mock).someMethod();

参数匹配器

When(list.get(anInt())).thenReturn(“hello”);

### 结果验证

验证调用次数

Verify(mock1,timeout(100).time(2)).get(anyInt());

Nerver()，没有被调用，相当于times(0);

atLeast(N) 至少被调用N次

atLeastOnce() 相当于atLeast(1)

atMost(N) 最多被调用N次

超时验证

Timeout

方法调用顺序

Inorder可以验证方法调用顺序

verifyNoMoreInteractions 和 verifyZeroInteractions

verifyNoMoreInteractions：查询是否存在被调用，但未被 verify 验证的方法

verifyZeroInteractions：verifyZeroInteractions

ArgumentCaptor 参数捕获器

可在验证时对方法的参数进行捕获，最后验证捕获的参数值。如果方法有多个参数都要捕获验证，那就需要创建多个ArgumentCaptor对象处理。

### Spy对象

Mockito提供给我们一种真实对象的操作方——Spy

### 安装

下载3 个库，创建并放到libs目录下，并添加工程依赖：

mockito-core-2.0.71-beta.jar，http://mockito.org/

objenesis-tck-2.4.jar，http://objenesis.org/

byte-buddy-1.4.5.jar，<http://bytebuddy.net/#/>

## EasyMock

### 主题步骤

* 使用EasyMock生成Mock对象
  + 使用EasyMock动态构建：  
    ResulteSet mockResultSet = EasyMock.create(ResultSet.class);
  + 使用IMocksControl构建：  
    IMocksControl control = EasyMock.createControl();  
    java.sql.Connection mockConnection = control.createMock (Connetcion.class);  
    java.sql.Resultset mockResultset = control.createMock(ResultSet.class);
* 设定Mock对象的预期行为和输出（Record状态，录制预期行为和输出）
  + Invoke mockObject.method()
  + 通过expectLastCall获取上一次方法调用对应的IExpectationSetters实例
    - mockObject.method(parameters…);  
      expectLastCall().andReturn(“My return value”);
  + 通过IExpectationSettes实例设定Mock对象的预期输出
    - 产生返回值
      * 一次设定：expectLastCall().andReturn(“My return values”);
      * 默认设定：expectLastCall().andStubReturn(mockObject);
    - 异常
      * 一次设定：IExpectationSetters<T> andThrows(Throwable throwable)
      * 默认设定：void andStubThrow(Throwable throwable)
    - 设定次数
      * expectLastCall().andReturn(“My return values”).time(3);
      * expectLastCall().andReturn(“My return values”).time(3,5);
      * expectLastCall().andReturn(“My return values”).atLeastOnce();
      * expectLastCall().andReturn(“My return values”).anyTimes();
* 将Mock对象切换到Replay状态
  + Replay(mockObject)
  + Control.replay()
* 调用Mock对象进行单元测试
* 对Mock对象的行为进行验证
  + Verify(mockObject)
  + Control.replay()

### Mock对象的重用

1. reset
2. IMocksControl reset

### 参数匹配器

1. anyObject()
2. aryEq()
3. isNull()
4. notNull()
5. sam()
6. Lt(X value)\leq\geq\gt
7. startWith)\contains()\endWith()
8. matches()

自定义参数匹配器需要实现org.easymock.IArgumentMatcher借口，matches实现匹配逻辑，appendTo(StringBuffer buffer)添加匹配失败的提示的信息；

需要使用静态方法包装，通过reportMatcher方法报告给EasyMock。

Public Static String sqlEquals(String in){

reportMatcher(new SQLEquals(in));

return in;

}

mockStatment.executeQuery(sqlEquals(SELECT \* FROM table))；

### 特殊Mock对象类型

Stcik Mock——时序敏感

Nick Mock——使用Mock对象对非预期方法调用默认为抛出AssertionError，使用此种返回0，null，或者false

## Junit

### 基本用法

* 创建TestJunit.java测试类
* 测试类中添加testMethodName()的方法
* 方法中添加Annotation @Test
* 测试中使用Junit的assertEquals API检查
* 创建TestRunner类
* 使用JunitCore类则runClasses方法运行上述测试类的案例：  
  Result result = JUnitCore.runClasses(TestJunit.class);
* 获取在Result Object中运行的测试案例的结果
* 获取Result Object的getFailures()方法中的失败结果：  
  for(Failure failure : result.getFailures()){  
   System.out.println(failure.toString());  
  }
* 获取Result Object的wasSuccessful()方法中成功结果:  
  System.out.println(result.wasSuccessful());

### API

Junit中junit.framework包含了它的核心类

* Asert
  + Void assertEquals(Boolean expected, Boolean actual)
  + Void assertFalse(Boolean condition)
  + Void assertNotNull(Object object)
  + Void assertNull(Object object)
  + Void assertTrue(Boolean condition)
  + Void fail()
  + Void assertSame(expectedObj,resultObj)
  + Void assertArrayEquals(expectedArray, resultArray)
* TestCase
  + Int countTestCase()
  + TestResult createResult()
  + String getName()
  + TestResult run()
  + Void run(TestResult result)
  + Void setName(String name)
  + Void setup()
  + Void tearDown()
  + String toString()
* TestResult
* TestSuite

### 注释

* @Test:说明依附在Junit的public void方法作为一个测试案例
* @Before：该方法在test方法前运行，比如创建对象
* @After：在test后运行，比如释放在Before中分配的资源
* @BeforeClass：在public void方法中加该注释表明该方法在类中所有方法前运行
* @AfterClass:是方法在所有测试结束后执行，清理活动
* @Ignore：忽略不需要执行的测试

执行顺序：BeforeClass->Before->test 1->After->Before->test 2->After->AfterClass

### 套件测试

捆绑几个单元测试案例一起执行。@RunWith\@Suite

* 创建一个java类
* [在类上附上@RunWith(Suite.class)](mailto:在类上附上@RunWith(Suite.class))注释
* 使用@Suite。SuiteClasser注释给Junit测试类加上引用

@RunWith(Suite.class)

@Suite.SuiteClasses({

TestJunit1.class,

TestJunit2.class

})

Public class JunitTestSuite{

}

### 时间测试

@Test(timeout=1000)

### 异常测试

@Test(expected = ArithmeticException.class)

### 参数化测试

* [用@RunWith(Parameterized.class)](mailto:用@RunWith(Parameterized.class))来注释test类
* 创建一个由@Parameters注释的公共的静态方法，它返回一个对象的集合作为测试数据集合
* 创建一个公共的构造函数，它接受和一行测试数据等同的东西
* 为每一列测试数据创建一个实例变量
* 用实例对象为测试数据的来源来创建你的测试用例

# JDBC

JDBC（Java Data Base Connectivity,java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。JDBC提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序。

## 常用接口

### Driver

Driver接口由数据库厂家提供，作为java开发人员，只需要使用Driver接口就可以了。在编程中要连接数据库，必须先装载特定厂商的数据库驱动程序，不同的数据库有不同的装载方法。

### DriverManager

用于管理JDBC驱动的服务类，程序中使用该类主要功能是获取Connection对象。

Public static synchronized Connection getConnetcion(String url, String user, String pass) throws SQLException.

### Connection

Connection与特定数据库的连接（会话），在连接上下文中执行sql语句并返回结果。

常用方法：

createStatement()：创建向数据库发送sql的statement对象。

prepareStatement(sql) ：创建向数据库发送预编译sql的PrepareSatement对象。会将SQL语句提交到数据库进行预编译。需要为预编译的SQL语句传入参数，比Statement多了如下方法void setXXX(int paramenterIndex, XXX value)

prepareCall(sql)：创建执行存储过程的callableStatement对象。

setAutoCommit(boolean autoCommit)：设置事务是否自动提交。

commit() ：在链接上提交事务。

rollback() ：在此链接上回滚事务。

Savepoint setsavepoint(String name)：创建一个保存点

Void setTransactionIsolation(int level):设置事务的隔离级别

### Statement

* ResultSet executeQuery(String sql) throws SQLException
* Int executeUpdate(String sql) throws SQLException
* Boolean execute(String sql) throws SQLException
* closeOnCompletion(),当所有依赖于statement的resultset关闭时，该statement会自动关闭。
* isCloseOnCompletion()，用于判断Statement是否打开了CloseOnCompletion
* executeLargeUpdate(),DML语句影响超过Integer.MAX\_VALUE时使用

### ResultSet

* Void close():释放
* Boolean absolute(int row):结果集指针移到row行，若为负数，则倒数
* Void beforeFirst():初始化
* Boolean first()
* Boolean previous()
* Boolean next()
* Boolean last()
* Void afterLast()

## JDBC编程步骤

### 加载数据库驱动

Class.forName(driverClass)

### 获取数据库连接

DriverManager.getConnection(String url, String user,String pass)

URL遵循如下写法,jdbc：subprotocol：other stuff

Jdbc:mysql://hostname:port/databasename

### 创建Statement对象

通过Connection对象创建Statement

1. createStatement()
2. prepareStatement(String sql)
3. prepareCall(String sql)

### 执行SQL

使用Statement执行SQL

1. execute()-返回boolean，使用getResultSet()、getUpdateCount()
2. executeUpdate()
3. executeQuery()

### 操作结果集

ResultSet

### 回收数据库资源

关闭ResultSet、Statement、Connection

## Statement与PreparedStatement对比

* PreparedStatement预编译SQL，性能更好
* PreparedStatement无须拼接SQL，编程更简单
* PreparedStatement可以防止SQL注入，安全性更好

## 结果集

默认方式打开的ResultSet是不可更新的，如希望更新，可在创建Statement或PrepareStatement加入额外参数。

* ResultSetType：Result.TYPE\_FORWARD\_ONLY：指针只能向前移动；

ResultSet.TYPE\_SCROLL\_INSENSITIVE:自由，底层数据不影响ResultSet内容

ResultSet.TYPE\_SCROLL\_SENSITIVE: 自由，底层数据影响ResultSet内容

* ResultSetConcurrency：控制并发类型

ResultSet.CONCUR\_READ\_ONLY:只读的并发模式

ResultSet.CONCUR\_UPDATEBLE:可更新的并发模式

Pstmt = conn.prepareStatement(sql, ResultSet.TYPE\_SCROLL\_INSENSITVIE, ResultSet.CONCUR\_UPDATABLE)

若定义为可更新结果集，需要满足

1. 所有数据集必须来自一个表
2. 选出数据集必须包含主键列

可使用ResultSet的upateXxx(int columnIndex,Xxx value)修改，最后通过调用ResultSet的updateRow()提交修改

使用结果集更新、删除、新增记录。Rs.insertRow()，Rs.updateRow()，Rs.deleteRow()

## Blob类型数据

Blob（binary long Object）常用于存储大文件，图片，声音等。储存Blob数据需要使用PreparedStatement，该对象有一个方法setBinaryStream(int paremeterIndex,INputStream x)

## 示例

### 连接MySql

|-Class.forName(“com.mysql.jdbc.Driver”)

|-Connection con = DriverManager.getConnection(“jdbc:mysql://localhost:8080/datename,user,password”);

|-

## Batch Insert

String [] queries = {

"insert into employee (name, city, phone) values ('A', 'X', '123')",

"insert into employee (name, city, phone) values ('B', 'Y', '234')",

"insert into employee (name, city, phone) values ('C', 'Z', '345')",

};

Connection connection = new getConnection();

Statement statement = connection.createStatement();

For(String query: queries){

Statement.addBatch(query);

}

Statement.executeBatch();

Statement.close();

Connection.close();

有可能导致数据库注入或者OutOfMemory

String sql = "insert into employee (name, city, phone) values (?, ?, ?)";

Connection connection = new getConnection();

PreparedStatement ps = connection.prepareStatement(sql);

final int batchSize = 1000;

int count = 0;

for (Employee employee: employees) {

ps.setString(1, employee.getName());

ps.setString(2, employee.getCity());

ps.setString(3, employee.getPhone());

ps.addBatch();

if(++count % batchSize == 0) {

ps.executeBatch();

ps.commit();

}

}

ps.executeBatch(); // insert remaining records

ps.close();

connection.close();

catch (DataTruncation ex) {

getLogger(FNM01I01LS2000Dao.class.getName()).log(Level.SEVERE, null, ex);

}

catch (SQLException | ArrayIndexOutOfBoundsException ex) {

getLogger(FNM01I01LS2000Dao.class.getName()).log(Level.SEVERE, null, ex);

}

## 事务处理

ACID(Atomicity,Consistency,Isolation,Durability)

### MySql对事务的支持

* SET AUTOCOMMIT = 0，取消自动提交处理，开启事务处理
* SET AUTOCOMMIT = 1，打开自动处理，关闭事务处理
* START TRANSACTION，启动事务
* BEGIN，启动事务，相当于执行START TRANSACTION
* COMMIT，提交事务
* ROLLBACK，回滚全部操作
* SAVEPOINT pointName，设置事务保存点
* ROLLBACK TO SAVEPOINT pointName，回滚操作到保存点

## 使用元数据分析数据库

### DatebaseMetaData

可以得到数据库的一些基本信息，比如数据库的名称、版本

* String getDatabaseProdctName() throws SQLException,数据库名称
* Int getDriverMajorVersion(),主版本号
* Int getDriverMinorVersion(),次版本号
* ResultSet getPrimaryKeys(String catalog, String schema, String table)throws SQLException
  + TABLE\_CAT String:表类别
  + TABLE\_SCHEM String:表模式
  + TABLE\_NAME String:表名称
  + COLUMN\_NAME String:列名称
  + KEY\_SEQ short:主键中的序列号
  + PK\_NAME String:主键的名称

### ResultSetMetaData

* Int getColumnCount() throws SQLException
* Boolean isAUtoIncrement(int column) throws SQLException:判断指定列是否是自动编号
* String getColumnName(int column) throws SQLException:返回列的名称

# Eclipse

## 环境配置

### 本地需要Groovy环境

<http://www.gradle.org/downloads>

下载后解压，配置GRADLE\_HOME环境变量。

### Groovy安装

ERROR：GroovyDynamicElementReader cannot be resolved to a type

[https://github.com/groovy/groovy-eclipse/wiki](https://github.com/groovy/groovy-eclipse/wiki" \t "_blank)查找相对应的eclipse版本

### 集成GIT

|-地址：<http://download.eclipse.org/egit/updates>

|-Check If existed Egit



|-Build local repository

|-Method 1



|-Method 2



|-Commit to local repository，Team-commit



|-Team Remote Push



eclipse的egit跟TortoiseGit使用的密钥是不共用的，需要再次添加



### Aptana3插件

用来编辑html文件。

<http://download.appcelerator.com/aptana/studio3/plugin/update/stable/>

使用如下网址，支持html

<http://download.eclipse.org/releases/juno>

## Clean

### 功效

eclipse为了提高效率，并不是每次启动项目都会检查插件，通过clean就是强制eclipse去检查已安装插件。

我们都知道.java文件是通过编译成.class文件运行的，而clean后会删除已经编译生成的.class文件并重新部署项目。

总起来将就是强制检查已安装插件，清除以前编译的信息，重新部署项目。

### clean后无法正常运行程序

解决方案：

Properties->Java Compiler->Building->Build Path Problems->Abort Build when Build Path Errors Occur，把这个选项勾选掉就可以运行程序了。

## 快捷键

1. ctrl+shift+r：打开资源
2. ctrl+o：快速outline
3. ctrl+e：快速转换编辑器
4. ctrl+page down或ctrl+page up：可以浏览前后的选项卡
5. ctrl+2，L：为本地变量赋值
6. alt+shift+r：重命名
7. alt+shift+l以及alt+shift+m：提取本地变量及方法
8. shift+enter及ctrl+shift+enter
9. Alt+方向键(上下)：移动
10. Alt+方向键(左右)：导航历史中跳转
11. Control-Q: 回到最后一次编辑的地方
12. ctrl+m
13. ctrl+.及ctrl+1：下一个错误及快速修改
14. F3: 打开申明
15. Alt+CTL+方向键(上下)：复制
16. Alt+Shift+J:添加文档解释
17. CTL+H：查找
18. Ctrl+shift+F6:编辑页之间切换
19. Ctrl+shift+F8:java窗口和debug调试窗口切换快捷键
20. Ctrl+shift+L：Quick text search
21. Ctrl+shift+O：自动导入包
22. Ctrl+shift+F：格式化代码
23. Atl+shift+J：添加文档注释

## Debug模式

全局 单步返回 F7 退出当前的函数

全局 单步跳过 F6 一行一行执行

全局 单步跳入 F5 进入函数内部

全局 单步跳入选择 Ctrl+F5

全局 调试上次启动 F11

全局 继续 F8

全局 使用过滤器单步执行 Shift+F5

全局 添加/去除断点 Ctrl+Shift+B

全局 显示 Ctrl+D

全局 运行上次启动 Ctrl+F11

全局 运行至行 Ctrl+R

全局 执行 Ctrl+U

### NullPointerException

NullPointerException是java.lang.NullPointerException的简称，是Java语言中的一个异常类，位于java.lang包中，父类是java.lang.RuntimeException，该异常在源程序中可以不进行捕获和处理。

当应用程序试图在需要对象的地方使用 null 时，抛出该异常。这种情况包括：

* 调用 null 对象的实例方法。
* 访问或修改 null 对象的字段。
* 如果一个数组为null，试图用属性length获得其长度时。
* 如果一个数组为null，试图访问或修改其中某个元素时。
* 在需要抛出一个异常对象，而该对象为 null 时。
* 应用程序将会抛出NullPointerException类的实例，表明其他对 null 对象的非法使用。

### SlowQueryReport

Tomcat

## 构建Spring-Boot

### 安装Spring Tool Suite



检查是否安装成功。

### 全量构造

|- mvnw -P snapshot,prepare install –DskipTests

|- mvnw -s ./settings.xml -f spring-boot-full-build -P full clean install

### 使用spring boot initialize快速构造

<https://start.spring.io/>



导入即可

### 使用STS工具快速构建



### 新建Maven配置

|-New Maven project



|-choose webApp



|-modify pom.xml



|-update project



|-choose and apply



|-Run



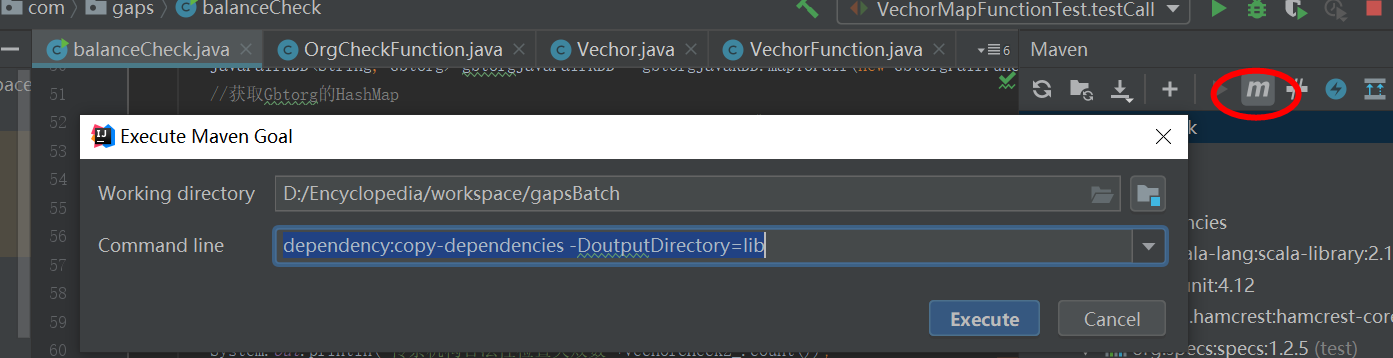
# IntelliJ

## 快捷键

## 打包

### 本地仓库导出所有依赖的jar包

在maven中使用命令：dependency:copy-dependencies -DoutputDirectory=lib。



若出现Non-parseable settings报错，则将setting.xml的格式更改为UTF-8

# Maven

## Maven简介

Apache Maven是一套软件工程管理和整合工具。基于工程对象模型POM概念，通过一个中央信息管理模块，Maven能够管理项目的构建、报告和文档。

是一个项目管理和整合工具，能帮助开发者完成：

|-构建

|-文档生成

|-报告

|-依赖

|-SCMs

|-发布

|-分发

|-邮件列表

工程被定义在一个xml文件中，pom.xml，Project Object Model（POM）。Maven使用约定而不是配置，开发者不再需要自己构建过程。只需要合理放置文件，而不需要再Pom.xml中定义任何配置。

| **配置项** | **默认值** |
| --- | --- |
| source code | ${basedir}/src/main/java |
| resources | ${basedir}/src/main/resources |
| Tests | ${basedir}/src/test |
| Complied byte code | ${basedir}/target |
| distributable JAR | ${basedir}/target/classes |

## 配置

### 下载

<http://maven.apache.org/download.html>

### 设置环境变量

1. 使用系统属性设置环境变量。  
   |-M2\_HOME=D:\SoftConfig\apache-maven-3.5.3\  
   |-M2=%M2\_HOME%\bin  
   |-MAVEN\_OPTS=-Xms256m -Xmx512m
2. 将%M2%添加到系统“Path”变量末尾

### 检查是否安装成功



### 使用阿里云仓库

Maven中央资源库： <http://search.maven.org/>

在settings.xml中加入

<mirror>

<id>nexus-aliyun</id>

<mirrorOf>central</mirrorOf>

<name>Nexus aliyun</name>

<url>http://maven.aliyun.com/nexus/content/groups/public</url>

</mirror>



### 更新Maven本地库



### Maven依赖机制

<dependencies>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.14</version>

</dependency>

</dependencies>

构建依赖自动下载，若未指定version，则有新版时会自动升级

### 定制库到Maven本地资源库

Kaptcha，流行的第三方JAVA库，被应用来生成验证码图片，用来阻止垃圾文件。其不在Maven中央仓库中。将其安装到Maven本地资源库中。

|- Kaptcha下载：<http://www.softpedia.com/get/Programming/Other-Programming-Files/Kaptcha.shtml>

|-执行命令：mvn install:install-file -Dfile=D:\SoftConfig\kaptcha-2.3-jdk14.jar -DgroupId=com.google.code -DartifactId=kaptcha -Dversion={2.3} -Dpackaging=jar

|-POM中加入依赖关系：

<dependency>

<groupId>com.google.code</groupId>

<artifactId>kaptcha</artifactId>

<version>2.3</version>

</dependency>

### 从Maven模板创建项目

mvn archetype:generate -DgroupId={project-packaging} -DartifactId={project-name}-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

这告诉 Maven 来从 maven-archetype-quickstart 模板创建 Java 项目。如果忽视 archetypeArtifactId 选项，一个巨大的 Maven 模板列表将列出。

创建JAR项目

mvn archetype:generate -DgroupId=com.ldc -DartifactId=NumberGentor -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

若需要创建Web项目（WAR）

mvn archetype:generate -DgroupId=com.ldc -DartifactId=CounterWebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false

### Maven目录布局

<http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>



### Eclipse IDE

为了使它成为一个 Eclipse 项目，在终端进入到 “NumberGenerator” 项目，键入以下命令：mvn eclipse:eclipse，生成.classpath，.project两个新文件，生成JAR

要导入项目到Eclipse IDE中，选择 “File -> Import… -> General->Existing Projects into Workspace”



若为WEB项目，需要使用mvn eclipse:eclipse -Dwtpversion=2.0，选项 -Dwtpversion=2.0 告诉 Maven 将项目转换到 Eclipse 的 Web 项目(WAR)，而不是默认的Java项目(JAR)。

可配置pom.xml来避免每次构建都加上参数，配置maven-eclipse-plugin

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-eclipse-plugin</artifactId>

<version>2.9</version>

<configuration>

<!-- Always download and attach dependencies source code -->

<downloadSources>true</downloadSources>

<downloadJavadocs>false</downloadJavadocs>

<!-- Avoid type mvn eclipse:eclipse -Dwtpversion=2.0 -->

<wtpversion>2.0</wtpversion>

</configuration>

</plugin>

此时可使用mvn eclipse:eclipse，构建WAR

### 更新POM

默认的 pom.xml 太简单了，很多时候，你需要添加编译器插件来告诉 Maven 使用哪个 JDK 版本是用来编译项目。

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>2.3.2</version>

<configuration>

<source>1.6</source>

<target>1.6</target>

</configuration>

</plugin>

</plugins>

</build>从更新JUnit 3.8.1到最新的 4.11。



在终端，再次发出同样的命令 mvn eclipse:eclipse ,Maven将从Maven中心储存库下载插件项目依赖关系（JUnit），它会自动保存到你的本地仓库。

