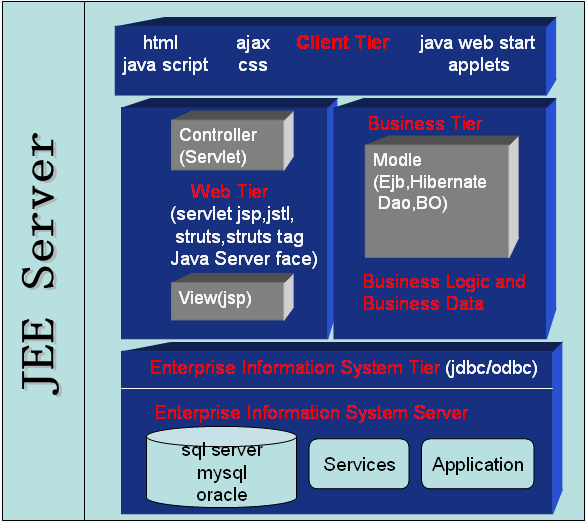
Java se（j2se）桌面开发

Java ee（j2ee）web开发

Java me（j2me）手机开发



Jdk全称Java Development Kit中文Java开发工具包

Jre（Java runtime environment）Java运行环境

Java 基本格式

public class …

{

public static void main(String[] arys)

{

}

}

Java源文件.java文件

Java字节码文件.class文件

Java虚拟机（Java virtual machine）实现多平台运行

Java.exe（解释执行器）将字节码文件加载到Java虚拟机，字节码文件在Java虚拟机中执行

常用整数类型byte（1字节），short（2字节），int（4字节），long（8字节）

常用小数（浮点）类型float，double

布尔类型boolean

字符类型char（2字节）单个字符（可以存放汉字） Sting字符串

数据精度byte<short<int<long<float<double 低精度数据会自动向高精度数据转换

高精度数向低精度数强制转换格式 低精度类型 变量=（低精度类型）高精度数据

从控制台输入数据方法

InputStreamReader 变量=new InputStreamReader（System.in）；

BufferedReader 变量名=new BufferedReader（变量）；

或者变量名=new BufferedReader（new InputStreamReader（System.in））；

变量=变量名.readLine();

从控制台输入的任何数据机器全认为是字符串

字符串转float型数据格式 float 变量=Float.parseFloat（变量名）；

Switch语句的条件表达式数据类型需要和case后的常量类型一致

Switch中的数据类型可是是byte，short，int，char，enum

计算机语言发展汇编语言（面向机器），c语言（面向过程），Java（面向对象）

Java同一个包下只能有一个public类

引用类型之间用=赋值没有使用new语句表示两个变量间地址赋值即两个变量指向同一地址和内容

this关键字只能在类中使用表示指向本类的内容

super关键字也只能在类中使用表示指向本类的基类的内容

satatic关键字表示静态内容



package关键字一般放在文件开始头部

包命名一般是小写字母

常用的包java.lang.\*自动引入 java.util.\*工具包 java.net\*网络开发包 java.awt.\*窗口工具包

extends关键字继承 Java只支持继承一个基类

eclipse是开源的但只支持开发java，c，c++

myeclipse是eclipse基础上安装了myeclipse插件

eclipseme插件可以支持手机软件开发

PHP插件支持PHP

关键字abstract定义抽象内容 抽象内容不能被实现

关键字interface定义接口 类进行实现接口时要实现接口的所有接口内的方法

关键字implements表示类实现接口

接口不能被实例化 接口的所有方法不能有实现部分 抽象类可以有实现的方法

类可以实现多个接口只能继承一个类

接口中的变量只能是公共静态常量的即（public，static，final）

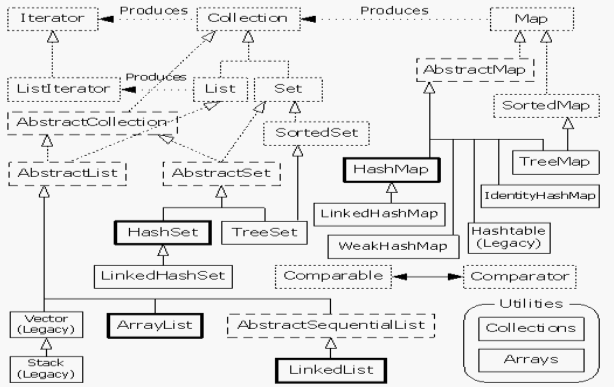
接口不能继承类但可以继承接口

关键词final定义不能被修改以及不能被覆盖的内容

Eclipse帮助快捷键在edit,content assist,default

设置快捷键在window,preference,general,keys

集合类



List结构的集合类ArrayList类，LinkedList类，Vector类，Stack类

Map结构的集合类HashMap类，Hashtable类 Map结构的集合类有键值

Set结构的集合类HashSet类，TreeSet类

Queue结构的集合Queue接口

字符串比较equals（）方法实现

System.exit（）方法终止Java虚拟机

关键词Iterator迭代定义

ArrayList与HashMap是异步，Vector与Hashtable是同步的

<>泛型定义

Java异常

检查性异常：java.lang.Exception

运行期异常：java.lang.RuntimeException

错误：java.lang.Error

异常基类java.lang.Throwable类

关键字FileReader定义读取文件

计算机命令窗口ping 网址表示显示该网址的IP地址

计算机命令窗口tracert 网址表示网络到达该网址所经过的路由器

计算机命令窗口route print表示路由表

计算机命令窗口netstat –an表示连接到本机的主机

关键字Socket定义连接的IP地址

异常处理语句格式

try{

检查的语句

}catch（异常类型（Exception（最大） 变量）{

变量.处理方法

}finally{

有无异常该语句都会执行

}

方法printStackTrace（）显示异常内容并显示异常的行数

方法getMessage（）显示异常内容

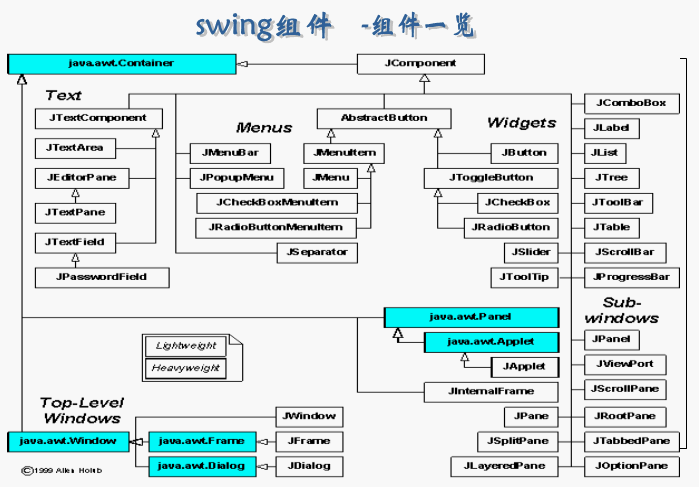
方法名后加throws Exception表示异常抛出

方法substring（）取字符串中的字符串

图形用户界面（Graphics User Interface，GUI）

GUI开发工具包awt（Abstract Window Toolkit），swing，swt，JFace

Java GUI开发工具常用包java.awt.\*与javax.swing.\*



关键词JFrame容器类定义GUI的容器，一般使用类继承JFrame即extends JFrame

JFrame对象.setDefaultCloseOperation（JFrame.ExIT\_ON\_CLOSE）；关闭Java虚拟机

关键词JButton按钮，JRadioButton单选按钮，BasicArrowButton箭头按钮，JToggleButton触发器按钮

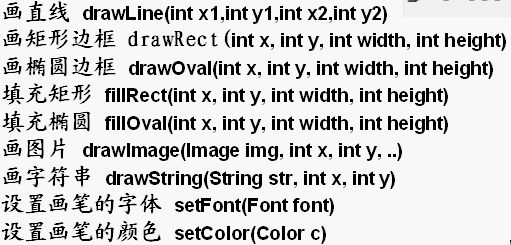
Java提供5中布局管理器FlowLayout流式布局管理器，BorderLayout边界布局管理器，GridLayout网格布局管理器，CardLayout卡片布局管理器，GridBagLayout网格布局管理器

Component类提供两个绘图方法：

paint(Graphics g)绘制组件的外观

repaint()刷新组件的外观

Graphics绘图类包含以下绘图方法



多行注释方法：选中多行按Ctrl+/，取消多行注释也是同样按键

java.awt.event包和javax.swing.event包中定义了各种事件类型，常见的事件类有：

ActionEvent 通常在按下按钮，或双击一个列表项或选中某个菜单时发生

AdjustmentEvnet 当操作一个滚动条时发生

ComponentEvent 当一个组件隐藏、移动、改变大小时发送

ContainerEvent 当一个组件从容器中加入或者删除时发生

FocusEvent 当一个组件获得或是失去焦点时发生

ItemEvent 当一个复选框或是列表项被选中时，当一个选择框或选择菜单被选中

KeyEvent 当从键盘的按键被按下，松开时发生

MouseEvent 当鼠标被拖动、移动、点击、按下..