### 将Maven变量加入Eclipse



此处无法修改。



由下列路径的xml定义



修改为本地的Maven路径



### 项目打包

在项目路径下，mvn package，最终在target路径下生成jar



### 运行Jar文件

java -cp NumberGentor-1.0-SNAPSHOT.jar com.ldc



## 创建WEB项目（WAR）

|- mvn archetype:generate -DgroupId=com.ldc -DartifactId=CounterWebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false

|- mvn eclipse:eclipse -Dwtpversion=2.0

|-更改pom.xml文件



|-手工创建文件夹

|-mkdir java\com\ldc\controller

|-type nul>.test（生成空文件）

|-echo fileContent>a.txt（生成非空文件）

|-生成BaseController.java文件



|-创建Spring配置文件mvc-dispatcher-servlet.xml



|-修改web.xml文件



|-移动文件src\main\webapp\index.jsp到WEB-INF/pages下

|-新建type nul>logback.xml，内容为<?xml version="1.0" encoding="UTF-8"?>

|-mvn package



Sol：添加JAVA\_HOME环境变量，使用jdk的路径，生成如下



|- 如果想通过 Eclipse 服务器这个项目插件(Tomcat 或其它容器)调试，这里再输入，如果一切顺利，该项目的依赖将被装配附加到 Web部署项目



|-查看POM.xml的tomcat依赖



|-mvn tomcat:run：启动tomcat，部署默认端口在8080

## 使用模板来创建项目

### mvn archetype:generate

输出在控制台，无法查看，需要使用mvn archetype:generate > templates.txt //waiting few seconds,then exits,CTL+C退出。

使用该插件创建maven目录，如：

Src-main-java-package

-test-java-package

-resources

### 默认模式

如果知道那个archetypeArtifactld使用，可跳过交互模式

|-maven-archetype-quickstart (Java Project)

mvn archetype:generate -DgroupId=com.ldc.core -DartifactId=ProjectName -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

|-maven-archetype-webapp (Java Web Project)

mvn archetype:generate -DgroupId=com.ldc.web -DartifactId=ProjectName -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false POM

### 定义

需要说明的是每个工程应该只有一个 POM 文件。

所有的 POM 文件需要 project 元素和三个必须的字段：groupId, artifactId,version。

在仓库中的工程标识为 groupId:artifactId:version

POM.xml 的根元素是 project，它有三个主要的子节点：

| 节点 | 描述 |
| --- | --- |
| groupId | 这是工程组的标识。它在一个组织或者项目中通常是唯一的。例如，一个银行组织 com.company.bank 拥有所有的和银行相关的项目。 |
| artifactId | 这是工程的标识。它通常是工程的名称。例如，消费者银行。groupId 和 artifactId 一起定义了 artifact 在仓库中的位置。 |
| version | 这是工程的版本号。在 artifact 的仓库中，它用来区分不同的版本。例如： com.company.bank:consumer-banking:1.0 com.company.bank:consumer-banking:1.1. |



## 清理项目

### 获取最新部署

在基于Maven的项目中，很多缓存输出在“target”文件夹中。如果想建立项目部署，必须确保清理所有缓存的输出，从面能够随时获得最新的部署。

Mvn clean

### 部署进行生产

要部署您的项目进行生产，它总是建议使用 “mvn clean package“, 以确保始终获得最新的部署。

## 单元测试

Mvn test

运行单个测试

mvn -Dtest=TestApp2 test

## 项目资源安装到本地资源库

Mvn install，自动部署到本地资源库

Mvn clean install，部署最新的资源

## 生成基于Maven的项目文档站点

在Maven中，可以使用“mvn site”，为您的项目信息生成文档站点。

## 部署站点

### Apache配置WebDAV

基于Web的分布式创作和版本控制(WebDAV)允许用户读取，并通过HTTP服务器写入文件夹和文件。

|-启动WebDAV

|-在pom.xml中部署站点

<distributionManagement>

<site>

<id>liudingchaoServer</id>

<url>dav:http://127.0.0.1/sites/</url>

</site>

</distributionManagement>

|-告诉Maven使用“wagon-webdav-jackrabbit”扩展部署

<build>

<extensions>

<extension>

<groupId>org.apache.maven.wagon</groupId>

<artifactId>wagon-webdav-jackrabbit</artifactId>

<version>1.0-beta-7</version>

</extension>

</extensions>

</build>

### 配置WebDAV身份验证

|-MAVEN安装路径下的D:\SoftConfig\apache-maven-3.5.3\conf\setting.xml加入

<servers>

<server>

<id>liudingchaoServer</id>

<username>admin</username>

<password>199010281</password>

</server>

</servers>

### 部署

Mvn site:deploy



## 部署基于Maven的war文件到Tomcat

### 指令

Tomcat 7

发布URL = http://localhost:8080/manager/text

命令 = mvn tomcat7:deploy

Tomcat 6

发布 URL = http://localhost:8080/manager/

命令 = mvn tomcat6:deploy

### Tomcat认证

|-添加具有角色管理器GUI和管理脚本的用户



<role rolename="manager-gui"/>

<role rolename="manager-script"/>

<user username="admin" password="123456" roles="manager-gui,manager-script" />

|-Maven认证



|-Tomcat7 Maven插件

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<url>http://localhost:8080/manager/text</url>

<server>TomcatServer</server>

<path> /CounterWebApp</path>

</configuration>

</plugin>

### 发布到Tomcat

操作TomcatWAR文件

mvn tomcat7:deploy

mvn tomcat7:undeploy

mvn tomcat7:redeploy

## Eclipse创建一个MavenWeb项目

|-创建Maven



|-select a simple project

|-config



|-rwrite pom.xml, 添加servlet依赖项和Tomcat maven插件



|-check struct of maven web project

|-write a helloworld java

|-mvn clean install

|-If U wanna invoke maven command in eclipse IDE,U should config as



|-click run

|- <http://localhost:8080/hello>

## 其他指令

|-mvn dependency:tree 查看依赖树

# Apache服务器

## 简介

Apache HTTP Server（简称Apache）是Apache软件基金会的一个开放源码的网页服务器，是世界使用排名第一的Web服务器软件，可以在大多数计算机操作系统中运行，由于其多平台和安全性被广泛使用，是最流行的Web服务器端软件之一。

## 搭建

### 下载

<http://httpd.apache.org/download.cgi>







httpd-2.4.33-o102o-x86-vc14-r2.zip

### 安装Apache服务器



|-需要修改apache24文件夹为apache，否则会出现如上报错

|-进入apache24\bin

|-CMD运行httpd -k install



|-修改Apache\conf\httpd.conf





## 操作

### 启动

|-httpd -k start

|-找到bin下ApacheMonitor.exe文件，选择“start”

|-点击”开始“菜单，选择”运行“，输入”services.msc“后回车，在“服务”窗口中找到“Apache2.4”，选择“启动



### 测试Apache服务器

<http://localhost/>



## 常用指令

Stop Apache httpd -k stop

Restart Apache httpd -k restart

Uninstall Apache Service httpd -k uninstall

Test Config Syntax httpd -t

Version Details httpd -V

Command Line Options List httpd –h

## 配置WebDAV

### 修改httpd.conf









### 创建用户密码

htdigest -c "D:\SoftConfig\Apache\user.password" DAV-upload admin



生成如下文件

### 配置WebDAV文件夹

|-新建文件夹



|-新增如下语句

<Directory "D:\SoftConfig\Apache\sites">

Dav On

Order Allow,Deny

Allow from all

AuthType Digest

AuthName DAV-upload

Options Indexes

AuthUserFile "D:/Program Files/apache-server/Apache24/user.passwd"

AuthDigestProvider file

# Allow universal read-access, but writes are restricted

# to the admin user.

<LimitExcept GET OPTIONS>

require user admin

</LimitExcept>

</Directory>

#



以上语句说明启动WebDAV访问sites文件夹。

访问的时候用user.passwd文件进行身份验证。

AuthName是DAV-upload的





Apache2.2版本用上列语句表示全部权限，2.4需要使用Require all granted

上面的语句需要改写为

|  |
| --- |
| DavLockDB "D:/SoftConfig/Apache/DavLock"  Alias /sites "D:/SoftConfig/Apache/sites"  <Directory "D:/SoftConfig/Apache/sites">  Dav On  Require all granted  AuthType Digest  AuthName DAV-upload  Options Indexes  AuthUserFile "D:\SoftConfig\Apache\user.passwd"  AuthDigestProvider file  # Allow universal read-access, but writes are restricted  # to the admin user.  <LimitExcept GET OPTIONS>  require user admin  </LimitExcept>  </Directory> |

修改完之后运行httpd –k start成功

### 使用WebDAV客户端验证

|- <http://www.bitkinex.com/>

## Tomcat服务器

Apache是普通服务器，本身只支持html，可通过插件支持php,还可以与Tomcat连通(单向Apache连接Tomcat,就是说通过Apache可以访问Tomcat资源。反之不然)。Apache只支持静态网页，但像php,cgi,jsp等动态网页就需要Tomcat来处理。

### 安装

<https://tomcat.apache.org/>



# JAVA

## 注释

/\*\*

\*Description

\*/文档注释，会被提取到API文档中

如果使用javadoc标记

@author

@version

@deprecated:不推荐使用的方法

@param

@return

@see:参见

@exception

@throws

Javadoc默认不会提取@author、@version，主要指定-author\-version

三种，单行注释，多行注释，文档注释

## 数据类型

### 直接量的类型

Int:二进制0B、0b开头；八进制：0开头；十六进制0x、0X

Long：3L，0x12L

Float：5.34F，3.14E5f

Double:5.34,3.14E5

Boolean：true，false

Char：’a’,’\n’,’\u0061’

String：“”

Null

正无穷：Double，Float类的POSITIVE\_INFINITY

负无穷：Double，Float类的NEGATIVE\_INFINITY

非数：NaN，不与任何数相等，甚至NaN

65->’A’,97->’a’,48->’0’

127,32767,2147483647

数值可以使用下划线double pi = 3.14\_159\_23\_3

### 算术运算符

+\-\\*\/\++\--

Math

### 赋值运算符

=

### 位运算符

&、|、~、^、<<、>>、>>>

### 比较运算符

>，<,>=，<=,==,!=

### 逻辑运算符

&&\||\!\^\&\|

&与|，不短路的与和或

### 三目运算符

?:

## 流程控制

SWITCH，判断条件成立，则顺序执行，不再判断，因此需要加break

Break可以跳出外围循环，需要加上tag，如outloop:，tag只有放在循环语句之前才有用

## 数组

不要同时指定长度和分配初始值。

数组需要初始化后才能使用，初始化即分配内存空间，并为每个数组元素指定初始值。

* 静态初始化，程序员指定元素，系统决定系统长度
* 动态初始化，程序员指定长度，系统分配初始值

两种不要混合使用。

整数型：int,long,short,byte，初始值为0

浮点型：float,double，初始值为0.0

字符型：char，初始值为\u0000

布尔型：boolean，初始值false

引用类型：接口、数组、类，初始值null

### 静态初始化

Int [] arr = new int[]{1,2,3};

Int [] arr ={1,2,3};//简化版

### 动态初始化

Int[] arr = new int[5];

### Foreach

For（type var: array）

### 初始化

初始化多维数组的时候需要指定最左边维的大小

### Array

二分查找需要数组按照升序排列

* Int binarySearch（type[] a,type key）
* Int binarySearch(type[] a,int fromIndex,int toIndex,type key)
* Type[] copyOf(type[] original,int length)
* Type[] copyOfRange(type[] original,int from,int to)
* Boolean equals(type[] a,type[] a2)
* Void fill(type[] a,type val)
* Void fill(type[]a, int fromIndex,int toINdex,type val)
* Void sort(type[] a)
* Void sort(type[] a,int fromIndex,int toIndex)
* String toString(type[] a):顺序连接，多个数组元素使用英文逗号，和空格隔开

JAVA 8增强了Arrays类的功能

* Void parallerPrefix(xxx[] array,XxxBinaryOperater op):计算结果作为新元素，op计算公式包括left、right两个形参，当计算第一数组元素时，left的值默认为1
* Void parallelPrefix(xxx[] array,int fromIndex,int toIndex,XxxBinaryOperator op)
* Void setAll(xxx[] array,IntToXxxfunction generator)
* Void parallerseAll(xxx[] array,IntToXxxfunction generator)
* Void parallerSort(xxx[] a)
* Void parallerlSort(xxx[] a)
* Spliterator.OfXxx spliterator(xxx[] array):将数组的所有的元素都转换成对应的Spliterator对象
* Spliterator.OfXxx split erator(xxx[] array,int startInclusive,int endExclusive)
* XxxStream stream(xxx[] array)
* XxxStream stream(xxx[] array,int startInclusive,int endExclusive)

把char型数字转换成int型数字，因为他们的ASCII码值恰好相差48

Int num = numstr.charAt(i) - 48

## 方法

* 方法不能独立定义，方法只能在类体内定义
* 从逻辑意义上来看，方法要么属于类，要么属于内的对象
* 永远不能独立执行方法，执行方法必须使用类或者对象，this

Java中方法都是按值传递。

定义形参可变的方法，public static void test(int v1,string … v2)

如果需要调用多形参的方法，且只有一个参数，可使用

Olv.test(new String[]{“aa”})

## 对象和内存控制

This总是指向调用该对象的对象。有两种情况：

* 构造器中引 用该构造器正在初始化的对象
* 在方法中引用调用该方法的对象

新建一个对象，person p1 = new person();

首先会在堆中生成person类的空间，初始化static变量，后面才是分配对象

内存管理分为：内存分配和内存回收。

局部变量：形参、方法内的局部变量、代码块内的局部变量，储存在栈中

成员变量：使用static修饰：静态变量或类变量

否则为非静态变量或实例变量

非法前向引用

Static int num1 = num2 + 2；

Static int num2 = 20;

非法前向引用

Int num1 = num + 2；

Int num2 = 20;

一个实例变量，一个类变量，可以

Int num1 = num2 + 2;

Static int num2 = 20 ;

### 实例

三种初始化：

1. 定义实例变量时指定初始值；
2. 非静态初始块中对实例变量指定初始值；
3. 构造器

1、2比3早，1,2的顺序按照他们在源程序中顺序

Javap –c x.class反编译

### 类变量

初始化

1. 定义实例变量时指定初始值；
2. 静态初始块中对实例变量指定初始值；

### 父类构造器

当调用某个类的构造器创建Java对象时，系统会优先调用父类的非静态初始块，再调用父类的构造器

This代表正在初始化的JAVA对象。

当变量的编译和运行的类型不同时，通过该变量访问它引用的对象的实例对象时，该实例对象的值由申明该变量的类型决定；

通过该变量调用它引用的对象的实例方法时，该方法的行为由它实际所引用的对象来决定。

### 父子实例的内存控制

针对成员方法，编译器会将父类的方法直接转移到子类中，因此若子类重写，则会完全覆盖父类的函数。而针对成员变量，则不会出现该现象。两个变量依然存在。

Java程序允许某个方法通过return this返回该方法的JAVA对象，但不允许直接return super，甚至不允许直接将super当成一个引用变量使用。

### Final修饰符

* Final可修饰变量，被赋初始值之后，不能对它重新赋值
* Final可修饰方法，被final修饰的方法不能被重写
* Final可修饰类，不能派生类

Final修饰的变量必须现实指定初始值，而只能在如下三个位置指定

1. 定义final实例对象的时候指定初始值
2. 在飞静态初始化块中
3. 构造器中

Final修饰的类变量，同样必须显示指定

1. 定义final类变量时
2. 静态初始化块中

Final可用于宏替换

### 字符串缓冲池

Java会缓冲曾经用过的字符串。

局部内部内才可以访问局部变量，并且变量必须要用final修饰。普通局部变量而言，它的作用域就是停留在该方法内，当方法执行结束，局部变量也随之小时；但内部类则可能产生隐式闭包。

## 代码块

执行顺序

|-静态代码块

|-构造块

|-构造方法

### This

1. 调用本类的构造方法
2. this表示当前对象
3. 表示类中的属性

## 封装Encapsulation

Private(当前类)->default(当前包)->protected(子类)->public(公共)

Import package.sub.\*;\*代表类不能代表包

Java默认为所有的源文件导入java.lang包下的所有类，因此String，System无需导入

使用import可以省略写包名。而使用import static（静态导入）可以连类名都可以省略

## Java的常用包

Java.lang:这个包下包含了Java语言的核心类，如String、Math、System和Thread类等，该包自动导入

Java.util：这个包包含有Java的大量工具类，例如Arrays和List、Set

Java.net:包含Java网络编程相关

Java.io:包含了一些Java输入/输出的编程相关的类/接口

Java.text：包含一些Java格式化相关的类

Java.sql:包含Java进行JDBC数据库编程相关的

Java.awt：包含了抽象窗口工具集的相关类/接口，用于构建GUI

Java.swing:包下包含了Swing图形用户界面编程，可用于构建平台无关的GUI

java.lang.reflect：反射机制的包

## 多态Polymorphism

Instanceof运算符的前一个操作数通常是一个引用类型变量，后面一个操作数通常是一个类。用于判断前面的对象是否是后面的类，后者其子类、实现类的实例。Instanceof运算符前面的操作数要么与后面的类相同，要么与后面的类具有父子继承关系，否则会引起编译错误。

## Java增强包

Int型自动装箱，128以下会相等，超过则不等。Inetger有一个cache数组，-128~127，若超过则会重新新建一个Integer实例。

==号对于引用对象，只有它们指向同一个对象时，==判断才返回true。==不可用于比较类型上没有父子关系的两个对象。

Hello直接量和new String(“hello”)，JVM会使用常量池管理这些字符串，当调用new String（“hello”）,JVM会先使用常量池来保存“hello”直接量，再调用String类的构造器来创建一个新的String对象，会被保存在堆内存中，会产生两个字符串对象。

JVM常量池保证相同的字符串直接量只有一个，不会产生多个副本。

## Equals

正确重写equals()方法应该满足一下条件：

* 自反性
* 对称性
* 传递性
* 一致性
* 对任何不是null的x，x.equals(null)一定返回false

## 单例类（Singleton）

如果一个类始终只能创建一个实例，这个类成为单例类。将构造器用private修饰。一旦把构造器隐藏起来，就需要提供一个public方法作为该类的访问点，用于创建该类的对象，且该方法必须使用static修饰。此外，还需要缓存已创建的对象，因此需要为该类使用一个成员变量保存，需用static修饰。

## 不可变类

Java提供的8个包装类和java.lang.String类都是不可变类。创建实例后，其实例变量不可变。

* 使用private和final修饰符来修饰该类的成员变量
* 提供带参数构造器，用于根据传入的参数来初始化成员变量
* 仅为该类的成员变量提供getter方法，不要为该类的成员变量提供setter方法
* 如果有必要，重写Object类的hashcode()和equals()方法

设计的时候需要考虑到如果是引用型成员变量，需要采取一些措施实现不可变类。如改写设置引用变量的方法，也改变此变量的getter方法。

### 缓存实例的不可变类

使用数组实现缓存

## 抽象类

抽象类和方法必须使用abstract修饰符来定义，有抽象方法的类只能被定义成抽象类，抽象类里没有抽象方法。

* 抽象类和方法必须使用abstract修饰，抽象方法不能有方法体
* 抽象类不能被实例化，无法使用new来调用构造函数
* 抽象类可以包括成员变量、方法（普通和抽象）、构造器、初始化块、内部类（接口、枚举）

Final和abstract不能同时使用。

Static和abstract不能同时修饰某个方法。

## 内部类

内部类作用：

* 内部类提供了更好的封装，可以把内部类隐藏在外部类之内，不允许同一个包中的其他类访问。
* 内部类可以访问外部类的私有数据。
* 匿名内部类适合用于创建那些仅需要一次使用的类
* 内部类比外部类可以多使用三个修饰符：private、protected、static
* 非静态内部类不能拥有静态成员

### 静态内部类

归属于类

### 局部内部类

把一个内部类放在方法中定义

### Java8改进的匿名内部类

匿名内部类继承一个父类或实现一个借口。匿名内部类定义时会直接创建一个实例后该类定义消失。

* 匿名内部类不能定义成抽象类
* 匿名内部类中不能定义构造函数
* 如果局部变量被匿名内部类访问，则自动变为final

New interfaceimp(){}

### Java8新增的Lambda表达式

Lambda表达式支持将代码块作为方法参数，Lambda表达式允许使用更简洁的代码创建只有一个抽象方法的接口。

(形参)->{代码块}

若代码块只有一行，可以省略花括号；若此时要求返回，可以省略rerun。还可以使用方法引用和构造器引用。

Lambda表达式的类型，也称为目标类型，必须是函数式接口。函数式接口代表只能包含一个抽象方法的接口。

JAVA8提供了@FunctionalInterface，告诉编译器检查该接口必须是函数式接口。

因为Lambda表达式接口被当成对象，可以用来赋值。

Runnable r = Lambda；Runnable是一个函数式接口。两者参数要一致。

* 将Lambda表达式赋值给函数式接口类型的变量
* 将Lambda表达式作为函数式接口类型的参数传给某个方法
* 使用函数式接口对Lambda进行强制类型转换（Runnable）（Lambda）

可以使用方法引用替换Lambda表达式，如：

1. Converter c1 = s->Integer.valueOf(s);
2. Converter c1 = Integer::vauleOf;

引用特定对象

1. Converter c2 = s->“fkit.irg”.valueOf(s);
2. Converter c2 = “fkit.irg”.valueOf;

引用某类对象的实例方法

Mytest mt = （a,b,c）-> a.substring(b,c)

Mytest mt = String::substring;

引用构造器

YourTest yt = （String a）->new JFrame(a);

YourTest yt = JFrame::new;

## Enum枚举类

* 枚举类可以实现一个或多个接，默认继承java.lang.Enum类，而不是继承Object。
* 使用enum定义、非抽象的枚举类默认为final，不能派生
* 枚举类的构造器只能使用private访问控制符
* 枚举类型的所有实例必须在枚举类的第一行显示列出，否则枚举类永远不能产生实例，列出这些实例，系统会自动添加public static final

枚举类默认提供了values()方法，可以遍历所有的枚举值。

Java.lang.Enum提供了几个方法

* Int compareTo(E o)
* String name(),返回名称
* Int ordinal()，返回枚举值在枚举类中的索引值
* String toString(),返回枚举常量的名称
* Public static<T extends Enum<T>> T valueOf(Class<T> enumType,String name)

枚举.values()，获取枚举类的数据

Enum实现了Comparable,Serializable接口

Clone()/cmpareTo()/equals()/hashCode()/name()/ordinal()/valuesOf()

## 对象与垃圾回收

垃圾回收之前都会调用finalize(),可使对象重新复活

对象在内存中的状态：

1. 可达状态
2. 可恢复状态
3. 不可达状态

强制垃圾回收

1. System.gc()
2. Runtime.getRuntime().gc()

运行时使用-verbose:gc，可以看到每次垃圾回收时提示消息。

Finalize()定义在Object类的实例方法

1. 永远不要主动调用某个对象的finalize()
2. Finalize()何时被调用具有不确定
3. 当JVM执行可恢复对象的finalize()方法时，变成可达
4. 当JVM执行finalize()出现异常，垃圾回收机制不会报告异常

## 对象的软、弱和虚引用

Java.lang.ref包下提供三个类，SoftReference、PhantomReference和WeakReference，分别表示软引用，虚引用和弱引用。

## JAR包

三种发布方式

1. 使用平台相关编译器将整个应用编译成平台相关的可执行文件。丧失跨平台性
2. 为应用编辑一个批处理文件。
3. 将一个程序制作成可执行JAR包，会映射成javaw.exe打开，使用jar cvfe test.jar test.Test test

## 常用类

### String

两种实例化

|-String str = “hello”;-保存在字符串池中

|-String str = new String(“hello”); -可使用intern()方法入池

常用方法

|-Sting(char[] value)\(char[] value,int offset,int count)\(byte[] value)

|-public char[] toCharArray()

|-public char charAt(int index)

|-public byte[] getByte()

|-length\indexOf\trim\substring\split\toUpperCase()\toLowerCase()

|-boolean startWith\endWith\equals\equalsIgnoreCase()

|-replaceAll(String regex, String replacement)

### System

Getnev()\getPropeties()\getProperty

|-public static void exit(int status)

|-publi static long currentTimeMillis()

|-public static void arraycopy(Object src,int srcPos,Object dest,int destPos,int length)

|-public static Properties getProperties()

|-public static String getProperty(String key)

System.getProperties().list(System.out)

垃圾对象的回收：

对象回收前都会调用finalize()方法

protected void finalize() throws Throwable

### Runtime

Runtime表示运行操作类，是一个封装了JVM进程的类。构造方法私有化，属于单例设计。

|-public long freeMemory()\maxMemory()

|-public void gc()

|-public Process exec(String command) throws IOException

Runtime rt = Runtime.getRuntime();

Rt.exec(“notepad.exe”)

可使用Process提供的destory关闭该任务

Runtime runtime = Runtime.*getRuntime*();

System.***out***.println(runtime.freeMemory());

Process process = **null**;

**try** {

process = runtime.exec("notepad.exe");

} **catch** (Exception e) {

// **TODO**: handle exception

}

**try** {

Thread.*sleep*(1000);

} **catch** (Exception e) {

// **TODO**: handle exception

}

process.destroy();

### Object

Equals(),finalized(),getClass(),hashCode();toString(),super.clone()

### String\StringBuffer\StringBuilder

后面两者可变，append()\insert()\reverse()\setCharAt()\setLength()

StringBuffer是线程安全的，而StringBuilder没有实现，因而性能略高

### Random/ThreadLocalRandom

Random rand = new Random(System.currentTimeMills());

nextBoolean\Float\Int\Long\Double\Int(int n)

### BigDecimal

构建时不要使用double来构建，会造成精度丢失

使用BigDecimal f1 = new Bigdecimal(“0.05”);

BigDecimal f2 = new BigDecimal.valuesOf(0.01);

### Date

|-date(),creat date object use current time

|-date(long date)

|-after()\before()\compareTo()

|-ouput with format:Data date = new Date();  
 String day = String.format(“%te”,date);

%te,一个月的某一天

%tb,指定语言的月份简称

%tB,全称

%tA,星期几全称

%ta，星期几简称

%tc，全部日期和时间信息

%tY，4位年份

%tj，一年的第几天

%tm：月份

%td：一个月的第几天

%ty：二位年份

%tH：hous in 24

%tM：minute

%tS：second

%tF：2018-05-06

%tD：05/06/18

%tr:11:01:01 pm

%tT:23:01:01

%tR:23:01

|-public void setTime(long time)

|-public long getTime()

### Calendar

|-public static final intYEAR\MONTH\DAY\_OF\_MONTH\MINUTE\SECOND\MILLSECOND

|-public static Calendar getInstance()

|-public Boolean after(Object when)

|-public Boolean before(Object when)

|-public int get(int field)

### DateFormat

DateFormat是一个抽象类

|-public static final DateFromat getDateInstance();//日期

|- public static final DateFromat getDateInstance(int style, Locale aLocale);//根据Locale得到

|-public static final DateFromat getDateTimeInstance();//日期时间

|-public static final DateFromat getDateTimeInstance(int dateStyle,int timeStyle,Locale aLocale)

eg:

df=DateFormat.getDateTimeInstanece(DateFormat.YEAR\_FIELD,DateFormat.ERA\_FIELD,

new Local(“zh”,”CN”))

### SimpleDateFormat

|-pubilc SimpleDateFormat(String pattern)

|-public Date parse(String source) throws PareseException

|-public final String format(Date date)

eg:(String->Date)

SimpleDateFormat sd = new SimpleDateFormat(“yyyy-MM-dd HH:mm:sss.SSS”);

Date d = sd.parse(“2018-5-10 16:45:17.123”);

### Math

|-sin(double a)\cos\tan\asin\acos\atan

|-toRadians(double angdeg):change to radian

|-toDegrees(double angrad)

|-exp\log\log10\sqrt\cbrt\pow

|-ceil(>=current smallest interger)\floor\rint(nearst,default even)\round

|-max\min\abs

|-random:0.00~1.00(Math.random())

|-Random:Random r = new Random(seedValue);  
 nextInt()\netInt(int n)(0~n)\nextLong\nextBoolean\nextFloat\nextDouble\nextGaussian

### DecimalFormat

Java.text.DecimalFormat

|-0,num,if exist show num else 0

|-#,num,if exist show num else show nothing

|-.\-\,\E

|-%

|-\u2030,乘以1000转换为千分数，前缀或者后缀

Eg：

1. DecimalFormat myFormat = new DecimalFormat(pattern):  
   String output = myFormat.format(value);
2. DecimalFormat myFormat = new DecimalFormat();  
   myFormat.applyPattern(pattern);  
   String output = myFormatformat(value);

### BegInteger

|-BigInteger(String val)

|-public BigInteger add(BigInteger val)\subtract\multiply\divide\max\min

|-public BigInteger[] divideAndRemainder(BigInteger bal)

### BigDecimal

|-BigDecimal(double/int/String val)

|-add\subtract\multiply\divide

RoundingMode中的舍入方式：

* ROUND\_UP：远离零方向舍入，向绝对值最大的方向，只要舍弃位非零即进位
* ROUND\_DOWN：向零方向靠拢，所有的位都舍弃
* ROUND\_CEILING：向正无穷方向舍入
* ROUND\_FLOOR：向负无穷方向舍入
* HALF\_UP：四舍五入（5进）
* HALF\_DOWN：四舍五入（5舍）
* HALF\_EVEN：银行家算法

### NumberFormat

|- public static Locale[] getAvailableLocale ();//返回所有语言环境的数组

|-public static final NumberFormat getInstance()；//当前默认语言环境的数字格式

|- public static NumberFormat getInstance(Locale inLocale);

|-public static final NumberFormat getCurrencyInstance()；//当前默认语言环境的货币格式

|- public static NumberFormat getCurrencyInstance(Locale inLocale);

### 对象克隆技术

protected Object clone() throws CloneNotSupportedException

必须要实现Cloneable接口，需要复写此方法，只能在方法中调用父类的clone()方法。

### Arrays类

|-public static Boolean equals(int[] a , int[] a2)

|-fill(int[] a,int val)\sort(int[] a)\binarySearch(int[] a, int key)\toString(int[] a)

### Comparable接口

需要实现Comparable接口，复写compareTo方法，才能使用java.util.Arrays.sort(对象数组)进行排序，若没有实现则会报错，ClassCaseException，因为在排序时，所有对象都向Comparable进行转换。

实现二叉树：

**class** BinaryTree{

**class** Node{

**private** Comparable data;

**private** Node left;

**private** Node right;

**public** Node(Comparable data) {

**this**.data = data;

}

**public** **void** addNode(Node node) {

**if** (node.data.compareTo(**this**.data)<0) {

**if**(**this**.left == **null**) {

**this**.left = node;

}**else** {

**this**.left.addNode(node);

}

}**else** {

**if**(**this**.right == **null**) {

**this**.right = node;

}**else** {

**this**.right.addNode(node);

}

}

}

**public** **void** printNode() {

**if**(**this**.left != **null**) {

**this**.left.printNode();

}

System.***out***.println(**this**.data + "\t");

**if**(**this**.right != **null**) {

**this**.right.printNode();

}

}

}

### Comparator

如果一个类已经开发完成，但建立初期没有实现comparable接口，此时无法排序，个Compator,在java.util包下。

此时需要指定好一个比较器的比较规则类才可以完成数组排序。

eg：

public class StudentComparator implements Comparator<Student>{  
 public int compare(Student s1,Student s2){}

}

排序：java.util.Arrays.sort(stu,new StudentComparator())

中文排序：

Comparator c = Collator.getInstance(Locale.CHINA);

## 获取系统时间

Date

Calendar

Java8专门新增一个java.time包

|  |
| --- |
| Date date = **new** Date();  Timestamp CUR\_time = **new** Timestamp(date.getTime()); |

## 定时调度

### Timer类

Timer类是一种线程设施，可实现在某一段时间或某一段时间后安排某一个任务执行一次或者定期重复执行。需与TimeTask配合。

|-public Timer()-启动

|-public void cancel()-终止，放弃已安排的任务，对正在执行的任务没有影响

|-public int purge-将所有已移除的任务移除，释放内存空间

|-public void schedule(TimeTask task,Date time):安排一个任务在指定时间执行，若超过则立即执行

|- public void schedule(TimeTask task,Date firstTime,long period):repeat

|- public void schedule(TimeTask task,long delay)

|- public void schedule(TimeTask task,long delay,long period)

|- public void scheduleAtFixedRate(TimeTask task,Date firstTime,long period)

|- public void scheduleAtFixedRate(TimeTask task,long delay,long period)

两者针对出现的延迟情况处理

### TimerTask类

|-public void cancel()

|-public void run()-引入接口Runnable，需要复写

|-public long scheduled ExectionTime()-返回最近一次要执行任务的时间，若正在执行，返回安排时间，一般在run中调用，判断当前是否有足够时间执行完成该任务

eg：

public void MyTask extends TimerTask{

public void run(){ }

}

Timer t = new Timer();

MyTask mytask = new MyTask();

t.schedule(mytask,1000,2000);

## 正则表达式

* Boolean matches(String regex)
* String replaceAll(String regex, String replacement)
* String replaceFirst()
* String[] split(String regex)

正则表达式所支持的合法字符

x 字符x，可代表任何合法字符

\0mnn

$匹配一行的结尾

^匹配一行的开头

### Pattern类和Matcher类

两个类都在java.util.regex中定义。

常用正则规范：

|-\\ \t \n [abc] [^abc] [a-zA-Z0-9] \d（数字） \D（非数字）

|-\w(字母、数字、下划线) \W \s(所有空白字符，换行、空格) \S

|-$行的结尾 ^行的开头 .匹配除换行符之外的任意字符

数量表示：

X-必须出现一次 X?(0或1次) X\*(0或1或多次) X+(1或多次)

X{n}出现n次， X{n，}必须出现n次以上 X{n,m}出现n~m次

逻辑运算符

XY，X规范后跟着Y规范

X|Y，X或Y规范

(X)作为一个捕获组规范

Pattern常用方法

|-public static Pattern cmpile(String regex);

|-public Matcher matcher(CharSequence input);

|-public String[] split(CharSequence input);

要取得Pattern实例必须调用compile()方法

Matcher类的常用方法

|-public Boolean matches()

|-public String replaceAll(String replacement)

eg:

String string = "1990-10-28";

Pattern pattern = Pattern.*compile*("-");

Matcher matcher = pattern.matcher(string);

System.***out***.println(matcher.matches());

String[] strings = pattern.split(string);

String strings2= matcher.replaceAll("%");

**for**(String x:strings) {

System.***out***.println(x);

}

System.***out***.println(strings2);

### String对正则表达式的支持

|-public Boolean matches(String regex)

|-public String replaceAll(String regex, String replacement)

|-public String[] split(String regex)

## 集合

当使用Iterator迭代访问Collection集合元素时，元素不能改变，只有通过Interator的remove方法删除上一次next方法返回的集合元素才可以。

Interator迭代器采用的快速失败机制，一旦迭代过程中检测到该集合已经被修改，程序立即引发ConcurrentModificationException，而不是现实修改后的结果。

可使用如下删除：

Iterator<type> it = list.iterator();

While(it.hasNext){

Type x = it.next();

If(){it.remove();}

}

Collection-Set-HashSet

-TreeSet

-List-LinkedList

-ArrayList

-Queue

Map-HashMap

-TreeMap

### Collection

Add(E o)/addAll(Collection <? Extends E>c)/clear()

contains(Object o)/containAll(Collection<?> c)/equals()/hashCode()/isEmpty()

Iterator<E> iterator()/remove(Object o)/removeAll(Collection<?> c)

retainAll(Collection<?> c)/size()/toArray()/public <T> T[] toArray(T[] a)

### List

|-indexOf()-取得重复数据的第一个

|-lastIndexOf()-取得重复对象最后一次出现的索引位置

|-ArrayList：实现可变长数组，可包含null，可快速随机访问，insert or delete教慢，需要后面所有数据移位

|-Iterator\ListIterator

List接口中的subList()方法为返回一个列表的子视图，所有的改变作用于原字符串

List.subList(20,30).clear()

### Set

不包含重复对象

### Map

|-put\containKey\containValue\get\keySet\values

### Java8新增的Predicate操作集合

Java8为Collection集合新增removeIf(Predicate filter)方法，Predicate也是函数式接口，可以使用Lambda作为参数。

### Collections

addAll()/reverse()/replaceAll()/sort()/swap()/EMPTY\_LIST/EMPTY\_SET/EMPTY\_MAP

### Stack

Empty()/peek()/pop()/push()/search()

### Properties

属性类Properties为Hashtable的子类

GetProperty()/setProperty()/list()/load()/loadFromXML/store()/storeToXML()

### 运算

* 并集：list.addAll(list2)
* 交集：list.retainAll(list2)
* 差集：list.remove(list2)
* 无重复的并集：list2.removeAll(list1)   
   list1.addAll(list2)
* Collections.shuffle()，打乱列表

## StringBuilder

StringBuilder相较于“+”，效率高，不会重复创建

StringBuilder（速度快）与StringBuffer(线程安全)

Append()\insert()\delete()\reverse()

## 泛型

允许在定义类、接口、方法时使用类型形参。

不管泛型的类型形参传入哪一种类型实参，对于Java来说，依然被当成一个类处理，因而静态方法、静态初始化块或静态变量的申明和初始化都不允许使用类型形参。

受限泛型通配符：list<?extends Shape>

下限通配符<? Super Type>

对于泛型如果实例化时不指定，则擦除泛型，使用Object表示，出现警告

使用通配符？可使用任意的泛型对象，但如果使用？接受泛型对象，则不能设置被泛型指定的内容

子类的泛型是无法使用父类的泛型，如Info<String>不能使用Info<Object>接收

泛型可以使程序的操作更加安全，可以避免发生类转换异常

### 泛型方法与通配符区别

类型通配符既可以在方法签名中定义形参的类型，也可以用于定义变量的类型；但泛型的类型形参必须在对应方法中显式申明。

### 泛型接口

两种实现方式：

1. 在子类的定义上声明泛型类型  
   interface Info<T>;  
   class InfoImp<T> implements Info<T>;  
   Info<String> I = new InfoImp<String>(“LDC”);
2. 直接在接口中指定泛型；  
   interface Info<T>;  
   class InfoImp implements Info<String>;  
   Info<String> I = new InfoImp(“LDC”);

### 泛型数组

使用泛型方法时，传递或者返回一个泛型数组

public static <T> T[] fun(T parm[])

## 国际化程序

国际化需要依托，java.util.Locale,表示一个国家语言类

java.util.ResourceBundle：用于访问资源文件

java.text.MessageFormat:格式化资源文件的占位字符串

### Locale

|-public Locale(String language)

|-public Locale(String language, String country)

zh-CN\en-US\fr-FR

### ResourceBundle

ResourceBundle类主要是用来读取属性文件，读取文件时指定名称，无需后缀.properties

|-public static final ResourceBundle getBundle(String baseName)

|-public static final ResourceBundle getBundle(String baseName, Locale locale)

|-public final String getString(String key)

eg:

Message.properties: info=hello

ResouceBundle rb = ResouceBundle.getBundle(“Message”);

rb.getString(“info”);

### 国际化程序实现

Message\_zh\_CN.properties:不支持中文，需要转化为Unicode，使用“native2ascii.exe”转换

Message\_en\_US.properties

Message\_fr\_FR.properties

Local zhLoc = new Local(“zh”, “CN”);

Local enLoc = new Local(“en”, “US”);

Local frLoc = new Local(“fr”, “FR”);

ResouceBundle zhrb = ResouceBundle.getBundle(“Message”,zhLoc);

ResouceBundle enrb = ResouceBundle.getBundle(“Message”,enLoc);

ResouceBundle frrb = ResouceBundle.getBundle(“Message”,frLoc);

zhrb.getString(“info”);

zhen.getString(“info”);

zhfr.getString(“info”);

### MessageFormat处理动态文本

Fromat中派生MessageFormat、DateFormat、NumberFormat

info=hello,{0}!,用占位符表示。

读取后需要使用MessageFormat处理，主要使用一下方法

public static String format(String pattern, Object … argument)

MessageFormat.format(str,{0},{1},{2}…)

eg:

stren = zhen.getString(“info”);

MessageFormat.format(stren,”LDC”)

### 使用类文件代替资源文件

1. 必须继承java.util.ListResourceBundle类
2. 需要复写gtContent()方法
3. 属性是个二维数组，key-value
4. 若同时出现Message\_zh\_CN.class > Message\_zh\_CN.properties > Message.properties

## 异常处理

Java异常包括两种，checked异常和runtime异常

把所有非正常的情况分为两种：异常Exception和错误Error。

RuntimeException可以不处理，其他Exception需要处理。

### 常见的异常类之间的继承关系



IndexOutOfBoundsException

NumberFormatException

ArithmeticException

### 访问异常信息

* getMerssage():返回该异常的详细描述字符串
* printStackTrace():将该异常的跟踪栈信息输出到标准错误输出
* printStackTrace(PrintStream s):输出到指定流
* getStackTrace()：返回该异常的跟踪栈信息

### 5 finally

异常处理代码中使用return不会退出finally语句，而System.exit(1)语句退出虚拟机则会。

不要在finally块中使用如return或throw等导致方法终止的语句。

### Java7的自动关闭资源的try语句

Try关键字后面紧跟圆括号，声明、初始化一个或多个资源，此处的资源是指那些在程序结束时显式关闭的资源（如数据库连接、网络连接等）。Try语句会在该语句结束时自动关闭这些资源。

为保证try可以正常关闭资源，这些资源必须实现AutoCloseable或Closeable接口，实现close()方法。

## 断言

assert boolean 表达式；

如果要使断言语句生效，需要使用-enableassertions参数，简写为-ea

java –ea Test

## AWT(Abstract Window Tookit)

### AWT容器

Container是Component的子类。



AWT主要提供两种主要的容器类型。

* Window：可独立存在的顶级窗口
* Panel：可作为容器容纳其他组件，不能独立存在，必须被添加到其他容器中

## Annotation

### @Override

强制编译器校验重写

### @Deprecated

表明该方法、类过程，其他程序访问时编译器会警告

### @SuppressWarnings

抑制编译器警告

@SuppressWarnings({“unchecked”, “deprecation”})

@SuppressWarnings(value={“unchecked”, “deprecation”})

Deprecation、unchecked

Fallthrough:switch操作case后未加break

Path：设置错误的文件路径、源文件路径

Serial：可序列化类上缺少serialVersionUID

Finaly：任何finally子句不能正常完成时候的警告

All

### @SafeVarags

堆污染警告

### @FunctionalInterface

规定该接口是函数式接口

### 自定义

使用@interface相当于继承了Annotation接口

Public @interface MyDefaultAnnotationMoreParm{

Public String key() default “”;

Public String value();

}

@MyDefaultAnnotationMoreParm(key = “”, value = “”)

Public @interface MyDefaultAnnotationArrayParm{

Public String[] value();

}

### Retention和RetentionPolicy

Retention定义中存在一个RetentionPolicy变量用于指定Annotation范围。

SOURCE：保留在源文件中\*.java，override、supresswarinings

CLASS：保留在源文件和变异之后的类文件\*.class

RUNTIME：还会加载到JVM中，deprecated

### 通过反射取得Annotation

Class<?> c = null;

C = Class.forName(“”);

Method toM = c.getMethod(“methodname”);

If(toM.isAnnotationPresent(className.class)){

className mda = null;

mda = toM.getAnnotion(className.class);

String key = mda.key();

}

### @Target

使用@Target限定Annotation的使用位置。

ElementType[]枚举类型，如ANNOTATION\_TYPE，用在注释的申明中。

### @Documented

自定义的Annotation可通过@Documented注释，生成javadoc的时候可以通过@Documented将一些文档的说明信息写入。

@Documented

Public @interface AnnotationName{}

生成文档：javadoc -d doc SimpleBeanDocumented.java

### @Inherited

标注父类的注释是否可以被子类所注释。需要被继承则添加。

## 多线程

创建线程：

|-实现Runnable接口

|-继承Thread类

推荐使用第一种，原因：

1）Thread仅支持单继承

2）创建大量的Thread类，开销较大

3）5.0中新增的很多简化线程的类是使用Runnable接口的

Start-就绪-阻塞-运行-终止

不能直接使用run()启动进程，需要使用Thread类的start()启动，实际是调用

private native void start0();

操作系统的函数，若多次调用，会报异常：IllegalThreadStateException

一个JAVA程序启动至少启动两个线程：1）main；2）垃圾收集线程

suspend()-挂起

resume()-恢复挂起

stop()-停止线程

以上三种均不推荐，可能造成死锁，可使用加入标志，stop中修改该标志

### Thread

Thread也实现了Runnable接口，位于java.lang。

可通过继承Thread来启动新线程。

Thread.State

(new thread)-New-(start())->Runnable-(run()finished or exception)->TERMINATED

Runnable: |-BLOCKED-得到和等待锁

|-TIMED\_WATTING-休眠状态-sleep()

|- WATTING-等待状态-wait()

|-Thread(Runnable target)

|-Thread(Runnable target, String name)

|-public static Thread currentThread()

|-getName()\getPriority()\isInterrupted()\isAlive()

|-run()\start()\setName()\setPriority()\toString()\sleep()

|- public final void join() throws Interrupted Exception;等待线程死亡，线程强制执行

|- public final sychronized void join(long millis) throws Interrupted Exception;等待millis，线程死亡

|-public static void yield();将正在执行的线程暂停，允许其他线程执行

|-public final void setDaemon(Boolean on)：将一个线程设置成后台运行

threada.join()-强制执行

Thread.sleep(100)

threada.interrupt()

threada.setDaemon(true)

### Runnable

该接口中只有run()方法。

|-新建类，实现Runnable接口，复写run()

|-实例化类对象

|-使用Thread类的构造方法创建Thread对象

|-调用Thread类的start()方法来运行新线程

### Attribute

1)Priority: gePriority()\setPriority()

2)nameAttribute:getName()\setName(),default:Thread-number

3)Id:gitId(),Class thread donnot support modify method of id

4)daemon thread

5)state:getState()-Thread.currentThread().getState()

### Control

1. Sleep:public static void sleep(long millis, int nanos)throws InterruptedException
2. Join: public final void join(long millis, int nanos)throws InterruptedException,with no para;with long millis
3. Stop:Though thread support stop() method ,but it’s not safe.Recommand use Boolean flag to control

### Sychronized

1. Sychronized method:expand much resource,not recommended
2. Sychronized block:
3. Volatile:notify JVM the para may be modify,need be checked
4. ReentrantLock(可重入锁):Java.util.concurrent-ReentrantLock()\lock()\unlock()  
   eg:lock.lock()  
    try{}  
    finally{lock.unlock()}
5. AtomicPara:java.util.concurrent.atomic,AtomicInteger(int initialValue)\getAndIncrement()

### Callable

Java.util.concurrent包含高并发

@FunctionalInterface

public interface Callable<V> {

V call() throws Exception;

}

### Priority

public static final int MIN\_PRIORITY- 1

public static final int NORMAL\_PRIORITY- 5

public static final int MAX\_PRIORITY- 10

main is NORMAL

### Sychronized

|-同步代码块

synchronized(同步对象){

需要同步的代码；

}

|-同步方法

死锁

wait()

notify()-唤醒第一个等待的线程

notifyAll()-唤醒所有等待的线程

## 类加载机制与反射

正常方式：引入需要的“包.类”名称->通过new实例化->取得实例化

反射方式：实例化对象->getClass()方法->得到完整的包.类名称

Class的常用方法：forName()/getConstructors()/getDeclaredFields()(单独定义的所有属性)/getFields()(继承而来的全部属性)/getMethods()/getMethod()/getInterfaces()/getName/

getPackage()/getSuperclass()/newInstance()/getComponentType()/isArray()

### Class实例化

Class.forName(“java.util.lang”);

New X().getClass();//需要明确的类

X.class();

### 调用类中的方法

* C = Class.forName(“java.util.\*\*\*”);//实例化class对象
* Obj = c.newInstance();//实例化操作对象，对象中需要有无参构造函数
* Met = obj.getClasss().getMethod(methodName,methodParameter1Type,…);//获取参数方法
* Met.invoke(obj, methodParameter1,…);//调用方法

### 操作属性

1. 可以通过调用类中的getter、setter方法
2. 直接通过Field类中的提供的set、get方法，由于类中属性大多数为private，需要使用setAccessible(true)方法将需要操作的方法设置为可以外部访问
   1. C = Class.forName(“org.\*\*\*”);
   2. Org = c.getInstance();
   3. nameField = c.getDeclaredField(“name”);
   4. nameFiled.setAccessible(true);
   5. nameField.set(org, “nameValue”);

### 使用反射操作应用类

### 类的生命周期

* 装载：通过类加载器把.class二进制文件装入JVM的方法区，并在堆区创建描述该类的java.lang.Class对象
* 链接：把二进制数据组装成可以运行的状态
  + 校验：确认该文件是否适合当前的JVM版本
  + 准备：为静态成员分配内存空间，设置默认值
  + 解析：转换常量池的代码为直接引用的过程，直到所有的符号引用都可被运行程序使用
* 初始化
* 对象实例化
* 垃圾收集
* 对象终结
* 卸载

### 动态代理

需要java.lang.reflect.InvocationHandle接口和java.lang.reflect.Proxy类的支持。

Public interface InvocationHandle{

Public Object invoke(Object proxy, Method method, Object[] args) throws Throwable

}

Proxy类专门完成代理的操作类。

Public static Object newProxyInstance(ClassLoader loader, Class<?>[] interfaces,

InvocationHandle h) throws IllegalArgumentException

ClassLoader

* BootStrap ClassLoader：C++编写
* Extension ClassLoader：用来进行扩张类的加载，对应jre\lib\ext目录中的类
* APPClassLoader：加载classpath制定的类

class MyInvocationHandler implements InvocationHandler{  
 private Object target;  
 public Object bind(Object target){  
 this.target = target;  
 return Proxy.*newProxyInstance*(target.getClass().getClassLoader(), target.getClass().getInterfaces(), this);  
 }  
 public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
 Object returnResult = method.invoke(this.target, args);  
 return returnResult;  
 }  
}

## JVM

.java->javac->.class->JVM(指令集、寄存器、类文件格式、栈、垃圾回收堆、存储区)

## 文件

File.separator(分隔符，根据操作系统决定是使用/或者\)

File.pathSeparator(表示路径的分隔符，如 ；)

New file(path);