TextEvent 当文本区和文本域的文本发生改变时发生

WindowEvent 当一个窗口激活、关闭、失效、恢复、最小化...

在java中一个类要当作线程来使用有两种方法:

继承Thread类，并重写run函数

实现Runnable接口，并重写run函数

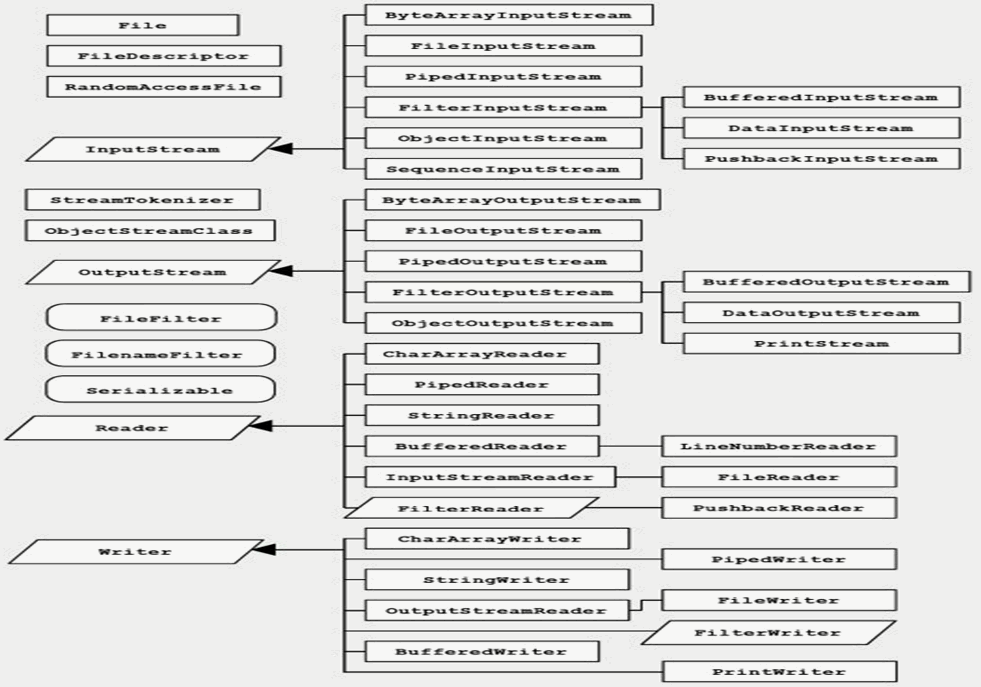
Thread.currentThread().getName()显示当前进程的名字（编号）

Java线程同步synchronized(object){需要的同步代码}

文件字节流输入流FileInputStream输出流FileOutputStream

文件字符流输入流FileReader输出流FileWriter

文件缓冲字符流输入流BufferedReader输出流BufferedWriter



数据库：

微软：sql server 和 acess

瑞典MySQL：AB公司mysql

Ibm公司：db2 海量数据库

美国Sybase公司：Sybase

Ibm公司：informix 安全性好，银行常用

美国oracle：oracle

IT公司排行：1标准 2服务 3产品 4项目

SQL是英文Structured Query Language的缩写（结构化查询语言）

binary 数据库存取视频

数据库备份：backup database 需要备份的数据库名 to disk=’路径’

数据库恢复：restore database 需要恢复的数据库名 from disk=’路径’

Java通过Jdbc-odbc连接数据库方法：

一．配置数据源：开始>控制面板>管理工具>数据源（ODBC）>添加，服务器中（.）本机

二．程序连接数据源：一般的程序：

1.//加载驱动

Class.*forName*("sun.jdbc.odbc.JdbcOdbcDriver");

2.//得到连接（数据库用户名和数据库登录密码根据数据库验证方式选择是否传入）

Connection connection=DriverManager.*getConnection*("jdbc:odbc:新添加数据源的名称","数据库用户名","数据库登录密码");

3.//创建Statement或者PreparedStatement（进行传送数据库操作指令到数据库）

Statement statement=connection.createStatement();

4.//执行（crud，数据库操作）

statement.executeUpdate("数据库操作指令");

数据库连接不关闭会导致数据库连接数达到最大无法连接问题的查询方式：dos窗口下输入命令netstat an

Statement与PreparedStatement区别：两者都可以讲SQL语句发送给数据库，PreparedStatement可以将SQL语句预编译并且缓存SQL编译后结果不用再次编译，还可以预防危险字符的输入

Java通过Jdbc连接数据库方法：

程序连接数据源：一般的程序：

1.//加载驱动

Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");

2.//得到连接（数据库用户名和数据库登录密码根据数据库验证方式选择是否传入）localhost是本机IP地址，也可以是其他主机IP地址，1433是sql server的默认端口号

Connection connection=DriverManager.getConnection("jdbc:microsoft:sqlserver://localhost:1433;databaseName=studentsdb","数据库用户名","数据库登录密码");

3.//创建Statement或者PreparedStatement（进行传送数据库操作指令到数据库）

PreparedStatement preparedStatement=connection.prepareStatement("数据库操作指令");

4.//执行（crud，数据库操作）

preparedStatement.executeQuery();

executeUpdate()一般用于数据库的内容的变化，executeQuery()一般用于数据库的查询

计算机命令窗口输入tracert IP地址表示查找连接到IP地址经过的路由器

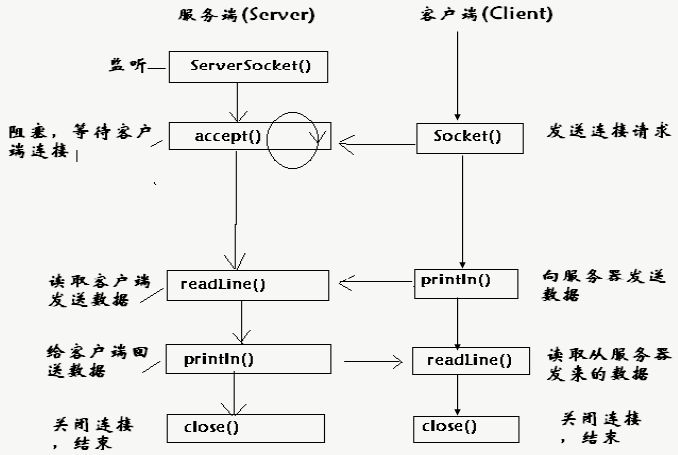
计算机命令窗口输入ping 127.0.0.1或者任意非全0全1的值表示回路，查看能否连接自己，查看网卡是否有问题

计算机命令窗口输入netstat -an表示查找计算机开放端口

计算机命令窗口输入netstat -anb表示查找计算机开放端口的进程id号pid

URL统一资源定位符（Uniform Resource Locator）

服务器实例化ServerSocket和Socket对象，将ServerSocket的对象调用accept函数赋值给Socket的对象



Tomcat文件conf：配置文件，server.xml：用于配置server相关的信息如tomact启动端口，配置Host，配置Context即web应用。

Web.xml：配置与web应用

Tomcat-users.xml：用户配置tomcat的用户密码和权限

<http://localhost:8080/web/index.html：URL>（统一资源定位）

http:协议，localhost：主机域名，8080：端口，web：web应用，index.html：资源文件

web/hello.html：URI

work：存放jsp被方问后生产的对应的servlet文件class文件

web应用下的web.xml文件中配置默认首页面：

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

配置虚拟目录在tomcat的conf目录下的server.xml的<Host>节点中添加<Context path=”/web应用名” docBase=”文件绝对路径” />

Context元素reloadable设置有文件更新tomcat会自动重新加载，unpackWAR设置运行前解压所有web应用

tomcat体系架构从大到小：tomcat（server），service，engine（引擎），host（主机），context（web应用）

开发servlet的三种方法：实现servlet接口，继承GenericServlet类，继承HttpServlet类

Servlet的init（ServletConfig config）初始化servlet，函数只会调用一次

将Servlet部署到web.xml文件

<servlet>

<servlet-name>servlet的注册名</servlet-name>

<servlet-class>servlet的绝对路径</servlet-class>

</servlet>

<servlet>

<servlet-name>servlet的注册名</servlet-name>

< url-pattern >servlet的访问路径（格式为：/名字）</url-pattern>

</servlet>

Servlet的访问路径使用通配符的格式为：/\*表示匹配所有，\*.扩展名

匹配标准：匹配度高优先选择，\*.扩展名优先级最低

使用javac编译有自定义包名的java文件，需要带命令参数格式为：javac –d .java文件

提交方式get与post的区别：get会将提交的信息显示到地址栏，get提交内容一般不大于2k，post理论无限制，get提交会立即处理响应。

Servlet配置<load-on-startup>启动优先级（数字）</load-on-startup>指定servlet的创建顺序

<init-param>

<param-name>参数名</param-name>

<param-value>参数值</param-value>

<init-param>通过getServletConfig()函数读取

setContentType("text/html;charset=utf-8");设置浏览器默认解析的编码格式

setCharacterEncoding("utf-8");设置程序编码格式

Servlet请求重定向函数sendRedirect(“跳转地址?参数1名=参数1值&参数2名=参数2值”)