OutputStream-FileOutputStream

-ByteArrayOutputStream

-PipedOutputStream

-FilterOutputStream----------PrintStream

-ObjectOutputStream DateOutputStream

DeflaterOutputStream-ZipOutputStream

InputStream-FileInputStream

-ByteArrayOutputStream

-PipedInputStream

-FilterInputStream-------------DataInputStream

-SequenceInputStream InflaterInputStream-ZipInputStream

-ObjectInputStream PushbackInputStream

Writer-OutputStreamWriter-FileWriter

-PrintWriter

Reader-InputStreamReader-FileReader

-BufferedReader

-FilterReader-PushbackReader

### 取得文件信息

|-判断是否为文件：public Boolean isFile();

|-判断是否为目录：public boolean isDirectory()

|-列出该集目录下所有文件：public [File](https://docs.oracle.com/javase/8/docs/api/java/io/File.html)[] listFiles()

Public static void listDir(File file){

If(file.isDirectory()){

File[] result = file.listFiles();

For(int x=0;x<result.size;x++){

listDir(result[x]);

}

}

}

### 常用方法

|-getName()

|-canRead()

|-canWrite()

|-exist()

|-length()

|-getAbsolutePath()

|-getParent()

|-isHidden()

|-Long lastModified()

|-listRoots()

|-mkdir()

|-renameTo()

### InputStream

InputStream io = System.***in***;

**byte**[] b = **new** **byte**[1024];

**try** {

System.***out***.println("Please input!");

io.read(b);

String input = **new** String(b);

System.***out***.println("Input is:"+input.trim());

}**catch** (IOException e) {

// **TODO**: handle exception

e.printStackTrace();

}**finally** {

**try** {

io.close();

}**catch** (IOException e) {

e.printStackTrace();

}

}

### OutputStream

OutputStream io = System.***out***;

**try** {

**byte**[] bs ="Hello World!\n".getBytes();

io.write(bs);

}**catch** (IOException e) {

// **TODO**: handle exception

e.printStackTrace();

}**finally** {

**try** {

io.close();

}**catch** (IOException e) {

e.printStackTrace();

}

}

### 文件输入流

FileInputStream(File file);

FileInputStream(String path);

**try** {

FileInputStream io = **new** FileInputStream("D:\\test.txt");

**byte**[] bs = **new** **byte**[1024];

**int** length;

**while**((length=io.read(bs))!=-1) {

String string = **new** String(bs, 0, length);

System.***out***.println(string);

}

}**catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

}

### 文件输出流

FileOutputStream

**try** {

File file = **new** File("D:\\test.txt");

**if**(!file.exists()) {

file.createNewFile();

}

FileOutputStream io = **new** FileOutputStream(file);

**byte**[] bs = **new** String("Heollo everbody!!!!").getBytes();

io.write(bs);

}**catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

}

### 缓存输入流

bufferedInputStream(inputStream in);

bufferedinputStream(inputStream in, int size);

### 缓存输出流

BufferedoutputStream

### 数据输入流

DateInputStream(inputStream in)

|-readBoolean()

|-readByte()

|-readChar()

|-readInt()

|-readFloat()

|-readUTF()

### 数据输出流

DateOutputStream

### 字符输入流

Reader-InputStreamReader

|-close()

|-mark()

|-read()

|-reset()

FIleReader

Window换行符为\r\n，linux为\n

### 字符输出流

Writer-PrintWriter

|-close()

|-flush()

|-write()

FileWriter

### 管道流

PipedInputStream/PipedOutputStream

若要进行管道输出，则必须把输出流连接到输入流上，如：

PipedOutputStream类中Public void connect(PipedInputStream snk) throws IOExcetpiton

### Scanner

Scanner(File source)

Scanner(String source)

Scanner(InputStream source)

|-findInLine()

|-nextInt()

|-nextShort()

|-nextFloat()

String input = **new** String("2 018-hleoolo-98912-gsgsgs-1234-agag8788");

Scanner scanner= **new** Scanner(input);

scanner.findInLine("\\d");

MatchResult result = scanner.match();

**for**(**int** i =0;i<result.groupCount();i++) {

System.***out***.println(result.group(i));

}

scanner.close();

### System中常量

Public static final PrintStream out\err;

Public static final InputStream in;

重定向方法

Public static void setOut(PrintStream out);

Public static void setErr(PrintStream err);

Public static void setIn（InputStream in）；

## 编码

### ISO8859-1

单字节编码，0~255

### GBK/GBK2312

中文的国际编码，双字节编码，GBK可以表示简体中文和繁体中文，二GB2312只能表示简单中文。

### Unicode

16进制表示编码

### UTF

UTF兼容IOS8859-1，为不定长编码，每个字符长度为1~6个字节不等，一般在中文网页使用。

### 本机编码

System.getProperty(“file.encoding”);

制定编码

Byte b[] = “中国，你好”.getBytes(“IOS8859-1”);

## 序列化

### Serializable

需要实现Serializable接口（java.io.Serializable-标识接口）

实现序列化接口的对象可以经过而二进制数据流进行传输，需要依靠对象输出流和输入流。（ObjectOutputStream、ObjectInputStream）。只有属性被序列化。

序列化和反序列化若JDK版本不一致，则会造成异常。当实现java.io.Serializable没有显式定义serialVersionUID，则会自动生成该long变量。

序列化：ObjectOutputStream，writeObject

反序列化：ObjectInputStream，readObject

### Externalizable

指定序列化的内容，实现该接口。需要复写以下方法。

WriteExternal(ObjectOutput out)

readExteranl(ObjectOutput in)

### Transient

使用transient申明可以不序列化

## 网络编程

TCP(Transmission Control Protocol)

UDP(User Datagram Protocol)

### InetAddress

java.net.InetAddress

|-getByName()-返回InetAddress

|-getHostAddress()

|-getHostName()

|-get()

|-getLocalHost()-返回InetAddress

|-toString()

### ServerSocket

ServerSocket();

ServerSocket(int port);

ServerSocket(int port, int backlog);

构造会抛出IOException

|-accept(),返回Socket

|-isBound()

|-getInetAddress()-返回InetAddress

|-isClosed()

|-bind()

|-getLocalPort()

### Socket

Socket(String host,int port);

Socket(InetAddress address,int port);

throws UnknownHostException, IOException

|-getInetAddress()-return InetAddress

|-getPort()

|-getLocalAddress()

|-close()

|-getInputStream()

|-getOutputStream()

### UDP

### URL

Uniform Resource Locator，统一资源定位符

* Public URL(String protocol, String host, String file) throws MalformedURLException
* Public URLConnection openConnection() throws IOException
* Public final InputStream openStream() throws IOException

### Encode\Decode

Public static String encode(String s, String enc) throws UnsupportedEcodingException

## 设计模式

### 工厂模式

### 代理模式

### 适配器模式

### 观察者设计模式

在java.util包中提供了Observable类和Observer接口，可完成观察者模式。

被观察者需要继承Observable类。

|-public void addObserver(Observer o)

|-public void deleteObserver(Observer o)

|-public void setChanged()

|-public void notifyObserver(Object arg)

每一个观察者需要实现Observer接口

Observer接口定义如下：

public interface Observer{

void update(Observer o, object arg);

}

## Logger

### 创建Logger对象

Static Logger getLogger(Sting name);

为制定子系统查找或创建一个logger。

Static Logger getLogger(Sting name，String resourceBundleName);

为制定子系统查找或者创建一个logger。

### Logger级别

|  |  |
| --- | --- |
| SEVERE | 严重 |
| WARNING | 警告 |
| INFO | 信息 |
| CONFIG | 配置 |
| FINE | 良好 |
| FINER | 较好 |
| FINEST | 最好 |
| ALL | 开启所有级别日志记录 |
| OFF | 关闭所有级别日志记录 |

Logger的默认级别为info。登记在jre的lib的logging.properties文件中

java.util.logging.ConsoleHandler.level = INFO

java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

### Handle

Handle对象从Logger中获取日志信息，并将这些信息导出。

java.util.logging.Handler

java.util.logging.MemoryHandler

java.util.logging.StreamHandler

java.util.logging.ConsoleHandler

java.util.logging.FileHandler

java.util.logging.SocketHandler

可通过setLevel(Level.OFF)来禁用Handle，并可通过执行适当级别的setLevel来重新启用。Handle类通常使用LogManager属性来设置Handle的Filter、Formatter和Level的默认值。

### Formatter

java.util.logging.Formatter

java.util.logging.SimpleFormatter

java.util.logging.XMLFormatter

每个日志记录Handler都有关联的Formatter。Formatter接收LogRecord，并将它转换为一个字符串。

## 实例

### 进销存管理系统

|-数据库设计

# 数据库

## SQL语句结构

### 数据查询语句（Data Query Language）

也成为数据减速语句，select\where\order by\group by\having

### 数据操作语言（Data Manipulation Language）

Insert\update\delect

### 事务处理语言（TPL）

保证被DML语句影响的表的所有行能及时更新

Begin transaction\commit\rollback

### 数据控制语言 （DCL）

语句通过grant\revoke获得许可，确定当个用户和用户组对数据库对象的访问

### 数据定义语言（DDL）

Create/drop

### 指针控制语言（CCL）

Declear cursor /fetch into/update where current

## SQL数据类型

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| CHARACTER(n) | 字符/字符串。固定长度 n。 |
| VARCHAR(n) 或 CHARACTER VARYING(n) | 字符/字符串。可变长度。最大长度 n。 |
| BINARY(n) | 二进制串。固定长度 n。 |
| BOOLEAN | 存储 TRUE 或 FALSE 值 |
| VARBINARY(n) 或 BINARY VARYING(n) | 二进制串。可变长度。最大长度 n。 |
| INTEGER(p) | 整数值（没有小数点）。精度 p。 |
| SMALLINT | 整数值（没有小数点）。精度 5。 |
| INTEGER | 整数值（没有小数点）。精度 10。 |
| BIGINT | 整数值（没有小数点）。精度 19。 |
| DECIMAL(p,s) | 精确数值，精度 p，小数点后位数 s。例如：decimal(5,2) 是一个小数点前有 3 位数小数点后有 2 位数的数字。 |
| NUMERIC(p,s) | 精确数值，精度 p，小数点后位数 s。（与 DECIMAL 相同） |
| FLOAT(p) | 近似数值，尾数精度 p。一个采用以 10 为基数的指数计数法的浮点数。该类型的 size 参数由一个指定最小精度的单一数字组成。 |
| REAL | 近似数值，尾数精度 7。 |
| FLOAT | 近似数值，尾数精度 16。 |
| DOUBLE PRECISION | 近似数值，尾数精度 16。 |
| DATE | 存储年、月、日的值。 |
| TIME | 存储小时、分、秒的值。 |
| TIMESTAMP | 存储年、月、日、小时、分、秒的值。 |
| INTERVAL | 由一些整数字段组成，代表一段时间，取决于区间的类型。 |
| ARRAY | 元素的固定长度的有序集合 |
| MULTISET | 元素的可变长度的无序集合 |
| XML | 存储 XML 数据 |

### 字符型（VARCHARVS CHAR）

VARCHARVS和CHAR用于存储字符串长度小于255的字符

VARCHARVS会去除空白字符，而CHAR按照定义自动填充空白字符

### 文本型（TEXT）

可存放超过二十亿个字。尽量避免使用

一旦向文本型字段输入任何数据（甚至是NULL），就会有2K空间自动被分配给该数据。至于删除该数据才能释放空间

### 数值型（INT、NUMERIC、MONEY）

#### INT\SAMLLINT\TINYINT

INT-4BYTE

TINYINT-1BYTE

#### NUMERIC

-10^38 – 10^38

#### MONEY\SMALLMONEY

SMALLMONEY: -214,748.3648-214,748.3647

MONEY: -922,337,203,685,477.5808-922,337,203,685,477.5807

### 逻辑型（BIT）

0/1

### 日期型

DATETIME: 1753年1月1日第一毫秒到9999年12月31日最后一毫秒

SMALLDATETIME: 从1900年1月1日到2079年6月6日的日期，它只能精确到秒

### SQL MS ACCESS\MYSQL\SQL SERVER数据类型

#### Microsoft Access 数据类型

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| Text | 用于文本或文本与数字的组合。最多 255 个字符。 |  |
| Memo | Memo 用于更大数量的文本。最多存储 65,536 个字符。**注释：**无法对 memo 字段进行排序。不过它们是可搜索的。 |  |
| Byte | 允许 0 到 255 的数字。 | 1 字节 |
| Integer | 允许介于 -32,768 与 32,767 之间的全部数字。 | 2 字节 |
| Long | 允许介于 -2,147,483,648 与 2,147,483,647 之间的全部数字。 | 4 字节 |
| Single | 单精度浮点。处理大多数小数。 | 4 字节 |
| Double | 双精度浮点。处理大多数小数。 | 8 字节 |
| Currency | 用于货币。支持 15 位的元，外加 4 位小数。**提示：**您可以选择使用哪个国家的货币。 | 8 字节 |
| AutoNumber | AutoNumber 字段自动为每条记录分配数字，通常从 1 开始。 | 4 字节 |
| Date/Time | 用于日期和时间 | 8 字节 |
| Yes/No | 逻辑字段，可以显示为 Yes/No、True/False 或 On/Off。在代码中，使用常量 True 和 False （等价于 1 和 0）。**注释：**Yes/No 字段中不允许 Null 值 | 1 比特 |
| Ole Object | 可以存储图片、音频、视频或其他 BLOBs（Binary Large OBjects）。 | 最多 1GB |
| Hyperlink | 包含指向其他文件的链接，包括网页。 |  |
| Lookup Wizard | 允许您创建一个可从下拉列表中进行选择的选项列表。 | 4 字节 |

#### MySQL 数据类型

在 MySQL 中，有三种主要的类型：Text（文本）、Number（数字）和 Date/Time（日期/时间）类型。

Text 类型：

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| CHAR(size) | 保存固定长度的字符串（可包含字母、数字以及特殊字符）。在括号中指定字符串的长度。最多 255 个字符。 |
| VARCHAR(size) | 保存可变长度的字符串（可包含字母、数字以及特殊字符）。在括号中指定字符串的最大长度。最多 255 个字符。**注释：**如果值的长度大于 255，则被转换为 TEXT 类型。 |
| TINYTEXT | 存放最大长度为 255 个字符的字符串。 |
| TEXT | 存放最大长度为 65,535 个字符的字符串。 |
| BLOB | 用于 BLOBs（Binary Large OBjects）。存放最多 65,535 字节的数据。 |
| MEDIUMTEXT | 存放最大长度为 16,777,215 个字符的字符串。 |
| MEDIUMBLOB | 用于 BLOBs（Binary Large OBjects）。存放最多 16,777,215 字节的数据。 |
| LONGTEXT | 存放最大长度为 4,294,967,295 个字符的字符串。 |
| LONGBLOB | 用于 BLOBs (Binary Large OBjects)。存放最多 4,294,967,295 字节的数据。 |
| ENUM(x,y,z,etc.) | 允许您输入可能值的列表。可以在 ENUM 列表中列出最大 65535 个值。如果列表中不存在插入的值，则插入空值。  **注释：**这些值是按照您输入的顺序排序的。  可以按照此格式输入可能的值： ENUM('X','Y','Z') |
| SET | 与 ENUM 类似，不同的是，SET 最多只能包含 64 个列表项且 SET 可存储一个以上的选择。 |

Number 类型：

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| TINYINT(size) | -128 到 127 常规。0 到 255 无符号\*。在括号中规定最大位数。 |
| SMALLINT(size) | -32768 到 32767 常规。0 到 65535 无符号\*。在括号中规定最大位数。 |
| MEDIUMINT(size) | -8388608 到 8388607 普通。0 to 16777215 无符号\*。在括号中规定最大位数。 |
| INT(size) | -2147483648 到 2147483647 常规。0 到 4294967295 无符号\*。在括号中规定最大位数。 |
| BIGINT(size) | -9223372036854775808 到 9223372036854775807 常规。0 到 18446744073709551615 无符号\*。在括号中规定最大位数。 |
| FLOAT(size,d) | 带有浮动小数点的小数字。在 size 参数中规定最大位数。在 d 参数中规定小数点右侧的最大位数。 |
| DOUBLE(size,d) | 带有浮动小数点的大数字。在 size 参数中规定最大位数。在 d 参数中规定小数点右侧的最大位数。 |
| DECIMAL(size,d) | 作为字符串存储的 DOUBLE 类型，允许固定的小数点。在 size 参数中规定最大位数。在 d 参数中规定小数点右侧的最大位数。 |

\*这些整数类型拥有额外的选项 UNSIGNED。通常，整数可以是负数或正数。如果添加 UNSIGNED 属性，那么范围将从 0 开始，而不是某个负数。

Date 类型：

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| DATE() | 日期。格式：YYYY-MM-DD  **注释：**支持的范围是从 '1000-01-01' 到 '9999-12-31' |
| DATETIME() | \*日期和时间的组合。格式：YYYY-MM-DD HH:MM:SS  **注释：**支持的范围是从 '1000-01-01 00:00:00' 到 '9999-12-31 23:59:59' |
| TIMESTAMP() | \*时间戳。TIMESTAMP 值使用 Unix 纪元('1970-01-01 00:00:00' UTC) 至今的秒数来存储。格式：YYYY-MM-DD HH:MM:SS  **注释：**支持的范围是从 '1970-01-01 00:00:01' UTC 到 '2038-01-09 03:14:07' UTC |
| TIME() | 时间。格式：HH:MM:SS  **注释：**支持的范围是从 '-838:59:59' 到 '838:59:59' |
| YEAR() | 2 位或 4 位格式的年。  **注释：**4 位格式所允许的值：1901 到 2155。2 位格式所允许的值：70 到 69，表示从 1970 到 2069。 |

\*即便 DATETIME 和 TIMESTAMP 返回相同的格式，它们的工作方式很不同。在 INSERT 或 UPDATE 查询中，TIMESTAMP 自动把自身设置为当前的日期和时间。TIMESTAMP 也接受不同的格式，比如 YYYYMMDDHHMMSS、YYMMDDHHMMSS、YYYYMMDD 或 YYMMDD。

#### SQL Server 数据类型

String 类型：

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| char(n) | 固定长度的字符串。最多 8,000 个字符。 | Defined width |
| varchar(n) | 可变长度的字符串。最多 8,000 个字符。 | 2 bytes + number of chars |
| varchar(max) | 可变长度的字符串。最多 1,073,741,824 个字符。 | 2 bytes + number of chars |
| text | 可变长度的字符串。最多 2GB 文本数据。 | 4 bytes + number of chars |
| nchar | 固定长度的 Unicode 字符串。最多 4,000 个字符。 | Defined width x 2 |
| nvarchar | 可变长度的 Unicode 字符串。最多 4,000 个字符。 |  |
| nvarchar(max) | 可变长度的 Unicode 字符串。最多 536,870,912 个字符。 |  |
| ntext | 可变长度的 Unicode 字符串。最多 2GB 文本数据。 |  |
| bit | 允许 0、1 或 NULL |  |
| binary(n) | 固定长度的二进制字符串。最多 8,000 字节。 |  |
| varbinary | 可变长度的二进制字符串。最多 8,000 字节。 |  |
| varbinary(max) | 可变长度的二进制字符串。最多 2GB。 |  |
| image | 可变长度的二进制字符串。最多 2GB。 |  |

Number 类型：

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| tinyint | 允许从 0 到 255 的所有数字。 | 1 字节 |
| smallint | 允许介于 -32,768 与 32,767 的所有数字。 | 2 字节 |
| int | 允许介于 -2,147,483,648 与 2,147,483,647 的所有数字。 | 4 字节 |
| bigint | 允许介于 -9,223,372,036,854,775,808 与 9,223,372,036,854,775,807 之间的所有数字。 | 8 字节 |
| decimal(p,s) | 固定精度和比例的数字。  允许从 -10^38 +1 到 10^38 -1 之间的数字。  p 参数指示可以存储的最大位数（小数点左侧和右侧）。p 必须是 1 到 38 之间的值。默认是 18。  s 参数指示小数点右侧存储的最大位数。s 必须是 0 到 p 之间的值。默认是 0。 | 5-17 字节 |
| numeric(p,s) | 固定精度和比例的数字。  允许从 -10^38 +1 到 10^38 -1 之间的数字。  p 参数指示可以存储的最大位数（小数点左侧和右侧）。p 必须是 1 到 38 之间的值。默认是 18。  s 参数指示小数点右侧存储的最大位数。s 必须是 0 到 p 之间的值。默认是 0。 | 5-17 字节 |
| smallmoney | 介于 -214,748.3648 与 214,748.3647 之间的货币数据。 | 4 字节 |
| money | 介于 -922,337,203,685,477.5808 与 922,337,203,685,477.5807 之间的货币数据。 | 8 字节 |
| float(n) | 从 -1.79E + 308 到 1.79E + 308 的浮动精度数字数据。  n 参数指示该字段保存 4 字节还是 8 字节。float(24) 保存 4 字节，而 float(53) 保存 8 字节。n 的默认值是 53。 | 4 或 8 字节 |
| real | 从 -3.40E + 38 到 3.40E + 38 的浮动精度数字数据。 | 4 字节 |

Date 类型：

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| datetime | 从 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度为 3.33 毫秒。 | 8 字节 |
| datetime2 | 从 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度为 100 纳秒。 | 6-8 字节 |
| smalldatetime | 从 1900 年 1 月 1 日 到 2079 年 6 月 6 日，精度为 1 分钟。 | 4 字节 |
| date | 仅存储日期。从 0001 年 1 月 1 日 到 9999 年 12 月 31 日。 | 3 bytes |
| time | 仅存储时间。精度为 100 纳秒。 | 3-5 字节 |
| datetimeoffset | 与 datetime2 相同，外加时区偏移。 | 8-10 字节 |
| timestamp | 存储唯一的数字，每当创建或修改某行时，该数字会更新。timestamp 值基于内部时钟，不对应真实时间。每个表只能有一个 timestamp 变量。 |  |

其他数据类型：

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| sql\_variant | 存储最多 8,000 字节不同数据类型的数据，除了 text、ntext 以及 timestamp。 |
| uniqueidentifier | 存储全局唯一标识符 (GUID)。 |
| xml | 存储 XML 格式化数据。最多 2GB。 |
| cursor | 存储对用于数据库操作的指针的引用。 |
| table | 存储结果集，供稍后处理。 |

## 一些重要的SQL语句

|  |  |
| --- | --- |
| **SQL 语句** | **语法** |
| AND / OR | SELECT column\_name(s) FROM table\_name WHERE condition AND|OR condition |
| ALTER TABLE | ALTER TABLE table\_name  ADD column\_name datatype  or  ALTER TABLE table\_name  DROP COLUMN column\_name |
| AS (alias) | SELECT column\_name AS column\_alias FROM table\_name  or  SELECT column\_name FROM table\_name AS table\_alias |
| BETWEEN | SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2 |
| CREATE DATABASE | CREATE DATABASE database\_name |
| CREATE TABLE | CREATE TABLE table\_name ( column\_name1 data\_type, column\_name2 data\_type, column\_name2 data\_type, ... ) |
| CREATE INDEX | CREATE INDEX index\_name ON table\_name (column\_name)  or  CREATE UNIQUE INDEX index\_name ON table\_name (column\_name) |
| CREATE VIEW | CREATE VIEW view\_name AS SELECT column\_name(s) FROM table\_name WHERE condition |
| DELETE | DELETE FROM table\_name WHERE some\_column=some\_value  or  DELETE FROM table\_name  (**Note:**Deletes the entire table!!)  DELETE \* FROM table\_name  (**Note:**Deletes the entire table!!) |
| DROP DATABASE | DROP DATABASE database\_name |
| DROP INDEX | DROP INDEX table\_name.index\_name (SQL Server) DROP INDEX index\_name ON table\_name (MS Access) DROP INDEX index\_name (DB2/Oracle) ALTER TABLE table\_name DROP INDEX index\_name (MySQL) |
| DROP TABLE | DROP TABLE table\_name |
| GROUP BY | SELECT column\_name, aggregate\_function(column\_name) FROM table\_name WHERE column\_name operator value GROUP BY column\_name |
| HAVING | SELECT column\_name, aggregate\_function(column\_name) FROM table\_name WHERE column\_name operator value GROUP BY column\_name HAVING aggregate\_function(column\_name) operator value |
| IN | SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1,value2,..) |
| INSERT INTO | INSERT INTO table\_name VALUES (value1, value2, value3,....)  *or*  INSERT INTO table\_name (column1, column2, column3,...) VALUES (value1, value2, value3,....) |
| INNER JOIN | SELECT column\_name(s) FROM table\_name1 INNER JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LEFT JOIN | SELECT column\_name(s) FROM table\_name1 LEFT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| RIGHT JOIN | SELECT column\_name(s) FROM table\_name1 RIGHT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| FULL JOIN | SELECT column\_name(s) FROM table\_name1 FULL JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LIKE | SELECT column\_name(s) FROM table\_name WHERE column\_nameLIKE pattern |
| ORDER BY | SELECT column\_name(s) FROM table\_name ORDER BY column\_name [ASC|DESC] |
| SELECT | SELECT column\_name(s) FROM table\_name |
| SELECT \* | SELECT \* FROM table\_name |
| SELECT DISTINCT | SELECT DISTINCT column\_name(s) FROM table\_name |
| SELECT INTO | SELECT \* INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name  *or*  SELECT column\_name(s) INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name |
| SELECT TOP | SELECT TOP number|percent column\_name(s) FROM table\_name |
| TRUNCATE TABLE | TRUNCATE TABLE table\_name |
| UNION | SELECT column\_name(s) FROM table\_name1 UNION SELECT column\_name(s) FROM table\_name2 |
| UNION ALL | SELECT column\_name(s) FROM table\_name1 UNION ALL SELECT column\_name(s) FROM table\_name2 |
| UPDATE | UPDATE table\_name SET column1=value, column2=value,... WHERE some\_column=some\_value |
| WHERE | SELECT column\_name(s) FROM table\_name WHERE column\_name operator value |

SELECT - 从数据库中提取数据

UPDATE - 更新数据库中的数据

DELETE - 从数据库中删除数据

INSERT INTO - 向数据库中插入新数据

CREATE DATABASE - 创建新数据库

ALTER DATABASE - 修改数据库

CREATE TABLE - 创建新表

ALTER TABLE - 变更（改变）数据库表

DROP TABLE - 删除表

CREATE INDEX - 创建索引（搜索键）

DROP INDEX - 删除索引

### SELECT DISTINCT

### INSERT INTO

INSERT INTO TABLE\_NAME VALUES(,,,)—全表插入

INSERT INTO TABLE\_NAME(,,,) VALUES(,,,)—指定插入

### 检测NULL

IS NULL/IS NOT NULL

### SELECT TOP

MY SQL-LIMIT

ORACLE-ROWNUM

* Select top number column\_name from table\_name where condition;
* Select column\_name from table\_name where condition limit number;
* Select column\_name from table\_name where condition and rownum <=number;

### SQL LIKE

%-零个或多个字符

\_-单个字符

[charlist]-定义要匹配的字符的集合和范围

[^charlist]或[!charlist]-定义不匹配字符的字符的集合和范围

Select \* from table\_name where column\_name like ‘[!a-c]’;

### JOIN

#### INNER JOIN/JOIN



#### LEFT JOIN



#### RIGHT JOIN



#### FULL JOIN



### UNION

相似数据结构组合，如果允许重复使用UNION ALL

### SELECT INTO

Select into复制一个表的数据插入到新表中

Select \*

Into new\_table\_name [in database]

From table\_name;

若只想复制结构，不复制值，使用

Select \*

Into newtable

From table\_name

Where 1=0;

### INSERT INTO SELECT

Insert into newtable(column)

Select column from table

Where condition;

### ALTER TABLE

ALTER TABLE TABLE\_NAME ADD COLUMN\_NAME DATATYPE;

ALTER TBALE TABLE\_NAME DROP COLUMN COLUMN\_NAME;

更改数据类型

ALTER TABLE TBALE\_NAME

ALTER /MODIFY COLUMN COLUMN\_NAME DATATYPE;

### AUTO INCREMENT

### VIEW

CREATE VIEW view\_name AS SELECT \* ;

DROP VIEW viewname;

WITH CHECK OPTION-检查条件

CREATE VIEW CUSTOMERS\_VIEW AS

SELECT name, age

FROM CUSTOMERS

WHERE age IS NOT NULL

WITH CHECK OPTION;

* + SELECT 子句不能包含 DISTINCT 关键字
  + SELECT 子句不能包含任何汇总函数（summary functions）
  + SELECT 子句不能包含任何集合函数（set functions）
  + SELECT 子句不能包含任何集合运算符（set operators）
  + SELECT 子句不能包含 ORDER BY 子句
  + FROM 子句中不能有多个数据表
  + WHERE 子句不能包含子查询（subquery）
  + 查询语句中不能有 GROUP BY 或者 HAVING
  + 计算得出的列不能更新
  + 视图必须包含原始数据表中所有的 NOT NULL 列，从而使 INSERT 查询生效。

### Date

|  |  |
| --- | --- |
| **函数** | **描述** |
| [NOW()](https://www.w3cschool.cn/mysql/func-now.html) | 返回当前的日期和时间 |
| [CURDATE()](https://www.w3cschool.cn/mysql/func-curdate.html) | 返回当前的日期 |
| [CURTIME()](https://www.w3cschool.cn/mysql/func-curtime.html) | 返回当前的时间 |
| [DATE()](https://www.w3cschool.cn/mysql/func-date.html) | 提取日期或日期/时间表达式的日期部分 |
| [EXTRACT()](https://www.w3cschool.cn/mysql/func-extract.html) | 返回日期/时间的单独部分 |
| [DATE\_ADD()](https://www.w3cschool.cn/mysql/func-date-add.html) | 向日期添加指定的时间间隔 |
| [DATE\_SUB()](https://www.w3cschool.cn/mysql/func-date-sub.html) | 从日期减去指定的时间间隔 |
| [DATEDIFF()](https://www.w3cschool.cn/mysql/func-datediff-mysql.html) | 返回两个日期之间的天数 |
| [DATE\_FORMAT()](https://www.w3cschool.cn/mysql/func-date-format.html) | 用不同的格式显示日期/时间 |

### SQL约束

NOT NULL 约束：保证列中数据不能有 NULL 值

DEFAULT 约束：提供该列数据未指定时所采用的默认值

UNIQUE 约束：保证列中的所有数据各不相同

主键约束：唯一标识数据表中的行/记录

外键约束：唯一标识其他表中的一条行/记录

CHECK 约束：此约束保证列中的所有值满足某一条件

索引：用于在数据库中快速创建或检索数据

删除约束：ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES\_PK;

#### SQL NOT NULL约束

P-id int NOT NULL;

#### SQL UNIQUE

UNIQUE(P-id);

#### SQL PRIMARY KEY约束

唯一标识。唯一，不能包含NULL。有且仅有一个

#### SQL FOREIGN KEY约束

一个表的forei key指向另一个表的primary key

#### SQL DEFAULT约束

#### SQL CHECK约束

CONSTRAINT chk\_Person CHECK (P\_Id>0 AND City='Sandnes')；

撤销约束

ALTER TABLE Persons

DROP CHECK chk\_Person

### SQL INTERSECT

返回重合部分

### SQL EXPECT

### SQL克隆表格

* SHOW CREATE TABLE table\_name
* CREATE TABLE clone\_name() ;
* INSERT INTO clone\_name() SELECT \* FROM table\_name;

### SQL INDEX

CREATE (UNIQUE) INDEX index\_name on table\_name(column\_name);

何时避免使用索引：

* 小的数据表不应当使用索引；
* 需要频繁进行大批量的更新或者插入操作的表；
* 如果列中包含大数或者 NULL 值，不宜创建索引；
* 频繁操作的列不宜创建索引。

### SQL子查询

* 子查询必须括在圆括号中。
* 子查询的 SELECT 子句中只能有一个列，除非主查询中有多个列，用于与子查询选中的列相比较。
* 子查询不能使用 ORDER BY，不过主查询可以。在子查询中，GROUP BY 可以起到同 ORDER BY 相同的作用。
* 返回多行数据的子查询只能同多值操作符一起使用，比如 IN 操作符。
* SELECT 列表中不能包含任何对 BLOB、ARRAY、CLOB 或者 NCLOB 类型值的引用。
* 子查询不能直接用在集合函数中。
* BETWEEN 操作符不能同子查询一起使用，但是 BETWEEN 操作符可以用在子查询中。

### SQL注入

* 模式匹配用户输入
* 转义那些对数据库有意义的特殊字符
* 要破解 LIKE 困境，必须有一种专门的转义机制，将用户提供的 '%' 和 '\_' 转换为字面值

### SQL HAVING子句

Having子句指定顾虑条件，对GROUP BY子句施加约束；

必须紧跟GROUP BY子句，在ORDER BY之前。

SELECT column1, column2

FROM table1, table2

WHERE [ conditions ]

GROUP BY column1, column2

HAVING [ conditions ]

ORDER BY column1, column2

## SQL事务

事务是任务序列，可以用户手工执行，也可由数据库程序自行执行。

### 属性

ACID-Atomicity\Consistency\Isolatin\Durability

#### 原子性

保证任务所有操作都执行完毕，否则全部回滚

#### 一致性

事务成功执行，数据库的状态得到正确的转变

#### 隔离性

不用事务相互独立、透明

#### 持久性

系统出现问题，之前执行成功的事务也会保留

### 事务控制

#### COMMIT

#### ROLLBACK

#### SAVEPOINT

创建还原点

SAVEPOINT SAVEPOINT\_NAME;

ROLLBACK TO SAVEPOINT\_NAME;

RELEASE SAVEPOINT;

#### SET TRANSACTION

命名事务，可以初始化事务，并制定事务的属性

SET TRANSACTION [ READ WRITE | READ ONLY ];

### 创建临时表

CREATE TEMPORARY TABLE;

## SQL函数

### SQL AGGREGATE函数

SQL MAX()/MIN()/COUNT()/AVG()/SUM()/FIRST()/LAST()

### SQL SCALAR函数

* UCASE() - 将某个字段转换为大写
* LCASE() - 将某个字段转换为小写
* MID() - 从某个文本字段提取字符
* LEN() - 返回某个文本字段的长度
* ROUND() - 对某个数值字段进行指定小数位数的四舍五入
* NOW() - 返回当前的系统日期和时间
* FORMAT() - 格式化某个字段的显示方式

### SQL日期函数

|  |  |
| --- | --- |
| 名称 | 描述 |
| ADDDATE() | 增加日期 |
| ADDTIME() | 增加时间 |
| CONVERT\_TZ() | 将当前时区更改为另一时区 |
| CURDATE() | 返回当前日期 |
| CURRENT\_DATE(), CURRENT\_DATE | CURDATE() 的别名 |
| CURRENT\_TIME(), CURRENT\_TIME | CURTIME() 的别名 |
| CURRENT\_TIMESTAMP(), CURRENT\_TIMESTAMP | NOW() 的别名 |
| CURTIME() | 返回当前时间 |
| DATE\_ADD() | 将两个日期相加 |
| DATE\_FORMAT() | 按照指定格式格式化日期 |
| DATE\_SUB() | 将两个日期相减 |
| DATE() | 从 date 或者 datetime 表达式中提取出日期部分 |
| DATEDIFF() | 将两个日期相减 |
| DAY() | DAYOFMONTH() 的别名 |
| DAYNAME() | 返回某天在用星期中的名称 |
| DAYOFMONTH() | 返回某天是当月的第几天 （1-31） |
| DAYOFWEEK() | 返回某天是该星期的第几天 |
| DAYOFYEAR() | 返回某天是一年中的第几天（1-366） |
| EXTRACT | 提取日期中的某一部分 |
| FROM\_DAYS() | 将天数转换为日期 |
| FROM\_UNIXTIME() | 将某个日期格式化为 UNIX 时间戳 |
| HOUR() | 提取小时 |
| LAST\_DAY | 返回参数日期所在月份的最后一天 |
| LOCALTIME(), LOCALTIME | NOW() 的别名 |
| LOCALTIMESTAMP, LOCALTIMESTAMP() | NOW() 的别名 |
| MAKEDATE() | 利用年份和某天在该年所处的天数来创建日期 |
| MAKETIME | MAKETIME() |
| MICROSECOND() | 由参数返回微秒 |
| MINUTE() | 由参数返回分钟 |
| MONTH() | 返回日期参数的月份 |
| MONTHNAME() | 返回月份的名字 |
| NOW() | 返回当前日期和时间 |
| PERIOD\_ADD() | 向年月格式的日期数据之间添加一段时间 |
| PERIOD\_DIFF() | 返回两个年月格式的日期数据之间的月份数 |
| QUARTER() | 返回日期参数所在的季度 |
| SEC\_TO\_TIME() | 将秒数转换为 'HH:MM:SS' 格式 |
| SECOND() | 返回参数中的秒数 (0-59) |
| STR\_TO\_DATE() | 将字符串转换为日期数据 |
| SUBDATE() | 以三个参数调用的时候是 DATE\_SUB() 的同义词 |
| SUBTIME() | 减去时间 |
| SYSDATE() | 返回函数执行的时的时刻 |
| TIME\_FORMAT() | 格式化时间 |
| TIME\_TO\_SEC() | 将时间参数转换为秒数 |
| TIME() | 返回参数表达式中的时间部分 |
| TIMEDIFF() | 将两个时间相减 |
| TIMESTAMP() | 只有一个参数时，该函数返回 date 或者 datetime 表达式。当有两个参数时，将两个参数相加。 |
| TIMESTAMPADD() | 在 datetime 表达式上加上一段时间 |
| TIMESTAMPDIFF() | 在 datetime 表达式上减去一段时间 |
| TO\_DAYS() | 将日期参数转换为天数 |
| UNIX\_TIMESTAMP() | 返回 UNIX 时间戳 |
| UTC\_DATE() | 返回当前 UTC 日期 |
| UTC\_TIME() | 返回当前 UTC 时间 |
| UTC\_TIMESTAMP() | 返回当前 UTC 日期和时间 |
| WEEK() | 返回参数的星期数 |
| WEEKDAY() | 返回日期参数时一个星期中的第几天 |
| WEEKOFYEAR() | 返回日期参数是日历上的第几周 (1-53) |
| YEAR() | 返回日期参数中的年份 |
| YEARWEEK() | 返回年份和星期 |

### SQL FIELD()

SELECT FIELD(STR,STR1,STR2,…);

在STR后面的STR1等寻找STR,找到返回索引，从1开始，若没找到返回0；

### SQL SQRT()

求方根

### SQL RAND()

生成0-1随机数

### SQL CONTACT()

将两个字符串连接成一个字符串

### SQL ISNULL()\NVL()\IFNULL()\COALESCE()

### SQL REPLACE()

replace(original-string，search-string，replace-string)

### SQL TRIM()

* MySQL: TRIM( ), RTRIM( ), LTRIM( )
* Oracle: RTRIM( ), LTRIM( )
* SQL Server: RTRIM( ), LTRIM( )

TRIM ( [ [位置] [要移除的字串] FROM ] 字串): [位置] 的可能值为 LEADING (起头), TRAILING (结尾), or BOTH (起头及结尾)。 这个函数将把 [要移除的字串] 从字串的起头、结尾，或是起头及结尾移除。如果我们没有列出 [要移除的字串] 是什么的话，那空白就会被移除。

LTRIM(字串): 将所有字串起头的空白移除。

RTRIM(字串): 将所有字串结尾的空白移除。

### SQL语句大全



## SQL反模式—BILL Karwin

### 解决树形结构的查询和存储

#### 路径枚举

定义一个path varchar(1000)用来保存路径如/1/2/4/7

获取子节点

Select \*from comment as c where c.path like ‘1/2/4’ || ‘%’

获得祖先路径

Select \* from comment as c where ‘1/2/4/7’ like c.path || ‘%’

#### 嵌套集

#### 闭包表

## 配置MySql数据库

### 安装MySql

1. 使用MySql官网的msi文件
2. 使用3306端口
3. MySql –u root –p登录
4. Show databases;

### 使用sqldbx连接本机数据库

1. 打开控制面板->管理工具->数据源



1. 数据源ODBC添加用户DSN





1. 注意检查安装的ODBC驱动的位数与数据源管理器的位数需要保持一致，C:\Windows\SysWOW64或者C:\Windows\System32下的ODBCad32.exe

### MySql存储机制

* MyISAM:对事务支持不好
* InnoDB：提供事务安全的存储

如果需要支持，需要在建表作业后加上ENGINE=MyISAM/InnoDB

### 常用指令

|-show databses;

|-create database [if not exits] databasename;

|-use databasename;

|-show tables;

|-desc tablename;

# LintCode实例

## A+B问题

|  |
| --- |
| int aplusb(int a, int b) {  // write your code here  return((a^b)+((a&b)<<1));  } |

# Github

## Config

可查询 ~/.gitconfig

### 设置姓名和邮箱地址

Git config –global user.name “”

Git config –global user.email “”

Git config –global color.ur true,显示颜色

### 设置私密文件

保存数据库的配置文件，每次提交会显示untracked files。

可在git工作区创建一个特殊的.gitignore文件，把要忽略的文件名填入

若想强制添加，可用-f

Git add –f name

使用git check-ignore来检查.gitignore文件

### 简写

|-st:status

|-co:checkout

|-ci:commit

|-br:branch

可以添加别名，git config --global alias.unstage 'reset HEAD'

git config --global alias.last 'log -1'

git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"

### 提高命令输出的可读性

Git config –global color.ui auto

### 设置SSH Key

ssh-keygen –t rsa –C “mail”

-t指定密钥类型，默认是rsa，可以省略

-C设置注释文字，比如邮箱

-f指定密钥文件存储文件名

id\_rsa文件是私有密钥

id\_rsa.pub是公开密钥

网页添加SSH-KEY，使用手机的私人密钥与GitHub进行认证和通信。

Ssh –T [git@github.com](mailto:git@github.com)

### 修改本地主目录路径

1. 修改起始位置
2. 删除目标后的cd-to-home，可实现bash启动默认指向该文件夹
3. 新建环境变量HOME
4. 将原目录下的.ssh\.gitconfig\.bash\_history等文件拷贝到新目录下即可





## Clone

Git clone [git@github.com:jiudc/Hello-World.git](mailto:git@github.com:jiudc/Hello-World.git)

Vi hello\_world.pht

i-Esc-：Wq

Git status

提交至代码库

Git add hello\_world.php

Git commit –m “Add hello world script by php”

Add加入暂存区，通过git commit命令提交

通过git log查看

## Git基础操作

### Git init—初始化仓库

生成.git目录，附属于该仓库的工作树

### Git status—查看仓库的状态

### Git add—向暂存区中添加文件

### Git commit—保存仓库的历史记录

#### 记述一行提交信息

使用-m参数

#### 记述详细提交信息

不适用-m参数，需要编辑

第一行：用一行文字简述

第二行：空行

第三行：记述更改的详细信息

### Git log—查看提交日志

只显示提交信息的第一行 –pretty=short

显示指定的目录、文件日志，git log filename/dirname

显示文件的改动,-p

### Git diff—查看更改前后的差别

Git diff,查看工作树和暂存区的差别

Git diff Head,查看工作树和最新提交的差别

### Git branch—显示分支一览表

Git branch –a, -a参数可以通水显示本地仓库和远程仓库的分支信息

Git branch

Git branch name

Git branch –d name（删除分支）

Git branch –D name（强制删除分支）

### Git checkout –b创建、切换分支

Git branch feature-A

Git checkout feature-A

### Git checkout –切回上一个分支

### Git merge—合并分支

Git merge –no-ff feature-A，no-ff表示禁用Fast forword（该模式下，删除分支，会丢掉分支信息）

### Git log—graph—以图表形式查看分支

### Git reset—回溯历史版本

Git reset –hard 哈希值

### Git reflog—查看当前仓库的操作历史

Git log只能查看以当前状态为终点的历史日志

### Git commit –amend—修改提交信息

### Git rebase –i—压缩历史

### Git commit –am “”

使用一步骤代替两部add和commit

### Git remote add—添加远程仓库

Git remote add orign [git@github.com:jiudc/git-tutorial.git](mailto:git@github.com:jiudc/git-tutorial.git)

Git会设置别名orign

### Git push—推送至远程仓库

Git push –u orign master,-u参数可以将orign仓库的master分支设置为本地仓库当前分支的upstream，添加了这个参数，将来运行git pull从远程仓库获取内容时，该分支可直接从orign的master分支获取内容。后面可通过git push orign master

### Git clone—获取远程仓库

Git clone git@...

Git checkout –b feature-D orign/feature-D,获取远程仓库的分支feature-D

### Git pull—获取最新的远程仓库的分支

Git pull origin feature-D

### Git remote –v—查看远端地址

### Git config –list—查看配置

Git add . //暂存所有的更改

Git checkout . //丢弃所有的更改

Git status //查看文件状态

Git commit – m “本次要提交的概要信息”



Ssh [git@github.com](mailto:git@github.com)测试连接



Ssh –T [git@github.com](mailto:git@github.com)测试连接



Ssh –t –p 22 [git@github.com](mailto:git@github.com)更换连接端口



### Git stash

将工作现场保存起来。

可用git stash list查看刚才的工作现场。可使用两种方式恢复现场：

|-git stash apply，恢复后，stash内容并不删除，需要使用git stash drop删除

|-git stash pop，恢复删除一步

### Git fetch--all

更新git remote中所有远程包含的最新的分支

## 多人协作

|-Git push orign dev,推送dev分支到远程，/origin/dev

|-git checkout –b dev orign/dev，建立远程origin的dev与本地的联系

|-git pull，若没有建立连接，则报错

|-git branch –set-upstream dev origin/dev

## 标签管理

1. 切换到需要打标签的分支
2. Git tag v1.0
3. 可以对commit id打上log，如git tag v0.9 6224937
4. 可用git tag命令查看
5. Git tag –m说明文字 –a标签名，可用-s用私钥签名
6. Git show <tagname>查看信息
7. Delete tag tagname
8. Git push origin tagname推送
9. Git push origin –tags，推送所有标签
10. 若需要删除已推送的标签，1）先删除本地标签git tag –d tagname 2）git push origin :refs/tags/tagnam

## 建立git服务器

1. 安装git：sudo apt-get install git
2. 创建git用户：sudo adduser git
3. 创建证书登录：收集所有需要登录的用户的公钥，就是他们自己的id\_rsa.pub文件，把所有公钥导入到/home/git/.ssh/authorized\_keys文件里，一行一个。
4. 初始化Git仓库：先选定一个目录作为Git仓库，假定是/srv/sample.git，在/srv目录下输入命令：sudo git init --bare sample.git  
   Git就会创建一个裸仓库，裸仓库没有工作区，因为服务器上的Git仓库纯粹是为了共享，所以不让用户直接登录到服务器上去改工作区，并且服务器上的Git仓库通常都以.git结尾。然后，把owner改为git：sudo chown -R git:git sample.git
5. 禁用ssh登录：通过编辑/etc/passwd  
   git:x:1001:1001:,,,:/home/git:/bin/bash 改为  
   git:x:1001:1001:,,,:/home/git:/usr/bin/git-shell
6. 克隆远程库：git clone
7. 管理公钥：Gitosis
8. 管理权限：Gitolite

## 快捷键

Shift+/显示快捷键

## Tips

### LF will be replaced by CRLF in hello\_world.php.

Windows中的换行符为CRLF，而linux下的换行符为LF，因而执行add会有该提示。解决方法是禁用自动转换

Git config –global core.autocrlf false

### Enter passphrase for key

去除密码。

Ssh-add

如果系统提示：could not open a connection to your authentication agent

Ssh-agent bash

然后执行ssh-add id\_rsa

# Linux

## 安装

### VMware安装

|-https://pan.baidu.com/s/1dFnkBrN

|-虚拟网络编辑器：nat

### Linux安装

|-Ubuntu server 64



需要启动如下服务：



## 改变linux配色方案

* git clone git://github.com/seebi/dircolors-solarized.git
* 将dircolors文件拷贝到~/.dircolors
* source ~/.bashrc

## Linux目录名

* /bin：二进制目录，存放许多用户级的GNU工具
* /boot：启动目录
* /dev：设备目录，Linux在这里创建设备节点
* /etc：系统配置文件目录
* /home：主目录，Linux在这里创建设备节点
* /lib：库目录
* /media：媒体目录，可移动媒体设备常用的挂载点
* /mnt：挂载目录
* /opt：可选目录，用于存放第三方软件包和数据文件
* /proc：进程目录，存放现有硬件及当前进程
* /root：root用户的主目录
* /sbin：系统二进制目录
* /run：运行目录，存放系统运行时的运行时数据
* /srv：服务目录，存放本地服务相关文件
* /sys：系统目录，存放系统硬件信息的相关文件
* /tmp：临时文件
* /usr：用户二进制文件
* /var：可变目录，可存放经常变化的文件，如日志