超文本传输协议（HTTP，HyperText Transfer Protocol）

http1.0称为短连接，http1.1称为长连接，是指请求时间间隔的长短

http的请求基本结构：

请求行（请求方式，请求的资源名称，http协议版本）

消息头（格式 消息名：内容）

空行

内容（格式：内容名字=内容体）

http请求消息头：

Accept：浏览器接受的格式

Accept-Charset：浏览器接受字符编码

Accept-Encoding：浏览器接受数据的格式

Accept-Language：浏览器支持的语言

Host：浏览器访问的主机和端口

If-Wodified-Since：浏览器缓存文件及时间

Referer：浏览器访问主机方式

Use-Agent：浏览器内核信息

Cookie：网页信息

Connection： 连接方式

Date：浏览器发送http请求时间

http的响应基本结构：

状态行（格式 http版本号 状态码 原因）

消息头（格式 消息名：内容）

空行

内容（格式：内容名字=内容体）

http响应消息头：

Location：浏览器重新定位地址

Server：服务器信息

Content-Encoding：数据格式

Content-Length：数据大小

Content-Language：数据语言

Content-Type：内容格式

Last-Modified：资源最近更新的时间

Refresh：多少时间后跳转到指定地址

Content-Disposition：attachment;filename=文件名称

Transfer-Encoding：传输编码，校验和

Set-Cookie：cookie信息

Expires：缓存数据方式（为了支持不同浏览器内核设置3个缓存数据方式）

Cache-Control：缓存数据方式

Pragma：缓存数据方式

Connection： 连接方式

Date：浏览器收到http响应时间

getWriter()用于回送字符数据

getOutputStream()用于回送字节数据（二进制数据）

由于web服务器会检测数据流是否关闭，未关闭服务器会关闭，所以getWriter()与getOutputStream()同时使用会第二个无法正常传输数据。

中文乱码原因：浏览器根据不同的请求提交方式有不同编码格式，tomcat服务器默认编码格式为iso-8859-1

解决方法：

浏览器显示编码格式设置：

response.setContentType("text/html;charset=utf-8");

response.setCharacterEncoding("utf-8");

post请求提交：request.setCharacterEncoding(“utf-8”);

get请求提交：new String(string.getBytes(“iso-8859-1”),”utf-8);

下载文件提示框编码格式设置：java.net.URLEncoder.encode(“中文名”,”utf-8”);

getHeader(“参数名”)返回请求头信息中参数名的值

服务器内部转发request.getRequestDispatcher("/转向的Servlet名").forward(request, response);转发后的request和response与转发前是同一个request和response

浏览器转发response.sendRedirect(“/web应用名/资源名（servlet）”)转发是在，request和response对象与转发之前不是同一个

分页变量

pageNow：现在显示第几页

pageSize：每页显示多少条记录

pageCount：共有多少页

rowCount：共有多少条记录

java web读取文件使用类加载器，类加载器读取资源默认是主目录是src目录

inputStream=类名.class.getClassLoader().getResourceAsStream("文件名");

浏览器的Cookie是在服务器创建会保存到浏览器

Cookie 名字=new Cookie(属性名, 属性值);cookie只能存放String类型数据

名字.setMaxAge(正数);设置cookie存放时间（单位是秒）

名字.setMaxAge(0);在浏览器退出后删除cookie

名字.setMaxAge(负数);负数与默认不设置一样在关闭浏览器时清除

response.addCookie(名字);cookie通过http响应返回给浏览器并保存

cookie可以多个浏览器共享

一个浏览器最多存放300个cookie，一个web站点最多存放20个cookie，每个cookie大小最大为4k

浏览器的cookie读取

Cookie 名字[]=request.getCookies();通过request类读取返回所有cookie

名字.getName()获得cookie的属性名，重名的cookie值会被替换

名字.getValue()获得cookie的属性值

一个web应用可以保存多个cookie，有最大数量限制

一个web应用的所有cookie都存放在同一个文件里

Cookie存放中文时可能会出现乱码或者错误，因为cookie中文编码格式默认为base64

cookie存放中文通过java.net.URLEncoder.encode(需要编码的值, 编码格式);进行编码

cookie取出中文通过java.net.URLDecoder.decode(需要解码的值, 解码格式);进行解码

cookie默认是明文存放不安全重要信息需要加密

密码加密存放，一般是用户输入密码进行加密后再存放到数据库，验证时对用户输入的密码加密后到数据库对比验证

Session是存在服务器端的内存中，服务器为每个用户的浏览器创建各自独享的session对象

服务器未发现session会自动创建一个，一般都是在浏览器第一次访问时创建

Session 名字=request.getSession(参数)获得服务器的session，参数为false表示服务器未找到session不去创建新的session，参数为true与默认不填表示服务器未找到session会创建新的session

session默认存在时间30分钟（单位是分钟），服务器（tomcat）下所有web应用的所有session默认时间修改在服务器（tomcat）的web.xml文件中，一个web应用下的所有session默认时间修改在相应web应用下的web.xml文件中，一个session的默认时间设置是名字.setMaxInactiveInterval(时间);设置session名为名字的存在时间（单位是秒），默认时间的设置优先级从大到小为session，web应用，服务器

名字.setAttribute(属性名（字符串）, 属性值（object）);设置session内容，属性名一样的值将会被替换

名字.getAttribute(参数);返回session中属性名为参数的值，返回类型为object

名字.removeAttribute(参数);删除session中属性名为参数的值，让session中属性名为参数值失效

重启服务器（tomcat），重新加载（reload）web应用，关机等操作会导致session失效

名字.invalidate();设置session立即失效（包含session中所有属性），一般用于安全退出

List集合有序，map集合无序

用户禁用浏览器的cookie后的session处理方法

response. encodeRedirectURL(url地址);用于对sendRedirect方法后的url地址进行重写

response.encodeURL(url地址);用于对表单提交和超链接的url地址进行重写

ServletContext是在服务器端web应用启动时创建的，一个ServletContext被改web应用的所有客户端共享，

ServletContext可以通过ServletContext 名字=this.getServletContext();方法获得也可以通过

ServletContext名字=this.getServletConfig().getServletContext();方法获得

ServletContext的生命周期从创建开始到web应用服务器关闭为止

web应用下的web.xml文件servlet标签中配置

<init-param>

<param-name>参数名</param-name>

<param-value>参数值</param-value>

</init-param>只会被该servlet访问

访问使用String 名字=this.getServletConfig().getInitParameter("参数名");

web应用下的web.xml文件中配置

<context-param>

<param-name>参数名</param-name>

<param-value>参数值</param-value>

</context-param>会被该web应用所有servlet访问

访问使用String 名字=this.getServletContext().getInitParameter("参数名");

ServletContext服务器web应用内部实现跳转this.getServletContext().getRequestDispatcher("跳转地址");

ServletContext读取文件

InputStream 名称=this.getServletContext().getResourceAsStream("资源名称及位置");

属性读取对象Properties

properties.load(inputStream流对象);加载文件流

properties.getProperty("属性名");获得属性值

获取文件的绝对路径String 名字=this.getServletContext().getRealPath("资源文件名称");

ServletContext能读取web目录和WEB-INF目录下的文件

其他文件下需要使用类加载器this.getClass().getClassLoader().getResourceAsStream("资源文件名称");

统计页面访问次数不包含网页刷新的方法

将页面跳转方式服务器内部跳转request.getRequestDispatcher("页面地址").forward(request, response);换成浏览器跳转response.sendRedirect("页面地址");

Jsp指令元素：

<%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>

Page指令的属性：

Language=”jsp嵌入的代码语言”

Import=”jsp引入的包或类名”