## 基础命令

区分大小写

* ls -F,区分文件夹文件,-R递归,-a全部，-l显示长列表,ls -alF
* ls -l my\_script，过滤匹配输出列表
* ls -l –time=atime，才可以显示文件的访问时间，否则默认显示修改时间
* ls -d只列出目录
* ls -i查看文件或者目录的inode编号
* mkdir -p,递归目录创建
* 创建文件：touch filename,也可用于改变修改日期，若想改变访问时间，可加上参数-a
* 复制：cp src dst，cp -i覆盖前会提示
* 递归复制：cp -R
* 链接文件：ln -s src dst，软链接
* ln src dst，硬链接，完全一样
* file能探测文件的内部，知晓文件的类型
* cat file，查看文件，-n行号，-b行数（不包括空行）
* cat -T不让制表符出现，一旦开始无法控制后面操作
* more显示文本文件，会在显示每页数据后停下来
* less，more的升级版
* 查看部分文件tail,默认情况显示文件的末尾10行
* tail -n 2 log\_file显示末尾2行，-f参数允许其他进程使用该文件时查看文件类容，hi保持活动状态，并不断显示添加到文件的内容。实时监测系统日志
* head，显示开头，head -5 log\_file
* 查看进程ps，默认情况下ps显示运行在当前控制台下的属于当前用户的进程
  + -A，显示所有进程
  + -N，显示与指定参数不符的所有进程
  + -a，显示除控制进程和无终端进程外的所有进程
  + -d，显示除控制进程外的所有进程
  + -e，显示所有进程
  + -C cmdlist，显示包含挂在cmdlist列表中的进程
  + -G grplist，显示组ID在grplist列表中的进程
  + -U userlist，显示属主的用户ID在userlist列表中的进程
  + -l，显示长列表
  + -f，显示完整格式输出
* ps -ef,ps -l,ps --forest
* top，实时显示进程
* kill pid，-s强终止
* killall 进程名
* mount 输出当前系统挂载的设备
* mount –t type directory，type指定了磁盘被格式化的文件系统，该命令需要root权限
  + vfat-Windows长文件系统
  + ntfs-高级文件系统
  + iso9660-标准CD-ROM文件系统
* mount参数
  + ro：以只读形式挂载
  + rw：读写
  + user：允许普通用户挂载文件系统
  + check=none：挂载文件系统的时候不进行完整性检查
  + loop：挂载一个文件
* umount：卸载，卸载的时候提示系统可用lsof /path/device/node获取它的进程信息
* df：可以查询已挂载磁盘的使用情况，-h参数可以按照用户易读的形式显示
* du：会显示当前目录下所有的文件、目录和子目录的磁盘使用情况
  + -c：显示所有已列出文件总的大小
  + -h：易读
  + -s：显示每个输出参数的总计
* sort：排序le
  + -b：排序时候忽略起始的空白
  + -C：不排序，如果数据无序也不要报告
  + -c：不排序，但检查输入数据是不是已排序；未排序报告
  + -d：仅考虑空白和字母，不考虑特殊字符
  + -f：默认情况下，会将大写字母排在前面；这个参数会忽略大小写
  + -g：按通用数值来排序，把值当做浮点数
  + -i：在排序时忽略不可打印字符
  + -k pos1 pos2：排序从pos1位置开始，如果指定了pos2，到pos2位置结束
  + -M：按照三字符月份名按月份排序
  + -m：将两个已排序的数据文件合并
  + -n：按字符串数值来排序，不转换为浮点数
  + -o file：将排序结果写出到指定的文件中
  + -R file：按随机生成的散列表的键值排序
  + -r：反序排序
  + -S size：指定使用内存的大小
  + -s：禁用最后重排序比较
  + -T dir：指定一个位置来存储临时工作文件
  + -t sep：指定一个用来区分键位置的字符
  + -u：和-c参数一起使用，检查严格排序；当不和-c一起使用，仅输出第一例相似的两行
  + -z：用NULL字符作为行尾，而不是用换行符
* 搜索数据：grep pattern file
  + -v：方向搜索，输出不匹配该模式的行
  + -n：显示匹配模式的行所在的行号
  + -c：只需要知道多少行匹配
  + -e：若需要制定多个匹配模式，可用-e来制定每个模式：grep –e t –e f file
* 压缩数据
  + gzip：压缩文件，.gz
  + gzcat：用来查看压缩过的文本文件的内容
  + gunzip：用来解压文件
* 归档数据：tar –cvf –tf –xvf(.tar) -xzvf(.tgz)
  + -A：将一个已有tar归档文件追加到另一个已有tar归档文件
  + -c：创建一个新的tar归档文件
  + -d：检查归档文件和文件系统的不用之处
  + -r：追加文件到已有tar归档文件结尾
  + -u：将比tar归档文件中已有的同名文件新的文件追加到该tar归档文件中
  + -x：从已有tar归档文件中提取文件
  + -C dir：切换到指定目录
  + -f file：输出结果到文件或设备file
  + -j：将输出重定向给bzip2命令来压缩内容
  + -p：保留所有文件权限
  + -v：在处理文件时显示文件
  + -z：将输出重定向给gzip命令来压缩内容
* 进程列表：(pwd; ls; cd/etc; echo $BASH\_SUBSHELL)，若echo命令返回零，则表示没有子shell，若大于零，则表明存在shell。{}这个花括号不会创建子shell而[]会
* sleep 10&:睡眠10s
* jobs：显示当前运行在后台的作业 jobs -l还可显示命令的PID
* 后台运行：(tar \*)&
* 协程：在后台生成一个shell，并在这个shell中执行命令。coproc My\_job { sleep 10; }

## 内建命令

### 外部命令

* which ps
* type -a ps

当外部命令执行时，会创建出一个子进程。这种操作成为衍生(forking)

### 内建命令

不需要子进程来执行，如cd、exit

命令有多种形式，要查看命令的不同实现，使用type命令的-a选项，which只显示外部命令文件

* history：查看历史命令
* ！！：会从shell历史记录中换回的命令，然后执行该命令
* history -a：在打开的会话向.bash\_history添加记录
* history -n：要强制读取.bah\_history文件，可以使用history -n命令
* ！n：可用于执行该行指令
* alias -p：查看当前可用别名
* alias li=’ls -li’;

## 环境变量

### 全局环境变量

* printenv HOME
* env $HOME
* 创建全局变量：my\_variable = ”I am Global now”; export my\_variable
* 子shell无法使用export改变父shell中全局变量的值

### 局部环境变量

* set：显示特定进程设置的所有环境变量，包括局部变量、全部变量和用户定义变量
* echo $my\_variable; my\_variable=”hello world‘’

### 删除环境变量

* unset my\_variable

### 设置PATH环境变量

* PATH=$PATH:dir,对PATH的修改只能持续到推出或者重启系统

### 永久改变环境变量

* 登录shell
  + /etc/profile
  + $HOME/.bash\_profile
  + $HOME/.bashrc
  + $HOME/.bash\_login
  + $HOME/.profile
  + $HOME下，shell会按照以上黄色标记顺序执行，运行第一个找到的文件，其余会被忽略
* 交互式shell：不是登录系统时启动的，检查HOME目录下的.bashrc文件
* 非交互式shell：BASH\_ENV环境变量

### 数组变量

* mytest=(ont two three)
* echo ${mytest[\*]}
* **获取数组长度**
  + arr\_length=${#arr\_number[\*]}或${#arr\_number[@]}均可，即形式：${#数组名[@/\*]}
* **读取某个下标的值**
  + arr\_index2=${arr\_number[2]}，即形式：${数组名[下标]}
* **对某个下标赋值**
  + 如果该下标元素已经存在，会修改该下标的值为新的指定值。
  + 如果指定的下标已经超过当前数组的大小，新赋的值被追加到数组的尾部。
* **删除操作**
  + 清除某个元素：unset arr\_number[1]，这里清除下标为1的数组；
  + 清空整个数组：unset arr\_number;
* **分片访问**
  + 分片访问形式为：${数组名[@或\*]:开始下标:结束下标}，注意，不包括结束下标元素的值。
* **模式替换**
  + 形式为：${数组名[@或\*]/模式/新值},例如：${arr\_number[@]/2/98}
* **数组的遍历**
  + for v in ${arr\_number[@]}; do

　　　　　echo $v;

　　done

## 文件权限

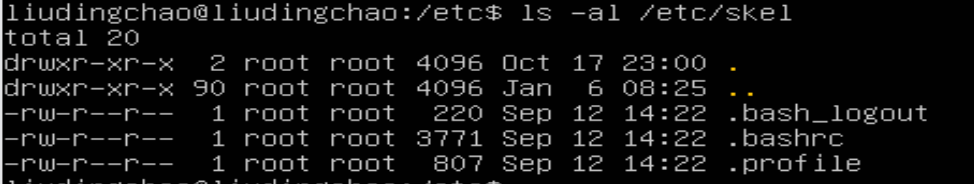
### /etc/passwd

* 登录用户名
* 用户密码
* 用户账户的UID(数字形式)
* 用户账户的组ID(GID)(数字形式)
* 用户账户的文本描述(称为备注字段)
* 用户HOME目录的位置
* 用户的默认shell

### /etc/shadow

* 与/etc/passwd文件中的登录名字段对应的登录名
* 加密后的密码
* 自上一次修改密码后过去的天数密码（自1970年1月1日开始计算）
* 多少天数才能更改密码
* 多少天后必须修改密码
* 密码过期前提前多少天提醒用户更改密码
* 密码过期后多少天禁用用户密码
* 用户账户被禁用的日期
* 预留字段给将来使用

### 添加新用户

* /usr/sbin/useradd -D:显示默认值  
  
  + 新用户会被添加到GID为100的公共组
  + 新用户的HOME目录将会位于/home/loginname
  + 新用户账户密码在过期后不会被禁用
  + 新用户账户未被设置过期日期
  + 新用户账户将bash shell作为默认shell
  + 系统会将/etc/skel目录下的内容复制到用户的HOME目录下  
    
  + 系统为该用户账户在mail目录下创建一个用于接收邮件的文件

使用useradd命令默认不产生home目录

要在创建用户时改变默认值，可以使用命令行参数

* -c comment：给新用户添加备注
* -d home\_dir
* -e expire\_date
* -f inactive\_days
* -g initial\_group
* -G group …：指定用户除登录组之外所属的一个或多个附加组
* -k 必须和-m一起使用，将/etc/skel目录的内容复制到用户的HOME目录
* -m 创建用户的HOME目录
* -M 不创建用户的HOME目录（当默认设置里要求创建时才使用这个选项）
* -n 创建一个与用户登录名的新组
* -r 创建系统账户
* -p passwd
* -s shell
* -u uid

修改系统默认值，可以在-D选项后跟上一个指定的值来修改系统默认的新用户设置

### 删除用户

* userdel -r，会同时删除用户目录

### 修改用户

* usermod
  + -l：修改用户账户的登录名
  + -L：锁定账户，使用户无法登录
  + -p：修改账户的密码
  + -U：解除锁定，使用户能够登录
* passwd
* chpasswd：从文件中读取登录名密码对，并更新密码
* chage：修改密码的过期日期
* chfn：修改用户账户的备注信息
* chsh：修改用户账户的默认登录shell

### 使用Linux组

* /etc/group文件
  + 组名
  + 组密码
  + GID
  + 属于改组的用户列表
* 创建新组：groupadd，创建新组时，默认没有用户被分配到改组。可用usermod命令  
  usermod -G shared rich，-g指定的组名会替换掉该账户的默认组。-G将改组添加到用户的数组的列表里，不会影响默认值。
* 修改组：-g GID、-n 组名

### 理解文件权限

* -：代表文件
* d：代表目录
* l：链接
* c：字符型设备
* b：块设备
* n：代表网络设备
* r\w\x
* 对象的属主、对象的属组、系统其它用户
* 默认的文件权限：umask

### 改变权限

* chmod options mode file
  + u:用户
  + g：组
  + o：其它
  + a：代表上述所有
* chown：更改宿主

|-sudo su登录高级管理员

|-关机，init 0（登录高级管理员）

|-sudo init 0（正常用户）

-h/--help，帮助文档

Ls - -help |more(翻页，spaces)

CTRL+C，强制停止，可多次键入

CTRL+D，退出当前用户（exit）

CD + /,回到根目录

CD，回到当前用户的根目录

Find / -name “”,从根目录查找,find .当前目录，find ./当前目录下的所有目录

Mkdir、rmdir、rm –rf

cp –R 源文件 目标位置

mv 移动、重命名

cat 查看文件 cat doc |more

tail 查看文件，从尾开始，默认后10行

tail –f 实时滚动 tail –f user.log |grep “2018-04-15” |more

head 查看文件，从头开始，默认头10行， head -20 doc

df –h 查看磁盘情况

du –sh 统计查看当前目录的使用情况du –sh \*，详细

alias eric =’ ls –l –a”，别名，unalias,取消别名

ls -alh

## VI

三种操作模式：命令模式、输入模式、末行模式

|-vi newfile

|-q!

|-wq!  
|-X

|-i

|-o当前行下一行插入

|-O 当前行的上一行插入

|-x 删除当前字符

|-h、l、k、j 左右上下

|-ctrl+f，向下翻整页 u半页

|-ctrl+b，向上翻整页 d半页

|-shift+4，跳到行首

|-shift+6，跳到行尾

|-w，跳转到当前贯标位置所在的后一个单子的首字符，b-前，e-后一个单词的尾字符

|-set nu,在编辑器中显示行号，set nonu

|-gg 跳到文件顶部

|-G 跳到文件尾部

|-数字+G 跳到某一行

|-dd删除整行/剪切

|-yy 赋值当前行  
|-p 粘贴

|-u 撤销当前操作

|-U 取消对当前行的所有操作

|-ctrl+r 对使用u命令撤销的操作进行恢复

|-dw 删除当前字符到单词尾，包括空格，de不包括空格，yw,ye

|-d+shifit+4 删除当前字符到行尾，d+shift+6到行首,y+shift+4,y+shift+6

|-J删除行尾的换行符，相当于合并前行和后行

|-\向下查找，下一个n

|-?向上查找，上一个N

|-:s/old/new 将当前行的第一个字符old替换成new

|-:s/old/new/g 将当前行的所有字符old替换成new

|-:#,#s/old/new/g 在行号#,#范围内替换所有的字符串

|-%s/old/new/g 全文范围内替换

|-s/old/new/c 加上c表示替换前用户确认

## Touch命令

Touch [options] filename

-a:改变访问时间

-m:改变改动时间

-t timestamp:改变范文时间和改动时间为timestamp

## Mkdir命令

Mkdir [options] dirname

-p:递归创建文件夹

Mkdir –p zsy/hxn

-mmode:新建文件夹，并设置文件夹的文件访问模式

Mkdir –m770 zsy

## Rm命令

Rm [options] filename/dirname

-f:强制删除

-r:递归删除

-i：删除前确认

## 构建基本脚本

### 创建shell脚本文件

* 在创建shell脚本文件时，必须在文件的第一行指定要使用的shell，格式为 #!/bin/bash
* testing = `date`与testing=$(date)
* 输出重定向>,>>追加
* 输入重定向<,<<
* expr，处理数学表达式或者$[operation ]，只能处理整数
* 浮点解决方案:bc
  + bc -q:不显示bash计算器冗长的欢迎信息
  + scale
  + 在脚本中使用bc：var1=$(echo “scale=4;3.44/5”|bc)
  + variable =$(bc<<EOF  
    options  
    statements  
    expressions  
    EOF  
    )

### 退出脚本

* Linux退出状态码
  + 0：命令成功结束
  + 1：一般位置错误
  + 2：不适合的shell命令
  + 126：命令不可执行
  + 127：没找到命令
  + 128：无效的推出参数
  + 128+x：与Linux信号x相关的严重错误
  + 130：通过Ctrl+C终止的命令
  + 255：正常范围之外的退出状态码

## 结构化命令

### if-then

if后面的命令如果返回值为零，则执行then

* if command  
  then  
   commands  
  fi
* if command; then  
   commands  
  fi

### if-then-else

* if command  
  then  
   commands  
  else  
   commands  
  fi

### test

* test condition
  + 数值比较
    - -eq
    - -ge
    - -gt
    - -le
    - -lt
    - -ne
  + 字符串比较,>或者<需要使用转义符
    - =
    - !=
    - <
    - >
    - -n str1:检查str1的长度是否非零
    - -z str1:检查str1的长度是否为
  + 文件比较
    - -d file：检查file是否存在并是一个目录
    - -e file：检查file是否存在
    - -f file：检查file是否存在并是一个文件
    - -s file：检查file是否存在并非空
    - -r file：检查file是否存在并可读
    - -w file：可写
    - -x file：可执行
    - -O file：检查file是否存在并属当前用户素有
    - -G file：检查file是否存在并且默认组与当前数组先攻
    - file1 -nt file2：检查file1是否比file2新
    - file1 -ot file2：旧

### 复合条件测试

* [condition1 ] && [condition2]
* [condition1 ] || [condition2]

### if-then高级特性

* (( expression ))，使用高级数学表达式
  + val++
  + val--
  + ++val
  + --val
  + !：逻辑求反
  + ~：位求反
  + \*\*
  + <<
  + >>
  + &
  + |
  + &&
  + ||
* [[ expression ]],提供了针对字符串比较的高级特性，模式匹配

### case

* case variable in  
  pattern1 | pattern2) commands1;;  
  pattern3) commands2;;  
  \*) default commands;;  
  esac

### for

* for var in list  
  do   
   commands  
  done

### 字符串拼接

* list = $list” Connecticut”:用于拼接

### 更改字符串分隔符

* IFS=$’\n’
* 可临时修改分隔符代码：  
  IFS.OLD=$IFS  
  IFS=$’\n’  
  <在代码中使用新的分隔符>  
  IFS=$IFS.OLD

### 通配符读取目录

* for file in /home/rich/test/\*
* if [-d “$file” ]:使用双引号可以支持目录中有空格

### C语言风格的for命令

* for(( i=1; i<=10;++ ))  
  do   
   statement  
  done

### while命令

* while test command  
  do  
   other commands  
  done

### until命令

* until test commands  
  do  
   other commands  
  done

### break

* break
* break n;1-当前循环；2-停止下一级的外部循环

### continue

* continue
* continue n

### read

* -a 后跟一个变量，该变量会被认为是个数组，然后给其赋值，默认是以空格为分割符。
* -d 后面跟一个标志符，其实只有其后的第一个字符有用，作为结束的标志。
* -p 后面跟提示信息，即在输入前打印提示信息。
* -e 在输入的时候可以时候命令补全功能。
* -n 后跟一个数字，定义输入文本的长度，很实用。
* -r 屏蔽\，如果没有该选项，则\作为一个转义字符，有的话 \就是个正常的字符了。
* -s 安静模式，在输入字符时不再屏幕上显示，例如login时输入密码。
* -t 后面跟秒数，定义输入字符的等待时间。
* -u 后面跟fd，从文件描述符中读入，该文件描述符可以是exec新开启的。

read -r userid name:自动读取下一行，读取完返回FALSE

## 处理用户输入

### 命令行参数

* $0,$1,${10}

### 读取脚本名

* name =$(basename $0)

### 参数统计

* $#:命令行参数个数
* ${!#}：返回最后一个变量名
* $\*：将变量当成单个参数
* $@：会单独处理每个参数
* shift：左移，$1会丢失，$0不变
* shift n
* #：去掉左边
* %：去掉右边
* 单一符号是最小匹配，两个符号是最大匹配
* ${file:0:5}：提取最左边的5个字节
* 对变量值的字符串做替换：
  + ${file/dir/path}：将一个dir替换为path
  + $(file//dir/pah)：将全部dir替换为path
  + ${file#\*/}：删掉第一个 / 及其左边的字符串
  + ${file##\*/}：删掉最后一个 / 及其左边的字符串
  + ${file#\*.}：删掉第一个 . 及其左边的字符串
  + ${file##\*.}：删掉最后一个 . 及其左边的字符串
  + ${file%/\*}：删掉最后一个 / 及其右边的字符串
  + ${file%%/\*}：删掉第一个 / 及其右边的字符串
  + ${file%.\*}：删掉最后一个 . 及其右边的字符串
  + ${file%%.\*}：删掉第一个 . 及其右边的字符串
* $$：本身的PID
* $!：shell运行的最后的后台process的id
* $\*：”$1 $2 …$n”输出所有的参数
* $@：”$1” … “$n”
* $#：参数个数

## Debug

### Segmentation fault(core dumped)

* 在gcc编译的程序时使用”-g”选项，使编译文件带有调试信息
* 检查Linux设置的在发生段错误时文件大小
  + ulimit -a
* 默认不产生，取消该限制
  + ulimit -c unlimited
* 用gdb分析core文件
  + gdb {executable} {dump file}
* bt：查看出错的堆栈(backtrace)
* frame {num}：分析每个栈桢中变量情况
* info locals：查看局部变量
* info args：查看参数
* where：查看调用参数

## Nmon监控工具

### 非交互模式

* -f：切换到非交互模式，必须是第一个参数
* -s：每多少秒采集一次数据
* -c：总共采集多少次数据
* -t：包括Top进程的状态
* -m：指定生成的文件目录

可用于生成尾号为.nmon的CSV文件，可使用nmon\_analyser把nmon生成的文件转换为excel。

<http://www.ibm.com/developerworks/wikis/display/Wikiptype/nmonanalyser>

# DB2

## 错误纠察

### SQL6048N

* 停止：db2admin stop/db2stop
* 修改hostname先不要重启
* 利用db2set –all可以显示db2的环境变量
* 再利用db2set –g DB2SYSTEM = <NEW-HOSTNAME>
* 修改ibm\SQLLIB\DB2中的db2nodes.cfg的host
* Restart
* 若报错"SQL1042C发生意外的系统错误"，查看db2diagl.log文件，会发现这样一个错误提示"帐户名与安全标识间无任何映射完成"，引用

db2extsec -a <new-hostname>/DB2ADMNS -u <new-hostname>/DB2USERS

命令来完成用户与新计算机名的映射

* 启动db2

# Scala

## 基本语法

* 区分大小写
* 类名：对于所有的类名的第一个字母要大写
* 方法名称：所有的方法名称的第一个字母要小写
* 程序文件名：程序文件的名称应该与对象名称完全匹配（新版本不需要）
* def main(args:Array[String])

## 包

### 定义包

* package

### 引用

Import语句可以出现在任何地方，而不是只能在文件顶部。如果想要引入包中的几个成员，可以使用selector（选取器）

* import java.awt.Color
* import java.awt.\_
* import java.awt.(Color, Font)
* import java.util.{HashMap=>JavaHashMap}//重命名成员
* import java.util.{HashMap=>\_,\_}//引入util包的所有成员，但是HashMap被隐藏了

注意：默认情况下，Scala总会引入java.lang.\_、scala.\_、Predef.\_。

## Scala数据类型

* Byte：-128到127
* Short：-32768到32767
* Int
* Long
* Float
* Double
* Char
* String
* Boolean
* Unit
* Null：scala.Null
* Nothing：任何其他类型的子类型
* Any：所有其他类的超类
* AnyRef：所有引用类的基类

### 符号字面量

‘x’被映射成预定义的Symbol的实例，scala.Symbol(“x”)

### 字符字面量

用单引号定义‘a’、‘\u0041’、‘\n’

### 字符串字面量

用双引号定义，多行字符串用三个双引号表示

## Scala变量

### 变量申明

* 变量：var
* 常量：val

### 变量类型申明

val\var VariableName : DataType [= Initial Value]

## 访问修饰符

* Private：比java严格，仅在成员定义的类或对象内部可见
* Protected：比java严格，只允许保护成员在定义了该成员的类的子类中被访问
* 作用域保护：private[x]、protected[x]，这个成员处理对[…]中的类或[…]的包中类及他们的伴生对象可见外，对其他所有类都是private。

## 运算符

* 算数运算符：+、-、\*、/、%
* 关系运算符：==、!=、>、<、>=、<=
* 逻辑运算符：&&、||、！
* 位运算：&、|、^、~、<<、>>、>>>

## 方法和函数

Scala方法(def)是类的一部分，而函数(val)是一个对象可以赋值给一个变量，其实是继承了Trait类的对象。

* 方法定义：def functionName ([参数列表]):[return type]={  
   function body  
   return [expr]  
  }
* 函数传名调用：使用=>符号来设置传名调用
* 指定函数参数名：指定函数参数名，不需要按照顺序向函数传递参数
* 可变参数：def printStrings(args:String\*)
* 递归函数：函数可以调用自身
* 默认参数值：可以为函数参数指定默认值，可以不传递参数，若传递则会取代默认值
* 高阶函数：操作其他函数的函数，def apply(f:Int=>String,v:int)=f(v)
* 函数嵌套：函数类定义函数，局部函数
* 匿名函数：箭头左边是参数列表，右边是函数体。Val inc=(x:Int)=>x+1
* 偏应用函数：不需要提供函数需要的所有参数，只提供部分，或不提供所需参数，使用\_替换确实的参数列表，val logWithDateBound=log(date,\_:String)
* 函数柯里化：是指将原来接受两个参数的函数编程新的接受一个参数的函数的过程。新的函数返回一个以原有第二个参数为参数的函数。

## 闭包

闭包是一个函数，返回值依赖于申明在函数外部的一个或多个变量。如：

var factor=3

var multiplier=(i:Int)=>i\*factor

## 字符串

Scala本身没有字符串，实际上是Java String。

* char charAt(int index)