Session=true|false 在jsp页面session对象是否允许使用

Buffer=none|8k|指定大小 out对象使用的缓冲区是多大，默认8k

AutoFlash=true|false buffer慢后是否自动刷新到浏览器

isThreadSafe=true|false 默认true表示jsp的线程安全由程序员控制，false对应的servlet将实现线程安全接口

errorPage=”网页地址” jsp页面出错会自动跳转到指定页面

isErrorPage=false|true 指定jsp是否用于error页面，默认false，true表示页面可以直接使用exception内置对象

contentType=”text/html;charset=编码格式” 指定网页以什么形式显示页面

pageEncoding=”编码格式”指定页面以什么编码格式翻译jsp生成相应的servlet

include指令

<%@ include file=”文件名” %> 引入一个文件（一般为jsp文件），jsp引擎将页面jsp与引入的文件翻译成一个servlet文件，也称为静态引入，引入jsp文件只需要保留page指令

Taglib指令 允许在jsp页面使用自定义标签

jsp脚本元素：

<% java代码 %>或者<%=java表达式%>

<%! 声明 %> 定义jsp全局变量或者函数，在对应servlet中是类成员变量或者函数

jsp动作元素：

<jsp:useBean> 创建一个javabean实例

<jsp:setProperty> 给一个javabean实例设置初始值

<jsp:param name=”参数名” value=”参数值”/> 给一个jsp设置参数

<jsp:getProperty> 取得一个javabean实例的类成员变量

<jsp:include file=”文件名”> 引入另外一个文件，动态引入页面jsp与引入的文件翻译成不同的servlet文件

<jsp:forward> 转发

<jsp:fallback>

jsp文件的注释：

<!-- 注释内容 --> html注释方式可以使用在jsp中，会在相应servlet中显示，返回给浏览器的静态页面带有注释内容，只是用户看不到

<%-- 注释内容 --%> jsp专用注释方法，不会在相应servlet中显示，建议使用这种方法，节省网络开销

jsp九大内置对象：属于java语言需要写在<%%>中

out 向客户端输出数据，字节流，对应servlet的JspWriter

request 接受客户端的http请求，对应servlet的HttpServletRequest

response 封装服务器jsp产出的http响应，对应servlet的HttpServletResponse

session 用于保存用户的信息，跟踪用户的行为，对应servlet的HttpSession

application 多个用户共享该对象，对应servlet的ServletContext

pageContext jsp页面的上下文的域对象，作用域为该jsp页面，对应servlet的PageContext

exception 表示运行时的一个异常，对应servlet的Throwable

page 表示jsp本身，对应servlet中this

config 表示jsp对应servlet的配置，可以得到web.xml中的参数，对应servlet中ServletConfig

jsp文件放在WEB-INF文件下浏览器不能通过地址直接访问到，这个服务器的安全机制

jsp取出变量值可以使用${变量名}

SMTP（简单邮件传输协议） POP3协议（邮局协议）

oracle认证：

oca（初级认证）oracle certified associate

ocp（中级认证）oracle certified professional

ocm（高级认证）oracle certified master

sqlplus常用命令：

conn[ect] 用户名/密码@网络服务名[as sysdba/sysoper]

连接数据库，特权用户身份连接需要sysdba或者sysoper

disc[onnect] 断开与当前数据库的连接

passw[ord] 修改密码，修改其他用户密码需要加上其他用户的用户名并且当前用户拥有相应dba权限

show user 显示当前用户名

exit 断开与当前数据库的连接并退出

交互命令：

&变量名 替代变量，变量执行时需要用户输入

edit 文件路径及名称 编辑指定的sql脚本

spool on

数据库语句

spool 文件路径及名称 将数据库语句结果输出到指定文件

spool off

显示设置环境变量：控制输出的格式

oracle除了sql语句外的其他sqlplus语句可以省略最后的“;”号

set linesize 字符数 设置每行显示字符的数量，默认20个字符

set pagesize 行数 设置每页显示行的数量，默认14行

create user 用户名 identified by 密码

[default tablespace 默认表空间名（设置用户默认属于表空间）

temporary tablespace 临时表空间名（设置用户临时属于表空间）

quota 大小 on 表空间名（设置用户在该表空间）]

拥有dba权限的用户创建新用户，密码不能用数字开头，创建的新用户无任何权限

alter user 用户名 identified by 新密码 修改其他用户的密码，当前用户拥有相应dba权限

grant权限/角色 to 用户名 拥有相应权限/角色的用户可以给其他用户授权相应权限/角色

oracle权限分为系统权限和对象权限

角色分为预定义角色和自定义角色，角色含有多种权限是权限的集合

revoke 权限/角色 from 用户名 拥有更高权限的用户回收权限/角色

drop user 用户名 [cascade] 删除用户，cascade表示删除用户把该用户下的所有数据一起删除

oracle新创建的用户在创建第一个数据对象时创建对应用户的方案（schema），用户创建的所有数据对象会放在用户对应的方案中

oracle访问其他方案中的数据对象：

grant 权限[all（所有权限）] on 数据对象 to 用户名 [with admin/grant option]登录要访问的方案对应的用户授权相应数据对象权限给相应用户，with admin/grant option表示用户可以将得到的系统权限/对象权限授权给其他用户

create profile 限制名 limit failed\_login attempts/password\_life\_time 失败登录次数/密码使用天数 password\_lock\_time/password\_grace\_time 密码锁定天数/密码使用宽限天数

拥有dba权限用户创建用户登录失败相应次数后锁定相应天数/用户密码使用天数的限制

alter user 用户名 profile 限制名 拥有dba权限的用户给其他用户设置限制

alter user 用户名 account unlock 拥有dba权限的用户给其他用户解锁限制

drop profile profile文件名，删除profile文件限制

查看计算机信息：命令行窗口输入systeminfo

Windows命令行窗口启动oracle：

lsnrctl start 启动监听

oradim –startup –sid 数据库实例名 启动数据库实例

Linux命令启动oracle：  
lsnrctl start 启动监听

sqlplus sys/change\_on\_install as sysdba 以sysdba身份登录，oracle10g以后版本可以使用（

sqlplus /nolog

conn sys/change\_on\_install as sysdba ）

startup

oracle在windows系统下的登录认证方式：

操作系统认证：当前windows用户属于ora\_dba组成员，可以使用conn 任意用户名/任意密码 as sysdba 验证登录oracle数据库

oracle数据库验证（密码文件验证）：

普通用户默认使用数据库密码验证

特权用户（sys用户）默认使用操作系统认证，验证不通过使用数据库密码验证

配置sqlnet.ora文件可以修改oracle默认登录认证方式

SQLNET.AUTHENTICATION\_SERVICES=(NTS/NONE/（NONE,NTS）)基于操作系统/oracle数据库密码文件/两者共存验证

Linux下oracle登录认证方式

默认Linux下的oracle数据库基于操作系统oracle数据库密码文件共存验证，默认sqlnet.ora文件中没有SQLNET.AUTHENTICATION\_SERVICES=(NTS/NONE/（NONE,NTS）)参数，添加参数NONE与NTS都是基于oracle数据库密码验证

oracle管理员密码恢复：

删除oracle数据库安装目录下PWD数据库实例名.ora文件（建议删除前备份）

在命令行窗口输入orapwd file=PWD数据库实例名.ora文件全路径 password=新密码 entries=特权用户最大数量 运行后重启oracle数据库

oracle数据库数据类型：

char(字符量) 定长，最大2000字符，一个字符长度是一个字节

函数dump(列名) 显示该列的详细信息（数据类型，使用字符量长度）

varchar2(字符量) 变长，最大可以存放4000字符

nchar(字符量) 定长，编码方式为unicode，一个字符长度是两个字节，最大2000字符

nvarchar2(字符量) 变长，编码方式unicode，最大4000字符

clob(character large object) 字符型大对象 变长，最大8tb

blob(binary large object) 二进制文件 变长，最大8tb

number(p,s) 变长，整数小数，p为有效位数（1<=p<=38），s为小数位数（-38<=s<=127）

默认数据保存范围：-1.0\*e^(-130)<=value<=1.0\*e^126（-1.0\*10^(-130)<=value<=1.0\*10^126）

默认保存在机器中范围：1~22byte

date 包含年月日时分秒，默认格式“日-月-年”，合法日期范围公元前4712年1月1日到公元4712年12月31日

TIMESTAMP(n) n取值0~9表示秒数的小数位，n可选，n为0是与date等价，日期随数据更新而自动更新

alter table 表名 add/modify (列名 数据类型); 添加/修改表中列

alter table 表名 drop column 列名; 删除表中列

rename 表名 to 新表名; 修改表名称

drop table 表名; 删除表

desc 表名; 查看表结构

crud（create/retrieve/update/delete）增删改查

oracle数据库插入表数据时会尝试将输入的类型转换成相应字段类型

truncate table 表名; 删除表中所有记录，保留表结构，速度快，不记录日志

savepoint 时间点名 设置保存点

rollback to 时间点名 撤销操作到保存的时间点

select [distinct]，表示查询结果是否显示重复数据

oracle的sql语句不区分大小写，内容参数区分大小写

nvl(列名,0)函数 0替换null返回

select 列名（单引号的字符串） || 列名（单引号的字符串） from 表名; 查询时将多列拼成一列返回

alter user 用户名 account lock; 锁定用户

like操作符属性：%表示匹配任意0到多个字符，\_表示匹配任意单个字符

order by属性默认asc升序排列，desc降序排列

oracle支持使用列的别名进行排序

avg()函数会忽略值为null的记录，去平均值的时候不会计算值为null的记录

sql语句默认是从右向左执行

oracle高效率分页查询：

select 表的别名.字段 from (select 表别名.字段,rownum 行别名 from (select 字段 from 表名 where 条件) 表别名 where 条件（rownum<=结束行）) 表的别名 where 条件（行别名>=开始行）;

oracle合并查询：

union 查询结果取并集

union all 查询结果不去除重复项取全集，不排序

intersect 查询结果取交集

minus 查询结果取差集，取在第一个数据集合中不在第二个数据集合中的数据

cube(字段,字段) 先分别对两个字段操作，再对两个字段的组合操作，返回所有结果

连接查询：

select 字段 from 表名 inner join 表的名 on 表名.字段=表的名.字段; 内连接查询

select 字段 from 表名 left join 表的名 on 表名.字段=表的名.字段;或者select 字段 from 表名,表的名 where 表名.字段=表的名.字段(+); 左外连接查询

select 字段 from 表名 right join 表的名 on 表名.字段=表的名.字段;或者select 字段 from 表名,表的名 where 表名.字段(+)=表的名.字段; 右外连接查询

select 字段 from 表名 full outer join 表的名 on 表名.字段=表的名.字段; 完全外连接查询

distinct 取唯一值

java连接oracle jdbc连接：

1.//加载驱动

Class.forName("oracle.jdbc.driver.OracleDriver");

2.//得到连接（数据库用户名和数据库登录密码根据数据库验证方式选择是否传入）localhost是本机IP地址，也可以是其他主机IP地址，1433是sql server的默认端口号

Connection connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:数据库实例名","数据库用户名","数据库登录密码");

sql语句分类：

dml语句 数据操作语句，需要手动提交，当退出控制台时oracle会自动提交

ddl语句 数据定义语言，oracle会自动提交

dql语句 数据查询语句

dtl语句 数据控制语言

oracle数据库看到的数据有可能和java程序看到的数据不一致，是因为事务控制原因，需要commit提交

to\_date(日期字符串,格式) 将日期字符串转成指定格式

oracle行迁移

create table 表名 as 查询语句

使用子查询完成更新

update 表名 set 字段=查询语句 where 条件

oracle数据库强大体现在用户管理，pl/sql编程，函数等方面丰富全面

oracle函数分类：

单行函数：对每行操作每行对应结果

单行函数分类：

字符函数：

replace(字段,需要替换的字符串,替换成的字符串) 替换返回结果，不改变数据库原本数据

instr(需要搜索的字符串,匹配的字符串,搜索的开始位置默认为1,到第几个匹配的字符串结束) 返回匹配的字符串在搜索的字符串中指定的位置

ascii(‘字符’) 返回指定的字符对应的十进制数

chr(整数) 返回整数对应的字符

oracle数据库虚表dual

concat(‘字符串’,’字符串’) 连接两个字符串

initcap(字符串) 将字符串首字母大写，其他字母小写返回

length(字符串) 返回字符串长度，汉字字母都为一个字符长度

lower(字符串) 返回字符串所有字符小写

upper(字符串) 返回字符串所有字符大写

lpad(字符串,填充后总长度,填充的字符串) 字符串左边填充字符串达到指定长度

rpad(字符串,填充后总长度,填充的字符串) 字符串右边填充字符串达到指定长度

ltrim(字符串,删除字符串) 字符串左边删除指定字符串

rtrim(字符串,删除字符串) 字符串右边删除指定字符串

substr(字符串,开始位置,取值个数) 从开始位置取字符串指定个数字符串

soundex(字符串) 返回字符串的发音规则

trim(指定字符串 from 字符串) 从字符串左右两边中去除指定字符串

数字函数：

abs(数值) 返回数值的绝对值

acos(数值) 返回数值反余弦的值

asin(数值) 返回数值的反正弦的值

atan(数值) 返回数值的反正切的值

ceil(数值) 返回数值的向上取整，返回大于或者等于数值的最小整数

floor(数值) 返回数值的向下取整，返回小于或者等于数值的最小整数

cos(数值) 返回数值的余弦值

cosh(数值) 返回数值的双曲余弦值

exp(数值) 返回e的数值次方值

ln(数值) 返回数值的对数值

log(数值,指定数值) 返回以数值为底指定数为指数的对数

mod(被除数,除数) 返回取模的值即被除数除以除数的余数

power(底数,指数) 返回底数的指数次幂的值

round(数值) 返回数值四舍五入后的值

trunc(数值,指定数值) 返回截取数值到小数指定数值位数部分，默认指定数值为0（即截取数值整数部分）

sign(数值) 返回正数（1），0（0），负数（-1）

sin(数值) 返回数值的正弦值

sinh(数值) 返回数值的双曲正弦值

sqrt(数值) 返回数值的2次方根

tan(数值) 返回数值的正切值

tanh(数值) 返回数值的双曲正切值

日期函数：

add\_months(日期,数值) 返回日期加减数值月份后的日期值

last\_day(日期) 返回日期月份的最后一天的日期值

months\_between(结束日期，开始日期) 返回开始日期到结束相差月份数

new\_time(日期,当地时区,转换时区) 返回日期从当地时区转换成指定时区的日期

next\_day(日期,’星期几’) 返回距离日期最近的下一个星期几的日期

sysdate 返回当前日期，默认不显示时间

转换函数：

chartorowid(字段) 将字符类型数据转换成rowid类型

rowidtochar(字段) 将rowid类型数据转换成字符类型

convert(字符串,字符集,指定字符集) 返回字符串从字符集转换到指定字符集

hextoraw(字段) 将十六进制转换成二进制

rowtohext(字段) 将二进制转换成十六进制

to\_char(列名,’格式’) 列转换成指定格式

to\_char(字符串/日期/数值,’格式’) 将数据转换成指定格式

时间格式：

yy：两位数字的年份

yyyy：四位数字的年份

mm：两位数字的月份

dd：两位数字的天

hh24：24小时制时间

hh12：12小时制时间

mi：分钟

ss：秒数

day：星期几

货币格式

9：显示数字，忽略0

0：显示数字，位数不足补0

.：在指定位置显示小数点.

,：在指定位置显示逗号,

$：在数值前加美元符号$

L：在数值前加本地货币符号

C：在数值前加国际货币符号

G：在指定位置显示分隔符

D：在指定位置显示小数点符号

to\_multi\_byte(字符串) 将字符串中单字节字符转换成多字节字符

to\_number(字符串) 字符串转数值

bfilename(文件名,指定文件) 将指定文件以文件名存入数据库

decode(字段,[字段对应值,返回对应字符串结果]) 根据字段的对应值返回相应的字符串结果

conuert(字段,指定内容,源) 将字段或者源转换成指定内容

dump(字符串,指定格式,开始位置,长度) 将字符串从指定开始位置到指定长度的字符串转换成指定格式

empty\_blob()，empty\_clob() 对大文件类型初始化操作

greatest(数据表达式[,数据表达式]) 返回数据表达式中最大的值，字符串比较对应编码大小

least(数据表达式[,数据表达式]) 返回数据表达式中最小的值，字符串比较对应编码大小

uid 返回当前用户的唯一标识用户id号

user 返回当前用户名

多行函数：对多行操作返回一行对应结果

avg([distinct|all],字段) 返回平均值，distinct忽略字段内重复值，默认all不忽略

max([distinct|all],字段) 返回最大值，distinct忽略字段内重复值，all不忽略

min([distinct|all],字段) 返回最小值，distinct忽略字段内重复值，all不忽略

stddev([distinct|all],字段) 返回标准差，distinct忽略字段内重复值，all不忽略

variance([distinct|all],字段) 返回协方差，distinct忽略字段内重复值，all不忽略

系统函数：

userenv() 返回当前用户环境信息

sys\_context(‘userenv’,指定信息) 返回用户指定信息

指定信息：

terminal：当前会话用户对应终端的唯一标识符

language：当前语言格式

db\_name：当前数据库名称

nls\_date\_format：当前会话用户对应的日期格式

session\_user：当前会话用户对应的数据库用户名

current\_schema：当前会话对应的默认方案名

host：数据库所在主机名称

oracle会在用户操作数据时加锁，防止其他用户也同时操作同一数据

提交事务commit后oracle会确认事务的变化，结束事务，删除保存点，释放锁，其他会话可以查看事务（transaction）变化后的数据

savepoint 保存点名 设置保存点

rollback to 保存点名 事务回滚到保存点，保存点只能回滚一次

rollback 取消所有事务

脏读：一个事务读取到另一事务未提交的操作后的数据

不可重复读：一个事务在不同时间由于其他事务提交更新删除操作造成读取数据不一致

幻读：一个事务在不同时间由于其他事务提交插入操作造成读取数据不一致

oracle数据库实现了国际标准中的read committed和serializable两个事务隔离级别，还有另外一个事务隔离级别read only

set transaction isolation level read committed（默认）/serializable;设置事务隔离级别

set transaction read only;级别与serializable一样，但只允许读操作

事务隔离级别国际标准化组织定义的事务处理级别ANSI/ISO SQL92标准



alter session set isolation level read committed/serializable;设置会话的隔离级别

java中设置事务隔离级别：

Connection.setTransactionIsolation(Connection.隔离级别名)