## 数组

* 申明：var z:Array[String]=new Array[String](3)
* 多维数组：var myMatrix=ofDim[Int](3,3)
* 数组方法：import Array.\_

|  |
| --- |
| **def apply( x: T, xs: T\* ): Array[T]**  创建指定对象 T 的数组, T 的值可以是 Unit, Double, Float, Long, Int, Char, Short, Byte, Boolean。 |
| **def concat[T]( xss: Array[T]\* ): Array[T]**  合并数组 |
| **def copy( src: AnyRef, srcPos: Int, dest: AnyRef, destPos: Int, length: Int ): Unit**  复制一个数组到另一个数组上。相等于 Java's System.arraycopy(src, srcPos, dest, destPos, length)。 |
| **def empty[T]: Array[T]**  返回长度为 0 的数组 |
| **def iterate[T]( start: T, len: Int )( f: (T) => T ): Array[T]**  返回指定长度数组，每个数组元素为指定函数的返回值。  以上实例数组初始值为 0，长度为 3，计算函数为**a=>a+1**：  scala> Array.iterate(0,3)(a=>a+1)  res1: Array[Int] = Array(0, 1, 2) |
| **def fill[T]( n: Int )(elem: => T): Array[T]**  返回数组，长度为第一个参数指定，同时每个元素使用第二个参数进行填充。 |
| **def fill[T]( n1: Int, n2: Int )( elem: => T ): Array[Array[T]]**  返回二数组，长度为第一个参数指定，同时每个元素使用第二个参数进行填充。 |
| **def ofDim[T]( n1: Int ): Array[T]**  创建指定长度的数组 |
| **def ofDim[T]( n1: Int, n2: Int ): Array[Array[T]]**  创建二维数组 |
| **def ofDim[T]( n1: Int, n2: Int, n3: Int ): Array[Array[Array[T]]]**  创建三维数组 |
| **def range( start: Int, end: Int, step: Int ): Array[Int]**  创建指定区间内的数组，step 为每个元素间的步长 |
| **def range( start: Int, end: Int ): Array[Int]**  创建指定区间内的数组 |
| **def tabulate[T]( n: Int )(f: (Int)=> T): Array[T]**  返回指定长度数组，每个数组元素为指定函数的返回值，默认从 0 开始。  以上实例返回 3 个元素：  scala> Array.tabulate(3)(a => a + 5)  res0: Array[Int] = Array(5, 6, 7) |
| **def tabulate[T]( n1: Int, n2: Int )( f: (Int, Int ) => T): Array[Array[T]]**  返回指定长度的二维数组，每个数组元素为指定函数的返回值，默认从 0 开始。 |

# Spark

## Spark组成

Spark(BDAS)：全称伯克利数据分析栈，通过大规模集成算法、机器、人之间展现大数据应用的平台。

* SparkCore
* SparkSQL
* SparkStreaming
* MLib
* GraphX
* BlinkDB
* Tachyon

## RDD

Resilient Distributed DataSet，弹性分布式数据集。代表一个不可变、可分区、里面的元素可并行计算的集合。

### RDD属性

* 一组分片(Partition)，数据集的基本组成单位。每个分片会被一个计算任务处理，并决定并行计算的粒度。可在创建RDD时指定RDD的分片个数，没有指定则会采用默认值。默认值为程序所分配到的CPU Core的数目。
* 一个计算每个分区的函数。Spark中RDD的计算是以分片为单位的，每个RDD都会实现computer函数已达到这个目的。
* RDD之间的依赖关系。
* 一个Partitioner，即RDD的分片函数。只有基于key-value的RDD，才会有Partitioner，非key-value的RDD的 的值是None。Partitioner不但决定了RDD本身的分片数量，也决定了parent RDD Shuffle输出时的分片数量。
  + 基于哈希的HashPartitioner
  + 基于范围的RangePartitioner
* 一个列表，存储存取每个Partition的有限位置。对于一个HDFS文件来说，这个列表保存的是每个Partition所在块的位置。Spark“移动数据不如移动计算”的理念，在进行任务调度时，会尽可能将计算任务分配到其所要处理数据块的存储位置。

### RDD创建方式

* 通过读取文件生成：sc.texeFile(PATH)
* 通过并行化的方式创建RDD：sc.parallelize(array)
* 读取数据库等其他操作

### API

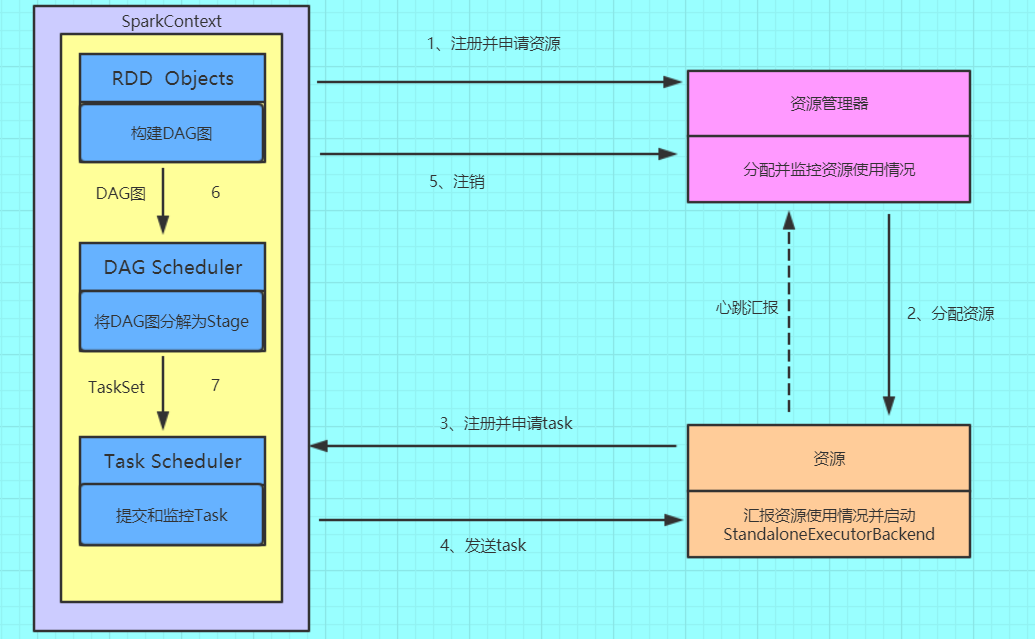
* Transformation：
  + map(func): 返回一个新的RDD，该RDD由每一个输入元素经过func函数转换后组成
  + filter(func): 返回一个新的RDD，该RDD由经过func函数计算后返回值为true的输入元素组成
  + flatMap(func): 类似于map，但是每一个输入元素可以被映射为0或多个输出元素（所以func应该返回一个序列，而不是单一元素）
  + mapPartitions(func): 类似于map，但独立地在RDD的每一个分片上运行，因此在类型为T的RDD上运行时，func的函数类型必须是Iterator[T] => Iterator[U]
  + mapPartitionsWithIndex(func): 类似于mapPartitions，但func带有一个整数参数表示分片的索引值，因此在类型为T的RDD上运行时，func的函数类型必须是(Int, Interator[T]) => Iterator[U]
  + sample(withReplacement, fraction, seed): 根据fraction指定的比例对数据进行采样，可以选择是否使用随机数进行替换，seed用于指定随机数生成器种子
  + union(otherDataset): 对源RDD和参数RDD求并集后返回一个新的RDD
  + intersection(otherDataset): 对源RDD和参数RDD求交集后返回一个新的RDD
  + distinct([numTasks])): 对源RDD进行去重后返回一个新的RDD
  + groupByKey([numTasks]): 在一个(K,V)的RDD上调用，返回一个(K, Iterator[V])的RDD
  + reduceByKey(func, [numTasks])：在一个(K,V)的RDD上调用，返回一个(K,V)的RDD，使用指定的reduce函数，将相同key的值聚合到一起，与groupByKey类似，reduce任务的个数可以通过第二个可选的参数来设置
  + aggregateByKey(zeroValue,seqOp, combOp, [numTasks])：先按分区聚合，再总聚合 。每次要跟初始值交流 例如：aggregateByKey(0)(\_+\_,\_+\_) 对k/y的RDD进行操作。
    - zeroValue:初始值
    - seqOp：combine的聚合逻辑，针对每个分区的操作
    - combOp：reduce端大聚合的逻辑
  + sortByKey([ascending], [numTasks])：在一个(K,V)的RDD上调用，K必须实现Ordered接口，返回一个按照key进行排序的(K,V)的RDD
  + sortBy(func,[ascending], [numTasks]):与sortByKey类似，但是更灵活。第一个参数是根据什么排序 ，第二个是怎么排序（false倒序）。第三个排序后分区数 ，默认与原RDD一样
  + join(otherDataset, [numTasks])：在类型为(K,V)和(K,W)的RDD上调用，返回一个相同key对应的所有元素对在一起的(K,(V,W))的RDD  相当于内连接（求交集）
  + cogroup(otherDataset, [numTasks])：在类型为(K,V)和(K,W)的RDD上调用，返回一个(K,(Iterable<V>,Iterable<W>))类型的RDD
  + cartesian(otherDataset)：两个RDD的笛卡尔积  的成很多个K/V
  + pipe(command, [envVars])：调用外部程序
  + coalesce(numPartitions,shuffle): 重新分区。第一个参数是要分多少区，第二个参数是否shuffle，默认false 。少分区变多分区true，多分区变少分区false。
    - coalesce(shuffle=true)==repartition()
    - 不经过shuffle不可能使分区增多
  + repartition(numPartitions): 重新分区必须shuffle。参数是要分多少区、少变多
  + repartitionAndSortWithinPartitions(partitioner)：重新分区+排序 。比先分区再排序效率高，对K/V的RDD进行操作
  + foldByKey(zeroValue,seqOp)：该函数用于K/V做折叠，合并处理 ，与aggregate类似。第一个括号的参数应用于每个V值  第二括号函数是聚合例如：\_+\_
  + combineByKey(createCombiner, mergeValue, mergeCombiners)：合并相同的key的值 rdd1.combineByKey(x => x, (a: Int, b: Int) => a + b, (m: Int, n: Int) => m + n)
    - createCombiner:给定一个初始值，用函数生成初始值。作用于每一个组的第一个元素上
    - combine:combine端聚合逻辑，在每个分区类进行
    - reduce:reduce端聚合逻辑，会把相同的key的数据拉到一个节点，并进行分组，在不同分区间进行
  + partitionBy(partitioner): 对RDD进行分区  partitioner是分区器
  + cache\persist: RDD缓存，可以避免重复计算从而减少时间，区别：cache内部调用了persist算子，cache默认就一个缓存级别MEMORY-ONLY，而persist则可以选择缓存级别
  + subtract（rdd）: 返回前rdd元素不在后rdd的rdd
  + leftOuterJoin: leftOuterJoin类似于SQL中的左外关联left outer join，返回结果以前面的RDD为主，关联不上的记录为空。只能用于两个RDD之间的关联，如果要多个RDD关联，多关联几次即可。
  + rightOuterJoin: rightOuterJoin类似于SQL中的有外关联right outer join，返回结果以参数中的RDD为主，关联不上的记录为空。只能用于两个RDD之间的关联，如果要多个RDD关联，多关联几次即可
  + subtractByKey: substractByKey和基本转换操作中的subtract类似只不过这里是针对K的，返回在主RDD中出现，并且不在otherRDD中出现的元素
* Action:
  + reduce(func): 通过func函数聚集RDD中的所有元素，这个功能必须是可交换且可并联的
  + collect()：在驱动程序中，以数组的形式返回数据集的所有元素
  + count()：返回RDD的元素个数
  + first()：返回RDD的第一个元素（类似于take(1)）
  + take(n)：返回一个由数据集的前n个元素组成的数组
  + takeSample(withReplacement,num, [seed])：返回一个数组，该数组由从数据集中随机采样的num个元素组成，可以选择是否用随机数替换不足的部分，seed用于指定随机数生成器种子
  + takeOrdered(n, [ordering])：按自然顺序或者指定的排序规则返回前n个元素
  + saveAsTextFile(path)：将数据集的元素以textfile的形式保存到HDFS文件系统或者其他支持的文件系统，对于每个元素，Spark将会调用toString方法，将它装换为文件中的文本
  + saveAsSequenceFile(path) ：将数据集中的元素以Hadoop sequencefile的格式保存到指定的目录下，可以使HDFS或者其他Hadoop支持的文件系统。
  + saveAsObjectFile(path) ：用于将RDD中的元素序列化成对象，存储到文件中。
  + countByKey()：针对(K,V)类型的RDD，返回一个(K,Int)的map，表示每一个key对应的元素个数。
  + foreach(func)：在数据集的每一个元素上，运行函数func进行更新。
  + aggregate
  + reduceByKeyLocally：该函数将RDD[K,V]中每个K对应的V值根据映射函数来运算，运算结果映射到一个Map[K,V]中，而不是RDD[K,V]
  + lookup(k)：作用于K-V类型的RDD上，返回指定K的所有V值
  + top(n)：按默认或者指定的排序规则返回前n个元素，默认按降序输出
  + fold
  + foreachPartition：这个函数也是根据传入的function进行处理，但是不同之处再有这里function传入的参数是一个partition对应数据的iterator
* 补充说明：
  + map按条顺次处理，mapPartitions按分区处理

## 基本概念

* Application：应用程序
* Driver：main()函数，创建SparkContext。由SparkContext负责与ClusterManager通信，进行资源申请，任务分配和监控。程序执行完毕后关闭SparkContext
* Executor：某个Application运行在Worker节点上的一个进程，该进行负责运行某些task，并且负责将数据存在内存或者磁盘上。在Spark on Yarn模式下，其进程名称为CoarseGrainedExecutor Backend，有且仅有一个executor对象，它负责将Task包装成taskRunner，并从线程池中抽取一个空闲线程运行Task，并行运行Task的数据就取决于分配给它的CPU个数
* Worker：集群中可以运行Application代码的节点。Standalone模式中指通过slave文件配置的worker节点，在Spark on Yarn模式中指的是NodeManager节点
* Task：在Executor进程中执行任务的工作单元，多个Task组成一个Stage
* Stage：每个Job会被拆分为很多组Task，作为一个TaskSet，名称为Stage
* DAGSchedule：根据job构建基于Stage的DAG，并提交Stage给TaskScheduler，其划分Stage的依赖是RDD之间的依赖关系
* TaskScheduler：将TaskSet提交给Worker（集群）运行，每个Executor运行什么Task就是在此处分配的

## 运行流程

* 构建Spark Application运行环境（启动SparkContext），SparkContext向资源管理器注册并申请运行Executor资源
* 资源管理器分配Executor资源并启动StandaloneExecutorBackend，Executor运行情况将随着心跳发送到资源管理器上
* SparkContext构建DAG，将DAG分解为Stage，并把Taskset发送给TaskScheduler。Executor向SparkContext申请Task
* TaskScheduler将Task发放给Executor运行同时SparkContext将应用程序程序发放给Executor
* Task在Executor上运行，运行完毕释放所有资源



## 运行特点

* 每个Application获取专属的executor进程，该进程在Application期间一直驻留，并以多线程方式运行tasks。Spark Application不能跨应用共享数据。
* Spark与资源管理器无关，只要能够获取executor进程，并能保持互相通信即可
* 提交SparkContext的Client应该尽量靠近Worker节点，最好在同一个Rack里，因为Spark Application运行过程中SparkContext和Executor之间有大量的信息交换；如果想在远程集群中运行，最好使用RPC将SparkContext提交给集群
* Task采用了数据本地性和推测执行的优化机制

## 监控

* 修改spark/conf/spark-defalut.conf的eventLog.enabled和dir
* 配置spark-env.sh的SPARK\_HISTORY\_OPTS的相关参数
* 启动sbin/start-history-server.sh

## DAG

在Spark中，会根据RDD之间的依赖关系将DAG划分为不同的阶段。

对于窄依赖，由于partition依赖关系的确定性，partition的转换处理就可以在同一个线程中完成，窄依赖就被spark划分到同一个stage中。

对于宽依赖，只能等父RDD shuffle处理完成后，下一个stage才开始接下来的计算。

### DAGScheduler

Job=多个stage，Stage=多个同种的task。Task分为ShuffleMapTask和ResultTask，Dependency分为ShuffleMapDependency和NarrowDependency。

* 接收提交Job的主入口，submitJob(rdd,…)或runJob(rdd,…)
  + 生成一个Stage并提交，判断是否有父Stage未完成，若有，提交并等待父Stage。结果是：DAGScheduler里增加了一些waiting stage和一个running stage。
  + running stage提交后，分析stage里Task的类型，生成一个Task描述，即TaskSet
  + 调用TaskScheduler.submitTask(taskset,…)方法，把Task描述提交给TaskScheduler。TaskScheduler依据资源量和触发分配条件，会为这个TaskSet分配资源并触发执行。
  + DAGScheduler提交job后，异步返回JobWaiter对象，能够返回job运行状态，能够cancel job，执行成功后处理并返回结果
* 处理TaskCompletionEvent
  + 如果task执行成功，对应的stage里减去这个task
    - 如果task是ResultTask，计数器Accumulator加一，在job里为该task置true，job finish总数加一。加完后如果finish数目与partition数目相等，说明这个stage完成了，标记stage完成，从running stages里减去这个stage，做一些stage移除的清理工作
    - 如果task是ShuffleMapTask，计数器Accumulator加一，在stage里加上一个output location，里面是一个MapStatus类。MapStatus是ShuffleMapTask执行完成的返回，包含location信息和block size(可以选择压缩或未压缩)。同时检查该stage完成，向MapOutputTracker注册本stage里的shuffleId和location信息。然后检查stage的output location里是否存在空，若存在空，说明一些task失败了，整个stage重新提交；否则，继续从waiting stages里提交下一个需要做的stage
  + 如果task是重提交，对应的stage里增加这个task
  + 如果task是fetch失败，马上标记对应的stage完成，从running stages里减去。如果不允许retry，abort整个stage；否则，重新提交整个stage。另外，把这个fetch相关的location和map任务信息，从stage里剔除，从MapOutputTracker注销掉。最后，如果这次fetch的blockManagerId对象不为空，做一次ExecutorLost处理，下次shuffle会换在另一个executor上去执行。
  + 其他task状态会由TaskScheduler处理，如Exception, TaskResultLost, commitDenied等。

### TaskScheduler

维护task和executor对应关系、executor和物理资源对应关系、在排队的task和正在跑的task。内部维护一个任务队列，FIFO或Fair策略，调度任务。

### 划分stage

从后往前推，遇到宽依赖就断开，划分为一个stage；遇到窄依赖就将这个RDD加入该stage中。

Task的类型分为：ShuffleMapTask和ResultTask。

DAG的最后一个阶段会为每个结果的partition生成一个ResultTask，即每个stage里面的Task数量是由该Stage中最后一个RDD的Partition数量所决定的。其余所有阶段都会生成ShuffleMapTask；被称为ShuffleMapTask是因为它需要将自己的计算结果通过shuffle到下一个stage中。

* submitTasks(taskSet)，接收DAGScheduler提交来的tasks
  + 为tasks创建一个TaskSetManager，添加到任务队列里。TaskSetManager跟踪每个task的执行状况，维护了task的许多具体信息。
  + 触发一次资源的索要。
  + 首先，TaskScheduler对照手头的可用资源和Task队列，进行executor分配(考虑优先级、本地化等策略)，符合条件的executor会被分配给TaskSetManager。
  + 然后，得到的Task描述交给SchedulerBackend，调用launchTask(tasks)，触发executor上task的执行。task描述被序列化后发给executor，executor提取task信息，调用task的run()方法执行计算。
* cancelTasks(stageId)，取消一个stage的tasks
  + 调用SchedulerBackend的killTask(taskId, executorId, ...)方法。taskId和executorId在TaskScheduler里一直维护着。
* resourceOffer(offers: Seq[Workers])，这是非常重要的一个方法，调用者是SchedulerBacnend，用途是底层资源SchedulerBackend把空余的workers资源交给TaskScheduler，让其根据调度策略为排队的任务分配合理的cpu和内存资源，然后把任务描述列表传回给SchedulerBackend
  + 从worker offers里，搜集executor和host的对应关系、active executors、机架信息等等
  + worker offers资源列表进行随机洗牌，任务队列里的任务列表依据调度策略进行一次排序
  + 遍历每个taskSet，按照进程本地化、worker本地化、机器本地化、机架本地化的优先级顺序，为每个taskSet提供可用的cpu核数，看是否满足
    - 默认一个task需要一个cpu，设置参数为"spark.task.cpus=1"
    - 为taskSet分配资源，校验是否满足的逻辑，最终在TaskSetManager的resourceOffer(execId, host, maxLocality)方法里
    - 满足的话，会生成最终的任务描述，并且调用DAGScheduler的taskStarted(task, info)方法，通知DAGScheduler，这时候每次会触发DAGScheduler做一次submitMissingStage的尝试，即stage的tasks都分配到了资源的话，马上会被提交执行
* statusUpdate(taskId, taskState, data),另一个非常重要的方法，调用者是SchedulerBacnend，用途是SchedulerBacnend会将task执行的状态汇报给TaskScheduler做一些决定
  + 若TaskLost，找到该task对应的executor，从active executor里移除，避免这个executor被分配到其他task继续失败下去。
  + task finish包括四种状态：finished, killed, failed, lost。只有finished是成功执行完成了。其他三种是失败。
  + task成功执行完，调用TaskResultGetter.enqueueSuccessfulTask(taskSet, tid, data)，否则调用TaskResultGetter.enqueueFailedTask(taskSet, tid, state, data)。TaskResultGetter内部维护了一个线程池，负责异步fetch task执行结果并反序列化。默认开四个线程做这件事，可配参数"spark.resultGetter.threads"=4
* TaskResultGetter取task result的逻辑
  + 对于success task，如果taskResult里的数据是直接结果数据，直接把data反序列出来得到结果；如果不是，会调用blockManager.getRemoteBytes(blockId)从远程获取。如果远程取回的数据是空的，那么会调用TaskScheduler.handleFailedTask，告诉它这个任务是完成了的但是数据是丢失的。否则，取到数据之后会通知BlockManagerMaster移除这个block信息，调用TaskScheduler.handleSuccessfulTask，告诉它这个任务是执行成功的，并且把result data传回去。
  + 对于failed task，从data里解析出fail的理由，调用TaskScheduler.handleFailedTask，告诉它这个任务失败了，理由是什么。

### SchedulerBackend

### 宽依赖(Wide Dependencies,Shuffle Dependencies)

重新Shuffle，1:n或者m:n。

* groupByKey
* join(父RDD不是hash-partitioned)
* partitionBy

### 窄依赖（Narrow Dependencies）

* map
* filter
* union
* join(父RDD是hash-partitioned)
* mapPartitions
* mapValues

## 广播变量

如果分布式计算里面分发的大对象不是广播变量，那么每个task就会分发一份，这在task数目十分多的情况下Driver的带宽会成为系统的瓶颈，而且会大量消耗task服务商的资源。如果申明为广播变量，那么每个executor拥有一份，这个executor启动的task会共享这个变量。

### 注意事项

* 广播变量只能读，不能修改
* RDD不能广播，因为RDD不存储数据，可以将RDD的结构广播出去
* 广播变量只能在Driver端定义，不能再Executor端定义
* 在Driver端可以修改广播变量的值，在Executor端无法修改
* Executor端用到了Drvier的变量，如果不使用广播变量在Executor中有多少个task就有多少Driver端的变量副本
* Executor端用到了Drvier的变量，如果使用广播变量在Executor中有一个Driver端的变量副本

### 累加器

累加器在Driver端定义赋初始值，累加器只能在Driver端读取最后的值，在executor

端更新。

## RDD容错

### lineage

每个RDD都包含了其是如何由其他RDD变换过来的以及如何重建某一块数据的信息。

Lineage容错原理：在容错机制中，如果一个节点宕机，针对于窄依赖，只要把丢失的父RDD分区重算，不存在冗余计算；宽依赖会有冗余计算开销。

### checkpoint

针对以下两种情况，RDD加上检查点：

* DAG的Lineage过长，如果重算，开销太大
* 在宽依赖上做checkpoint获得的收益更大

RDD的doCheckPoint方法相当于通过冗余数据来缓存数据。

实际CheckPoint使用方法：

sc.setCheckPointDir(“/tmp/spark/checkpoint”)

data.checkpoint

checkpoint写流程：Initialized->marked for checkpointiong->checkpointing in progress->checkpointed

## 数据倾斜

### 原因分析

* 程序上distinct、group by、join会触发shuffle，若某个key对应的数据量特别大就会发生数据倾斜
* 数据上，若表中若出现一张表的key存在大量的null值或相同一个数据，则会数据分布的时候倾斜
* 业务上，业务发展不平衡

### 现象

* 大多数Task执行会很快，当发生数据倾斜后的task会执行很长时间
* 有时候直接报OOM：JVM Out of Memory

### 定位

数据倾斜多发生在shuffle阶段，需要留意会触发shuffle的算子：distinct、groupByKe、reduceByKey、aggregateByKey、join、cogroup、repartition

* 某个task执行特别慢
  + 确定哪个stage导致，通过log或者Spark Web UI查看stage中task分配的数据量
  + 确定缓慢的stage，根据stage划分原理推断代码
* 某个task内存溢出
  + 从log中找到异常栈，异常栈一般会标明程序出错的位置
  + 在内存溢出代码位置附近找会触发shuffle的操作

## 调优

### 开发调优

* 避免创建重复的RDD
* 尽可能复用同一个RDD
* 对多次使用的RDD进行持久化：调用cache()或persist()，建议选择顺序为MEMORY\_ONLY>MEMORY\_ONLY\_SER>MEMORY\_AND\_DISK\_SER
* 尽量避免使用shuffle算子
* 使用map-site预聚合的shuffle操作：尽量使用map-side预聚合算子（如reduceByKey代替aggregateByKey），map-side预聚合使
* 使用高性能的算子
  + 使用aggregateByKey/reduceByKey代替groupByKey
  + 使用mapPartitions代替map
  + 使用foreachPartitions代替foreach
  + 使用filter之后进行coalesce
  + 使用repartitionAndSortWithinPartitions代替repartition和sort
* 广播大变量：可保证每个executor内存中只保留一份变量，否则每个task都有
* 使用Kryo优化序列化性能

### 建议

* 想要获取更好的表达式查询速度，可以将spark.sql.codegen设置为Ture；
* 对于大数据集的计算结果，不要使用collect() ,collect()就结果返回给driver，很容易撑爆driver的内存；一般直接输出到分布式文件系统中；
* 对于Worker倾斜，设置spark.speculation=true 将持续不给力的节点去掉；
* 对于数据倾斜，采用加入部分中间步骤，如聚合后cache，具体情况具体分析；
* 适当的使用序化方案以及压缩方案；
* 善于利用集群监控系统，将集群的运行状况维持在一个合理的、平稳的状态；
* 善于解决重点矛盾，多观察Stage中的Task，查看最耗时的Task，查找原因并改善

# OpenACC

# 码制

# Python

## Python基础

### 数据类型

* 整型
* 浮点型
* 字符串
  + 字符串连接和复制
    - ‘Alice’ + ‘Bob’
    - ‘Alice’+ 42
    - ‘Alice’\* 5
    - ‘Alice’\*’Bob’

### 数学操作符

* \*\*：指数
* %：取模
* //：整除
* /：除法
* \*：乘法
* -：减法
* +：加法

### 比较操作符

* ==
* !=
* <
* >
* <=
* >=

### 布尔操作符

* and
* or
* not

### 控制流

* if statement:  
   fun1  
  else:  
   fun2
* else:
* elseif statement:
* while statement:  
   fun3
* break
* continue
* for num in range(100)
  + range(0, 10, 2)：开始，停止，步长
* i = 0  
  while i<5:

### 异常处理

* try except
  + def spam(divideBy):  
     try:  
     return 42 / divideBy  
     expect ZeroDivisionError:  
     print(‘Error: Invalid argument.’)