约束：

not null 指定列的值不能是null

unique(字段) 指定列的值是唯一的，不能重复，可以为null

primary key(字段) 主键，唯一标识一条记录，不能为null，不能重复

foreign key(字段) references 表名(字段) 外键，关联另一张表的主键

check(列名约束条件) 指定列名需要符合约束条件

alter table 表名 modify 字段名 not null; 修改表名的字段名约束不能为null

alter table 表名 add constraint 约束名 约束种类(字段); 为表添加约束

alter table 表名 drop constraint 约束名 [cascade]; 删除约束,cascade表示级联删除，忽略表之间的关系直接删除

字段后定义的约束是列级约束，所有字段后定义的约束是表级约束

表级约束定义：constraint 约束名 约束类型(字段)

oracle数据库创建自增长序列

create sequence 序列名

start with 开始的序列号

increment by 每次自动增长长度

minvalue 最小值

maxvalue 最大值（nomaxvalue 没有最大值，最大能多大就多大）

cycle 循环，自增到最大值后再重新从最小值增加（nocycle 没有循环）

nocache 没有缓存（cache 数值：表示一次产生数值个缓存序列号，可以提高效率，但是可能出现跳号问题）

;

insert into 表名 values(序列名.nextval（主键字段或者unique字段）[,字段]); 使用序列

多个用户可以使用同一序列，但序列号是接着增加不是从最小值增加

select 用户名.序列名.currval（nextval） from dual; 查看当前序列值（自增后的序列值）

drop 数据对象类型 数据对象名; 删除数据对象

序列刚创建完成需要nextval操作（第一次为初始化）后currval（当前值）才存在

oracle索引：

单列索引：基于单个列创建的索引

复合索引：基于两列或者多列的索引

create index 索引名 on 表名(字段[,字段])

索引一般建立在数据量大的表经常查询使用的列上，一般索引层次不超过4层，一般不建立在逻辑型字段或者固定值字段上，建立索引后表的空间大小一般是原来表的1.2倍

select \* from system\_privilege\_map order by name; 查看所有系统权限

grant 权限名称 to 用户名 [with admin option]; 授予用户权限，with admin option表示用户可以将获得的权限授予其他用户

revoke 权限名称 from 用户名; 回收系统权限，不会级联回收

select distinct privilege from dba\_tab\_privs; 查看所有对象权限

grant 权限名称 on 数据库对象 to 角色名/（用户名 [with grant option]）; 授予用户权限，with grant option表示用户可以将获得的权限授予其他用户

revoke 权限名称 on 数据库对象 from 用户名[,角色名]; 回收对象权限，级联回收

select \* from dba\_roles; 查看所有角色

grant 角色名 to 用户名 [with admin option]; 授权用户担任角色，with admin option表示用户可以将担任的角色授予其他用户

select \* from role\_sys\_privs where role='角色名（字母大写）'; 查看角色的系统权限

select \* from dba\_sys\_privs where grantee='角色名（字母大写）'; 查看角色的对象权限

resource角色有个unlimited tablespace隐藏系统权限是查不到的，unlimited tablespace系统权限表示可以无限制使用表空间

select \* from dba\_role\_privs where grantee='用户名（字母大写）'; 查看用户担任的角色

dba角色不能启动和关闭数据库

create role 角色名 （not identified）/（identified by 密码）; 创建不需要/需要验证的角色

unlimited tablespace系统权限不能授予角色

drop role 角色名; 删除角色，级联删除

pl/sql（procedural language/sql）oracle在标准sql语言上的扩展

create [or replace] procedure 存储过程名(输入/输出参数名 in/out（默认输入参数） 输入/输出参数类型[,输入/输出参数名 in/out 输入/输出参数类型]) is/as

begin

执行语句;

end[/存储过程名];

/

创建存储过程，or replace表示当存储过程名重复替换原存储过程

exec/call 存储过程名(参数); 控制台执行存储过程

java程序执行存储过程：

CallableStatement=Connection.prepareCall("{控制台执行存储过程语句}");

CallableStatement.registerOutParameter(存储过程语句输出参数号, oracle.jdbc.OracleTypes.输出参数类型); 注册输出参数

CallableStatement.execute();

CallableStatement.getString(存储过程语句输出参数号); 取出输出参数的输出值

pl/sql编程规范：

单行注释—

多行注释/\*\*/

定义变量v\_作为前缀

定义常量c\_作为前缀

定义游标\_cursor作为后缀

定义例外e\_作为前缀

块（block）是pl/sql的基本程序单元

pl/sql块由定义部分、执行部分、例外处理部分组成：

declare

/\*定义部分，可选，定义常量、变量、游标、例外、复杂数据类型\*/

begin

/\*执行部分，必选，需要执行的pl/sql语句和sql语句\*/

exception

/\*例外部分，可选，处理运行的错误\*/

end;

dbms\_output是oracle提供的包，put\_line是dbms\_output包的一个存储过程

dbms\_output. put\_line(‘字符串’||变量名); 输出显示字符串与变量值的内容

set serveroutput on; 设置显示输出，默认情况下，oracle不显示输出

定义变量的格式是：变量名称 变量类型

异常的基本语法：

exception

when 异常名称 then

异常处理代码，可以是多行

[when 异常名称 then

异常处理代码，可以是多行]

end;

show error; 显示异常具体错误信息，默认不显示

函数基本语法：

create [or replace] function 函数名(参数名 参数类型[,参数名 参数类型])

return 数据类型 is

定义变量;

begin

执行语句;

return 数据类型;

end;

包的基本语法：

create [or replace] package 包名 is

procedure 过程名(变量名 变量类型);

function 函数名(变量名 变量类型) return 数据类型;

end;

包内的过程与函数只是声明并未实现，在包体中实现过程与函数

exec 包名.过程名/函数名(参数); 控制台调用包内的过程或者函数

exec 用户名.包名.过程名/函数名(参数); java程序调用包内的过程或者函数

包体的基本语法，包体与包是互相对应的

create [or replace] package body 包名 is

procedure 过程名(变量名 变量类型) is

begin

执行语句;

end;

function 函数名(变量名 变量类型) return 数据类型 is

定义变量

begin

执行语句;

return 数据类型;

end;

end;

pl/sql的数据类型分类：

标量类型（scalar）：

变量和常量定义：

名称 [constant] 数据类型 [not null] [:=初始值/（pl/sql表达式）];

constant：指定是否为常量

not null：指定变量值能否为null

:= 表示赋值，= 表示判断是否相等

变量名 表名.列名%type; 定义%type类型变量，自适应变量

复合类型（composite）：

pl/sql记录：

type 记录名 is record(

变量名 变量类型,

变量名 变量类型

);

记录名.变量名 调用记录名里的变量

pl/sql表：

type 记录表名 is table of 表名.列名%type

index by binary\_integer;

记录名(数值) 下标为数值的元素

参照类型（reference）：

type 游标名 is ref cursor;

变量名 游标名; 定义游标

open 游标变量名 for 查询语句; 打开游标

close游标变量名; 关闭游标

fetch 游标变量名 into 其他变量; 去除当前游标指向的行

游标变量名%notfound 判断游标是否指向记录末尾

lob（large object）

pl/sql条件控制语句：

判断语句：

if 条件表达式 then

执行语句;

[elsif 条件表达式 then

执行语句;]

[else

执行语句;]

end if;

case

when 条件 then 执行语句;

[when 条件 then 执行语句;]

end case;

循环语句：

loop

执行语句;

exit when 条件表达式;

end loop;

while 条件表达式 loop

执行语句;

end loop;

for 变量 in reverse 开始数值..结束数值 loop

执行语句;

end loop;

在定义部分定义变量只能赋初始值操作，不能进行变量赋值操作

输入变量不能进行赋值操作

pl/sql顺序控制语句：

<<标号名>> 创建标号

goto 标号名; 跳转到指定标号，继续执行语句

null; 空操作语句，无操作，增加代码可读性

存储过程返回多个值：

创建一个包，在包中定义游标

create [or replace] package 包名 is

type 游标名 is ref cursor;

end;

创建存储过程

create [or replace] procedure 过程名(输入/输出参数名 in/out（默认输入参数） 输入/输出参数类型,游标变量名 out 包名.游标名) is

begin

open 游标变量名 for 查询语句;

[close 游标变量名] 关闭游标，使用java程序时在java程序中关闭

end;

java中使用存储过程

CallableStatement.registerOutParameter(存储过程语句输出参数号, oracle.jdbc.OracleTypes.输出参数类型); 注册输出参数

CallableStatement.execute();

ResultSet=(ResultSet) callableStatement.getObject(存储过程语句输出参数号);取出输出参数的输出值

create [or replace] view 视图名 as 查询语句 [with read only]; 创建或修改视图，or replace表示替换，with read only表示不能通过视图对数据进行读以外的操作

drop view 视图名; 删除视图

视图不占用空间，不能添加索引

create [or replace] trigger 触发器名

before/after

insert/delete/update [of 列名] on 表名

[for each row]

[when 条件]

begin

执行语句

end;

创建触发器，for each row表示行级触发器，默认列级触发器，:old.列名表示修改修改前对应列的值，:new.列名表示修改修改后对应列的值

procedure raise\_application\_error(自定义错误编号（-2000到-20999，其他已经被oracle占用 ） in number,自定义错误提示信息 in varchar2（长度不超过2k，超过部分忽略）); 自定义错误，强制终止操作

ora\_client\_ip\_address 返回客户端的IP地址

ora\_database\_name 返回数据库的名称

ora\_login\_user 返回登录的用户名

ora\_sysevent 返回触发触发器的系统事件名

ora\_des\_encrypted\_password 返回用户des（MD5）加密后的密码

系统触发器：

create [or replace] trigger 触发器名

before/after logon/logoff on database

begin

执行语句

end;

ddl触发器

create [or replace] trigger 触发器名

after ddl on 方案名.schema

begin

执行语句

end;

alter trigger 触发器名 disable/enable; 暂时禁止/激活触发器

alter table 表名disable/enable all triggers;暂时禁止/激活表的所有触发器

drop trigger 触发器名; 删除触发器

begin

exception

[when 异常情况名 then

异常处理代码;]

[when others then

异常处理代码;]

end;

捕获异常，others表示默认出列出的情况外剩下所有情况

show parameter; 显示初始化参数

修改初始化参数在安装目录oracle\admin\orcl\pfile\init.ora文件中修改

oracle数据库导出分类：导出表，导出方案，导出数据库

导出命令需要在系统命令行窗口执行，一般导出的备份文件后缀是.dmp

exp userid=用户名/密码@数据库实例名 tables=(表名[,表名]) file=备份路径; 导出数据库表

exp userid=用户名/密码@数据库实例名 tables=(方案名.表名[,方案名.表名]) file=备份路径; 导出其他方案的表，需要dba权限或者exp\_full\_database

exp userid=用户名/密码@数据库实例名 tables=(表名[,表名]) file=备份路径 rows=n; 导出数据库表的结构

exp userid=用户名/密码@数据库实例名 tables=(表名[,表名]) file=备份路径 direct=y; 直接导出方式，导出速度快但数据库字符集要与客户端字符集一致，否则出错

exp userid=用户名/密码@数据库实例名 owner=用户名 file=备份路径; 导出数据库方案

exp userid=用户名/密码@数据库实例名 owner=(用户名[,用户名]) file=备份路径; 导出其他方案，需要dba权限或者exp\_full\_database

userid：指定执行导出操作的用户名，口令，连接字符串

tables：指定执行导出操作的表

owner：指定执行导出操作的方案

full=y：指定执行导出操作的数据库

inctype：指定执行导出操作的增量类型

rows：指定执行导出操作是否导出表中数据

file：指定导出文件名



struts（结构）是一个开源的web框架（frameset）

优点：程序更加规范，开发效率提高，程序可读性增加，程序可维护性增加

缺点：form表单有点不好，action是单态（影响网站并发性的处理），框架提过程序规范，约束程序的自由

struts-config.xml是struts的一个核心文件

struts组件一般包含：ActionServlet（总控制器），ActionForm（存放数据的表单），Action（分控制器），model（模型），jsp（页面）

struts开发一般步骤：

下载导入struts开发包

基础页面设计开发（jsp，业务功能界面）

开发ActionFrom和Action（servlet控制器）

配置struts-config.xml文件

友好界面设计开发（jsp，成功错误跳转页面）

在web.xml中配置ActionServlet

struts处理中文乱码问题（过滤器）

存放数据的表单类需要继承ActionForm抽象类，数据类中的成员函数设置获得属性值（set/get方法）必须与表单属性名（name属性值）相对应，一般为了方便数据类中的成员变量属性名与表单属性名（name属性值）一致

eclipse提示快捷键Alt+/

分控制器类需要继承Action类，并重写public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest request,HttpServletResponse response)函数

struts-config.xml文件配置ActionForm与Action、以及对应关系、跳转位置，一般文件放在/WEB-INF目录下

struts-config.xml文件配置：

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

<!DOCTYPE struts-config PUBLIC ".//Apache Softwate Foundation//DTD Struts Configuration 1.3//EN" "http://struts.apache.org/dtds/struts-config\_1\_3.dtd">

<struts-config>

<form-beans>

<form-bean name="表单名，一般规范化是小写表单类名" scope="request/session（表单类数据保存在的对象，默认session）" type="指定表单类的全路径（包含包名）"></form-bean>

</form-beans>

<action-mappings>

<action path="网址访问的资源名称一般为（/资源名）" name="关联表单类的表单名" type="指定控制器类的全路径" attribute="属性名（放入数据对象中对应的属性名）" scope="request/session（表单类数据保存在的对象，默认session）validate="false/true（是否对表单进行校验）">

<forward name="指定跳转名" path="指定跳转的页面"></forward>

</action>

</action-mappings>

</struts-config>

form-beans：配置表单

action-mappings：配置所有分控制器

action：配置分控制器

forward：配置跳转页面

web.xml文件配置：

<servlet>

<servlet-name>总控制器名</servlet-name>

<servlet-class>指定总控制器全路径（一般是引用路径org.apache.struts.action.ActionServlet）</servlet-class>

<init-param>

<param-name>config</param-name>

<param-value>指定配置信息路径即struts-config.xml的路径 [,struts-config.xml的路径（多个配置文件）]</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>总控制器名</servlet-name>

<url-pattern>指定请求都通过总控制器（一般是\*.do）</url-pattern>

</servlet-mapping>

mapping.findForward("指定页面的跳转名"); 跳转页面

提交表单数据后，总控制器通过反射机制实例化对应表单类，并调用相应的set方法存放表单数据

struts1的所有控制器（包括总控制器）都是单态的，即在整个程序运行工程中，始终是一个对象

jstl（jsp standard tag libaray）jsp标准标签库

优点：

提供应用程序服务器之间统一的接口，web应用可以在不同服务器之间移植

简化jsp在web应用程序中的开发

减少jsp中java片段甚至没有，代码更简洁

提高jsp开发速度，缩短开发周期

jsp使用jstl标签库需要下载相应的jar包，并在jsp开始部分引入

tld：tag library description标签库描述文件

<%@ taglib prefix="标签前缀（一般是c）" uri="对应标签的资源地址名（在对应的tld文件中）"%>

jstl标签分类：

一般标签：

<c:out value=”在页面上输出的内容/${属性名.变量名}” default=”当value没有值或未找到对应属性名时默认输出的内容” escapeXml=”输出value值的形式[true（文本）/false（html形式）]”></out>：

相当于<%out.println(“内容/( (属性名)数据对象域.getAttribute(“属性名”) ).get变量名”) %> 不同对象域中相同属性名在c:out中的优先级从高到低pageContext，request，session，application

<c:set var=”属性名” value=”属性值” scope=”存放的数据对象域”></c:set>

相当于<%数据对象域.setAttribute(“属性名”,”属性值”) %>

<c:remove var=”属性名” scope=”数据对象域（默认所有数据对象域）”></c:remove>删除指定数据对象域中的指定属性名的值

<c:catch var=”异常名”></c:catch>将捕获的异常信息存入指定异常名中，取出异常信息可以使用异常名.message

条件标签：

<c:if test=”${判断表达式（变量值会根据判断类型转换成相应的数据类型，字符串使用单引号，对象名.属性名）}”>执行的语句</c:if> 对象名.属性名相当于对象名.get属性名()的调用

jsp可以有多个引入指令

<c:choose>

<c:when test=”${判断表达式}”></c:when>

[<c:when test=”${判断表达式}”></c:when>]