### 列表

* spam = [‘cat’, ‘bat’, ‘rat’, ‘elephant’]
  + spam[-1]
  + spam[-3]
  + spam[2]
  + spam[1:4]
  + len(spam)
  + spam[-1] =spam[0]
  + [1, 2, 3] + [‘A’, ‘B’]
  + [‘A’, ‘B’] \* 3
  + del从列表中删除值,del spam[2]
* in
* not in
* 多重赋值：x,y,z.c =spam，变量数目必须和列表数目一致，否则报错
* index：spam.index(‘cat’)
* append()，插入到末尾
* insert(1, ‘world’)
* spam.remove(‘cat’)，删除不存在会有ValueError错误，出现多次，只会删除第一次的值
* sort()，不能对既有数字又有字符串的列表排序

### 函数

* print()，sep分隔符，end结尾
  + print(‘Hello’, end=‘’)：print()函数自动在传入的字符串末尾加上换行符，end可换
  + print(‘cat’, ‘dog’, ‘mice’,sep=’,’：多个字符串值，print会自动用空格分隔，使用sep参数替换分隔符
* input()：等待用户在键盘上输入一些文本，并按下回车键
* len(str)：字符串str的字符个数
* str()，转变为字符串
* int()，转变为整数，或者取整
* float()，浮点数
* sys.exit()：程序终止或者退出
* import
  + import random
  + from random import \*
* def hello()，定义函数
  + def hello(name):  
     print(‘Hello ’+ name)  
     return None
* global申明该变量为全局变量
* list(),转变为list，可变
* tuple()，转变为元组，不可变
* copy模块的copy()，复制列表或字典这样的可变值
* copy.deepcopy()，若复制的列表中包含列表，需要使用该函数
* random.shuffle()

### 元组数据类型

* eggs = (‘hello’, 42, 05)
* 不可变

### 字典和结构化数据

* {key:value}
  + keys()
  + values()
  + items()
  + in:检查字典中是否存在键或值
* 字典不排序
* get()，检查键是否存在，若不存在则返回默认值
* setdefault()，如果键的值不存在，则设置
  + spam.setdefault(‘color’, ‘white’)
* pprint.pprint(spam)，打印字典
* print(pprint.pformat(spam))

### 字符串

不可变

* 按下标取值
* 切片
* for
* len()
* in
* not in
* “”：之间可以使用单引号
* 转义字符
  + \’
  + \”
  + \t
  + \n
  + \\
* 原始字符串：r’That is cao\’ cat’，忽略所有的转义字符
* ‘‘‘ ’’’，多行字符串，之间的所有引号、制表符或换行，都被认为字符串的一部分
* 多行注释: “““ ”””
* upper()
* lower()
* isupper()
* islower()
* isalpha()
* isalnum()
* isdecimal()
* isspace()
* istitle():仅包含以大写字母开头、后面都是小写字母的单词
* startswith()
* endswith()
* join(),’abc’.join([‘My’,’cat’])->’Myabccat’
* split(),spam.split(‘\n’)
* rjust(),对齐
* ljust()
  + ‘hello’.rjust(10)
  + ‘hello’.rjunst(10,’\*’)
* center()
* strip()
* rstrip()
* lstrip()
* pyperclip模块copy()和paster()

### 正则表达式

* re
* regex = re.compile()
* mo=regex.search()
* mo.group()\mo.group(1)\mo.groups():使用括号分组
* |:匹配表达式的一个
  + heroRegex = re.compile(r’Batman|Tina Fey’
  + mo1 = heroRegex.search(‘Batman and)
* ？：0或1次，可选匹配
* \*：n次
* +：1次或more
* r’(ha){3,5}?’:？为表示非贪心，默认为贪心
* regex.findall():返回所有匹配，若分组，则返回元祖的列表
* \d,\D,\w,\W,\s(空格制表符或换行符),\S
* r’^Hello’：开头
* r’\d$’:以数字结束
* .：通配符，匹配除换行之外的所有字符.\.
* .\*：任意文本
* 通配符匹配换行：regex = re.compile(‘.\*’, re.DOTALL)
* re.IGNORECASE、re.I:不区分大小写
* sub替换：regex.sub(r’\1\*\*\*\*\*’, String1)，\1表示分组1替换
* re.VERBOSE：告诉re.compile忽略正则表达式中的空白符和注释
* 如果需要同时使用re.IGNORECASE、re.DOTAILL、re.VERBOSE需要使用|

### 读写文件

* os，os.path.join():生成文件路径
* os.getcwd():获取当前工作目录
* os.chdir()：切换目录
* os.makedirs()：创建新文件夹
* os.path.abspath(path):
* os.path.isabs(path)
* os.path.relpath(path,start):相对路径
* os.path.dirname(path):目录名称
* os.path.basename(path):基本名称
* os.path.getsize(path)：返回path参数中文件的字节数
* os.listdir(path):返回文件名字符串的列表
* os.path.exists(path)
* os.path.isfile(path)
* os.path.isdir(path)
* open(path),返回一个file对象
* file对象的read()
* readlines()：获取字符串的列表
* write()：写模式，添加模式
  + baconFile = open(‘bacon.txt’, ‘w’),写模式w会覆写原有文件
  + baconFile =open(‘bacom.txt’,’a’),添加模式

### shelve模块

* import shelve
* shelfFile = shelve.open(‘mydata’)
* shelfFile[‘cat’] = cats

运行后会生成文件mydata.bak\mydata.dat\mydata.dir

### 用pprint.pformat()函数保存变量

* pprint.pformat()提供了一个字符串，可以写入文本文件，如.py，此时可以使用import
  + fileObj.write(pprint.pformat(cat)+’\n’)

### 组织文件

* shutil模块
  + copy()
    - shutil.copy(‘c:\\spam.txt’, ‘c:\\delicious’)
  + move()
    - shutil.move(source, destination)
  + rmtree(path)：将删除path处的文件夹，包含所有文件和文件夹都别删除
  + os.unlink(path)：删除path处的文件
  + os.rmdir(path)：删除path处的文件夹，该文件夹必须为空
* send2trash模块
  + send2trash.send2trash，函数只是将文件送到垃圾箱
* 遍历目录树
  + os.walk(path)：在循环的每次迭代中，返回三个值：
    - for filderName, subfolders, filenames in os.walk(‘c:\\delicious’):
    - 1）当前文件夹的名称的字符串
    - 2）当前文件夹中子字符串的列表
    - 3）当前文件夹中文件的字符串的列表
* zipfile模块
  + 读取zip文件
    - exampleZip = zipfile.ZipFile(‘example.zip’)
    - exampleZip.namelist()
    - spamInfo = exampleZip.getinfo(‘spam.txt’)
    - spamInfo.file\_size
    - spamInfo.compress\_size
  + 从zip文件中解压缩
    - extractall(),参数用来指定解压缩的文件夹
  + 创建和添加到ZIP文件
    - 必须以写模式打开，newZip = zipfile.ZipFile(‘new.zip’, ‘w’)
    - newZip.write(‘spam.txt’, compress\_type = zipfile.ZIP\_DEFLATED)，指定压缩算法

### 异常

* raise异常，raise Exception(‘This is the error message.’)
* 断言：禁用断言-O

### 日志

* import logging  
  logging.basicConfig(level=logging.DEBUG,format =’ %(asctime)s - %(levevlname)s - %(message)s’)  
  logging.debug(‘Start of program’)
* 保存到文件：logging.basicConfig(filename = ‘myProgramLog.txt’, level=logging.DEBUG,format =’ %(asctime)s - %(levevlname)s - %(message)s’)
* logging.disable(logging.CRITICAL),可以禁止日志
* 日志级别：
  + DEBUG
  + INFO
  + WARNING
  + ERROR
  + CRITICAL

## 从Web抓取信息

### webbrowser

* webbrowser.open(url)

### requests

* requests.get():下载一个网页

path = r"C:\Users\Liudingchao\Src\Download"  
os.chdir(os.path.join(path))  
dir = os.getcwd();  
print("my name:" + dir)  
res = requests.get( 'https://gitee.com/it-ebooks/tutorialspoint-ebooks-zh/raw/master/TutorialsPoint%20AWT%20%E6%95%99%E7%A8%8B.epub')  
res.raise\_for\_status()  
playFile = open('test.jpg', 'wb')  
for chunk in res.iter\_content(100000):  
 playFile.write(chunk)  
playFile.close()

### Beautiful Soup

* 从HTML创建一个Beautiful Soup对象

res = requests.get('http://it-ebooks.flygon.net/tutorialspoint')  
res.raise\_for\_status()  
searchSoup = bs4.BeautifulSoup(res.text)

* 从硬盘HTML文件创建

exampleFile = open('example.html')  
searchSoup = bs4.BeautifulSoup(exampleFile)

* select()方法寻找元素
  + soup.select(‘div’):所有名为<div>的元素
  + soup.select(‘#author’):带有id属性为author的元素
  + soup.select(‘.notice’):所有使用CSS class属性名为notice的元素
  + soup.select(‘div span’):所有在<div>元素之内的<span>元素
  + soup.select(‘div > span’):所有直接在<div>元素之内的<span>元素，中间没有其他元素
  + soup.select(‘input[name]’):所有名为<input>，并有一个name属性，其值无所谓的元素
  + soup.select(‘input[type=”button”]’):所有名为<input>，并有一个type属性，其值为button元素

### selenium

* 导入selenum需要使用from selenium import webdriver
* 需要安装chrome驱动，<http://chromedriver.storage.googleapis.com/index.html>
* 案例：

browser = webdriver.Chrome()  
browser.get(dstUrl)

* find\_element\_\*:返回一个WebElement;
* find\_elements\_\*:返回WebElement\_\*对象的列表
  + browser.find\_element\_by\_class\_name(name)：匹配CSS类的name元素
  + browser.find\_element\_by\_class\_css\_selector(selector)：匹配CSS的selector元素
  + browser.find\_element\_by \_id(id)；
  + browser.find\_element\_by\_link\_text(text):完全匹配提供的text的<a>元素
  + browser.find\_element\_by\_partial\_link\_text (text):部分匹配
  + browser.find\_element\_by\_\_name(name)：匹配name属性值得元素
  + browser.find\_element\_by\_tag\_name(name)：匹配标签name的元素（只有该方法大小写无关，<a>元素匹配‘a’和‘A’）
* WebElement的属性和方法
  + tag\_name:标签名，如<a>
  + get\_attribute(name):该元素name属性的值
  + text:该元素内的文本
  + clear():对于文本段或文本区域元素，清除其中输入的文本
  + is\_displayed():如果该元素可见，返回True
  + is\_enabled():对于输入元素，如果该元素启用，返回True
  + is\_selected():对于复选框或单选框元素，如果该元素被选中，返回True
  + location:一个字典，包含键’x’和’y’，表示该元素在页面上的位置
* 点击页面
  + WebElement对象都有一个click()方法
* 填写并提交表单
  + 向Web页面的文本字段发送，需要找到文本字段的<input>或<text>元素，然后调用send\_keys()方法
  + emailElem = browser.find\_element\_by\_id(‘Email’)  
    emailElem.sendkeys(‘email@gamail.com’)  
    emailElem.submit()
  + 在任何元素上调用submit()，都等同于点击该元素所在表单的submit按钮
* 发送特殊键
  + from selenium.webdriver.common.keys import Keys
  + Keys.DOWN, UP, LEFT, RIGHT
  + Keys.ENTER, RETURN;
  + Keys.HOME, END, PAGE\_DOWN, PAGE\_UP
  + Keys.ESCAPE, BACK\_SPACE, DELETE
  + Keys.F1, F2, …, F12
  + Keys.TAB
    - htmlElem = browser.find\_element\_by\_tag\_name(‘html’)  
      htmlElem.send\_keys(Keys.END)  
      htmlElem.send\_keys(Keys.HOME)
* 点击浏览器按钮
  + browser.back():后退
  + browser.forward()：前进
  + browser.refresh():刷新
  + browser.quit():关闭窗口

## 处理Excel

### 读取Excel

* import openpyxl
* wb=openpyxl.laod\_workbok(“example.xlsx”)：返回一个workbook对象
* wb.get\_sheet\_name()：取得工作簿中所有表名的列表
* sheet=wb.get\_sheet\_by\_name(‘sheet3’)，depressed，现变更为wb[‘sheet3’]
* sheet=wb.get\_active\_sheet()：获取活动的workbook
* sheet.title：获得名称
* c=sheet[‘A1’]：获取Cell对象
* c=sheet.cell(row=1, column=2)：获取Cell，第一行第一列为1，非零
  + c.value
  + c.row
  + c.column
  + c.coordinate
* sheet.get\_highest\_row()/sheet.get\_highest\_column()：获取表的大小
* 列字母和数字之间的转换：
* from openpyxl.cell import get\_column\_letter, column\_index\_from\_string:转换
* 从表中取得行和列：sheet[‘A1’:’C3’]

for rowsOfCellObjects in activeSheet['A1':'C3']:  
 for cellObj in rowsOfCellObjects:  
 print(cellObj.coordinate, cellObj.value)

### 写入Excel

* wb=openpyx.Workbook():新建
* wb.create\_sheet()：<Worksheet “sheet1”>，返回一个新的Worksheet对象，名为SheetX
* wb.create\_sheet(index=0, title= “First Sheet” )：<Worksheet “First Sheet”>
* wb.remove\_sheet(wb.get\_sheet\_by\_name(“sheet1”))：接收一个Worksheet

### 设置单元格的字体风格

* 设置字体风格

wb = openpyxl.Workbook()  
sheet = wb["Sheet"]  
italic24Font = Font(size=24, italic=True)  
sheet['A1'].font = italic24Font  
sheet['A1'] = "Hello World"  
wb.save("Font.xlsx")

### Font对象

* Font()函数，关键字参数
  + name：字体名称
  + size：大小点数
  + bold：True表示粗体
  + italic：True表示斜体

### 公式

* sheet[‘B9’] = ‘=sum(b1:b8)’：直接赋值
* 如果调用load\_workbook()带有data\_only=True，则显示公式的结果，而不是文本

### 调整行和列

* 设置行高和列宽
* sheet.row\_dimensions[1].height = 70；
* sheet.column\_dimensions[‘B’].width = 20；

### 合并单元格

* sheet.merge\_cells(‘A1:D3’)，若要设置这个值，设置这个组合单元格左上角的单元格的值
* sheet.unmerge\_cells(‘A1:D3’)，拆分单元格

### 冻结窗格

* sheet.freeze\_pans =’C2’，冻结单元格上边的所有行和左边的所有列，本身不会冻结

### 图表

* append可用于一次添加多行数据

wb = Workbook(write\_only=True)  
ws = wb.create\_sheet()  
rows = [  
 ('Number', 'Batch 1', 'Batch 2'),  
 (2, 10, 30),  
 (3, 40, 60),  
 (4, 50, 70),  
 (5, 20, 10),  
 (6, 10, 40),  
 (7, 50, 30),  
]  
  
for row in rows:  
 ws.append(row)

* https://openpyxl.readthedocs.io/en/stable/index.html

## 保持时间、计划任务和启动程序

### time模块

* time.time()：从Unix纪元(1970年1月1日0点)那一刻的秒数
* time.sleep()：暂停

### 四舍五入

* round(time.time(), 2：时间处理

### datetime模块

* KeyboardInterrupt：当用户按下Ctrl+C则抛出该异常
* datetime.datetime.now()：当前时间
* dt = datetime.datetime(2015,10,21,16,59,0)：生成特定时刻datetime对象
* dt.year\month\day\hour\minute\second：属性
* datetime.datetime.fromtimestamp(time.time())与now()一样
* datetime可用比较操作符
* timedelta数据类型
  + delta = datetime.timedalta(days=11, hous=10)，关键字参数有weeks\days\hours\minutes\seconds\milliseconds\microseconds
  + 这些数字分别保存在days\seconds\microseconds属性中
  + delta.total\_seconds()获取总秒数
  + datetime对象转换为字符串，strftime()
    - oct21st=datetime.datetime(2015,10,21,16,29,0)  
      oct21st.strftime(‘%Y/%m/%d%H%M:%S’)
    - %Y：2014
    - %y：14
    - %m：09
    - %B：Novermber
    - %b：Nov
    - %d：31
    - %j：366
    - %w：0~6（周日~周六）
    - %A：Monday
    - %a：Mon
    - %H：00~23
    - %I：01~12
    - %M：分，00~59
    - %S：秒，00~59
    - %p：AM、PM
    - %%：%字符
  + 字符串转换成datetime，strptime()，p表示parse
    - datetime.datetime.strptime(‘October 21, 2015’, ‘%B %d,%Y’);

## 多线程

* threadObj = threading.Thread(target=takeAnap);takeANap()不带括号，表示这是将函数本身作为参数
* threadObj.start()
* 传递给新线程函数，使用args=[‘Cats’,’Dogs’]传递常规参数和kwargs={‘sep’: ‘ &’ }用于关键字参数。

## 从Python启动其他程序

* subprocess模块的Popen()函数
* subprocess.Popen(‘C:\\Windows\\System32\\calc.exe’)；
* 返回Popen对象
  + poll()，若该进程仍在运行，则返回None，若已经终止，无错终止返回0，其他非0
  + wait()，将阻塞
  + 传递命令行参数，sys.argv
  + subprocess.Popen([‘c:\\Windows\\notepad.exe’, ‘c:\\hello.txt’])；
* 使用默认的应用程序打开文件：
  + subprocess.Popen([‘start’, ‘hello.txt’],shell=True)；

## 发送电子邮件和短信

### SMTP

* 连接SMTP服务器
  + smtpObj = smtplib.SMTP(‘smtp.gmail.com’, 587)：TLS（Transport Layer Security，传输层安全协议）
  + smtpObj = smtplib.SMTP(‘smtp.163.com’, 465)：SSL（Secure Socket Layer，安全套接字层）
* 发送SMTP的“Hello”消息
  + smtpObj.ehlo()
* 开始TLS加密，SSL无需此步
  + smtpObj.starttls()
* 登录到SMTP服务器
  + stmpObj.login(“email”,“password”)：登录
  + 如果登录126，虽然密码正确，smtplib.SMTPAuthenticationError: (535, b'Error: authentication failed')，需要开启授权码
  + 
* 发送电子邮件

smtpObj.sendmail("ldc10@126.com", "908009280@qq.com", "Subject: Test for Python!\nDear Alice”)

* + raise SMTPDataError(code, resp)

sendMail = 'ldc10@126.com'  
receiveMail = '908009280@qq.com'  
msg = MIMEText('<html><h1>大佬好！</h1></html>', 'html', 'utf-8')  
msg['Subject'] = Header("test\_email", 'utf-8')  
msg['from'] = sendMail  
msg['to'] = receiveMail  
res = smtpObj.sendmail(sendMail, receiveMail, msg.as\_string())

* 从SMTP服务器断开
  + smtpObj.quit()

### IMAP

* 连接到IMAP服务器，创建IMAPClient对象，大多数需要SSL加密，SSL=TRUE

imapObj = imapclient.IMAPClient("imap.126.com", ssl=True)

* 登录到IMAP服务器

imapObj.login(dstMail, passWord)

* 查看所有的文件夹(IMAP可以支持创建文件夹)

folders = imapObj.list\_folders()

* 选择文件夹
  + select\_folder()，不输入参数默认为INBOX，可加参数readonly=True  
    'NO', [b'SELECT Unsafe Login. Please contact kefu@188.com for help']
  + 需要登录<http://config.mail.126.com/settings/imap/login.jsp?uid=ldc10@126.com>处理

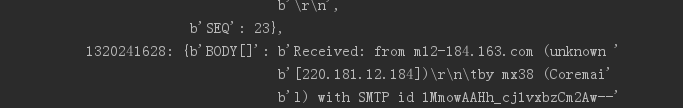
imapObj.select\_folder('INBOX', readonly=True)

* 搜索，serach()，返回邮件UID
  + ALL
  + BEFORE data, ON, SINCE
  + SUBJECT string, BODY, TEXT, FROM, TO, CC(抄送), BCC(密件抄送)
  + SEEN(已看标记), UNSEEN
  + ANSWERED, UNANSWERED
  + DELETED, UNDELETED
  + DRAFT, UNDRAFT(草稿)
  + FLAGGED, UNFLAGGED(重要或紧急)
  + LARGGER N, SMALLLER
  + NOT search-key
  + OR search-key1 search-key2

UIDS=imapObj.search(['ALL'])

* 获取具体内容

rawMessages = imapObj.fetch(UIDS, ['BODY[]'])

* 获取得数据如下，在’BODY[]’中的内容为RFC822，是专为IMAP服务器读取为设计的
* 
* 如果想要把获取时标记为已读，需要使用readonly=False
* 大小限制
  + imaplib.\_MAXLINE = 1000000
* 从原始消息中获取电子邮件地址
  + pip install pyzmail，此处有bug，换下列命令
  + 
  + easy\_install pyzmail
  + 创建PyzMessage对象

message = pyzmail.PyzMessage.factory(rawMessages[1320241627][b'BODY[]'])

* + 获取主题，地址等

subject = message.get\_address(“from”)；to、cc、bcc

* 从原始消息中获取正文
  + 文本
    - if message.text\_part:  
       messageTxt = message.text\_part.get\_payload().decode(message.text\_part.charset)
  + Html
    - if message.html\_part:  
       messageHtml = message.html\_part.get\_payload().decode(message.html\_part.charset)
* 删除电子邮件
  + imapObj.delete\_message(UIDs)
  + imapObj.expunge()
* 从IMAP服务器断开
  + imapObj.logout()

## BeautifulSoup

### Definition

Beautiful Soup是一个可以从HTML或XML文件中提取数据的Python库。

### 安装

<https://www.crummy.com/software/BeautifulSoup/bs4/download/>

下载后解压，到目标文件夹，

python setup.py build

python setup.py install

输入from bs4 import BeautifulSoup，若无提示则安装成功

若出现：

'You are trying to run the Python 2 version of Beautiful Soup under Python 3. This will not work.'<>'You need to convert the code, either by installing it (`python setup.py install`) or by running 2to3 (`2to3 -w bs4`).'

则版本兼容问题。

解决方案：

1. 将BeautifulSoup解压目录下的bs4文件夹复制到Python安装目录的Lib下
2. 将Python安装目录下的Tools\scripts的2to3.py拷贝到Lib下
3. Cmd中执行2to3.py bs4 –w

# CMD相关

## 网络相关

### 查看被占用的端口

* Netstat –ano列出所有端口



* Netstart –ano|findstr “1285”



* Tasklist|findstr “1264”



## 文件操作

### 创建文件

|-type nul>.test（生成空文件）

|-echo fileContent>a.txt（生成非空文件）

## 显示更改

|-chcp 936 更改为默认的GBK编码

# 八仙过海

## 设计树形数据结构

# Virtuosity

## Windows

设置Windows内置截图，发送快捷方式，设置快捷键

删除我的电脑里面其他的图标，按照如下路径删除下面东西



### Cmd创建文件

|-type nul>.test（生成空文件）

|-echo fileContent>a.txt（生成非空文件）

# 书单

|-学通Java的24堂课

|-Java开发实战经典（名师讲坛）——李兴华

|-编写高质量代码：改善Java程序的151个建议