<c:otherwise></c:otherwise>

</c:choose> 多分支判断语句相当于switch，case，default语句

迭代标签：

<c:forEach items=”${属性名}” var=”变量名（相当于for循环的i变量）” begin=”变量名取值开始位置” end=” 变量名取值结束位置” step=”变量名每循环一次取值长度”>循环执行的语句</c:forEach>相当于for循环语句，将指定属性名的集合对象中每个对象取出放入变量名中

<c:forTokens items=”${属性名}/（字符串 [，字符串]）” delims=”分割items的分隔符（，）” var=”变量名（相当于for循环的i变量）” begin=”变量名取值开始位置” end=” 变量名取值结束位置” step=”变量名每循环一次取值长度”>将指定属性名的集合对象或字符串中通过分割符分割后依次放入变量名中

判断变量是否为空的表达式：${empty/（!empty） 变量名}

url相关标签：

<c:redirect url=”url地址”></c:redirect> 重定向到指定url地址

<c:import url=”url地址”> 将指定url地址文件导入本jsp页面中

<c:param name=”属性名” value=”属性值”></c:param> 引入文件时传入的参数以及url地址跳转传入的参数都是通过${param.属性名}获取

</c:import>

struts标签分类：

html标签：

<html:button property=”名字” value=”值”></html:button>相当于<input type=”button” name=”名字” value=”值”>

<html:base ref=”url地址”></html:base> 设置超链接默认url地址

<html:image src=”图片地址”></html:image>相当于<input type=”image” name=”” src=”图片地址”> 显示指定图片按钮

<html:img src=”图片地址”/>相当于<image src=”图片地址”> 显示指定图片

<html:link href=”url地址”></html:link>相当于<a href=”url地址”></a> 显示指定的超链接

struts的有信息输入的html标签需要放在表单标签（<html:form></html:form>）中，否则会出错

get提交方式可以让浏览器收藏页面url地址时把url地址后面的参数一起收藏

struts的html标签与普通的html标签可以一起使用

多个过滤器执行顺序是根据部署顺序决定的

bean标签：

<bean:write name=”属性名” property=”对象的成员变量名” scope=”数据对象域（默认所有数据对象域）”/> 显示数据对象域中属性名对应的对象的指定成员变量名的值

property不是根据成员变量名指定属性名，而是根据set属性名()和get属性名()中的属性名指定属性名

使用工具引入struts会自动生成资源文件ApplicationResources.properties，资源文件中存入参数的格式：

变量名={参数个数号（0到4）} 外部输入的参数根据参数号可以替换指定位置的参数

#号表示注释

<bean:message key=”变量名” arg参数号（0到4）=”参数值”/> 将资源文件中变量名指定参数号的参数替换后显示

资源文件中中文字符需要转换成ASCII码

jdk中ASCII码转换器在jdk安装目录下的bin\native2ascii.exe

<bean:define id=”标识号” name=”变量名” value=”变量值” scope=”存入的数据对象域” property=”属性名”/> 定义变量或者对象存入数据对象域中

<jsp:useBean id=”对象名” class=”类名（包含包名）”></jsp:useBean> 实例化一个指定类名的对象

<jsp:useBean name=”对象名” property=”成员变量名” value=”值”></jsp:useBean> 给对象的指定成员变量赋值

logic标签

<logic:iterate id=”对象名” name=”属性名”>每个对象执行语句</logic:iterate> 将指定属性名对应的集合中的对象依次取出放入指定对象名中

一般标签选择使用顺序是jstl标签，el表达式，struts标签

<logic:empty/notEmpty name=”属性名”>执行语句</logic:iterate> 指定属性名是否空执行相应语句

<logic:greaterThan name=”对象名” property=”成员变量名” value=”值”>执行语句</logic:greaterThan > 对象的指定成员变量名的值大于指定值执行相应语句

控制器action继承DispatchAction（分派控制器）用于处理一类请求，配置分派控制器是指定parameter属性名称，属性名称对应的值用于指定请求处理的函数

使用分派控制器函数名一般自定义，指定处理的函数是在跳转重定向时在url地址后面加入参数parameter属性名称=属性值（指定处理函数的函数名），默认不加参数会指定默认函数execute()进行处理

数据对象域.getSession().invalidate(); 设置所有session失效

数据对象域.getSession().removeAttribute(属性名); 设置指定属性名的session失效

action标签中的forward标签是局部跳转，这个forward只能被这个action使用

<global-forwards>

[<forward name=”跳转名” path=”跳转路径”></forward>]

</global-forwards> 全局跳转，所有的action都能通过跳转名进行跳转

dynimic form（动态表单）不需要创建一个表单类直接在struts-config.xml中进行配置：

<form-beans>

<form-bean name=”表单名” type=”表单类型（org.apache.struts.action.DynaActionForm）”>

[<form-property name=”参数名” type=”参数类型”></ form-property>]

</form-bean>

</form-beans>

动态表单名.getString(“参数名”); 取出动态表单的指定参数名的参数值

动态表单名.get (“参数名”); 取出动态表单的指定参数名的参数对应的对象

表单中有文件控件需要重新指定表单的编码方式：

<form enctype=”multipart/form-data” ></form>

FormFile 文件名 定义表单文件变量

<input type="file"/> jsp页面文件输入

页面文件上传需要通过控制器将文件以流的形式存入服务器内存中

InputStream 文件流名称=FormFile.getInputStream(); 获得表单文件输入流对象

String 变量名=UUID.randomUUID().toString(); 获得一个不会重复的16进制数

int 变量名=字符串.lastIndexOf(“指定字符串”); 获得指定字符串在字符串最右边出现的位置

字符串.substring(开始位置（从0开始）,结束位置（不包括结束位置）); 返回字符串从开始位置到结束位置的截取子字符串

上传文件名称中文乱码问题可以通过过滤器解决

response.setHeader(“Content-Disposition”,”attachment;filename=下载的文件名”); 下载文件时设置返回浏览器的响应信息，表示要下载文件

java.net.URLEncoder.encode(“中文字符串”,”编码格式”); 下载的文件名中文需要转换

所有文件类型种类可以在web.xml文件中查看

hibernate是一个ORM（object/relation mapping）对象/关系映射框架，是jdbc的轻量级封装，一般用于项目持久层，

hibernate的核心配置文件hibernate.cfg.xml

hibernate的基础是java的反射机制

使用hibernate数据库表相对应的java类一般称为pojo（plain ordinary java object）简单的java对象

对象关系映射文件指定domain（域）对象和数据库表的映射关系，文件名一般规范是domain对象名.hbm.xml，位置一般放在对应domain对象同一个文件中

对象关系映射文件配置需要引入指定格式的dtd文件：

<!DOCTYPE hibernate-mapping PUBLIC “-//Hibernate/Hibernate Mapping DTD 3.0//EN” “http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd”>

对象关系映射文件内容一般格式：

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="类所在包名">

<class name="数据域对象名（类名）" table="数据库表名">

<id name="类成员属性名" column="对应数据库表列名" type="数据类型（全称）">

<generator class="产生主键值的策略">（默认assigned输入参数指定值）

<param name="产生主键值的策略">对应数据库中的策略名</param>

</generator>

</id>（id标签配置表主键字段，property标签配置普通字段）

[<property name="类成员属性名" type="数据类型（全称）">

<column name="对应数据库表列名" length="指定字段长度" not-null="字段是否允许为空（默认true允许）"></column>

</property>]

[<many-to-one name="类的对象属性名" column="数据库中关联的表外键名"></many-to-one>]（多对一关系关联）

[<set name="类的对象集合属性名" cascade="save-update（cascade表示级联操作：对数据对象进行操作的时候是否将其关联的数据对象也执行相应操作，save-update表示级联更新）">

<key column="数据库中关联表的外键名"></key>

<one-to-many class="类的对象集合属性的类名"/>

</set>]（一对多关系关联）

[<id name="类成员属性名" column="对应数据库表列名" type="数据类型（全称）">

<generator class="foreign">（一对一主键关联关系的外键生成策略）

<param name="property">类的对象属性名</param>

</generator>

</id>]（一对一主键关联关系的外键配置）

[<one-to-one name="类的对象属性名" constrained="关联是否生成外键，默认false"></one-to-one>]（一对一主键关系关联）

[<many-to-one name="类的对象属性名" unique="true"></many-to-one>]（一对一的外键关系关联hibernate会当成多对一标签的特殊情况来处理）

</class>

</hibernate-mapping>

hibernate.cfg.xml文件内容一般格式：

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

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="connection.driver\_class">数据库驱动</property>

<property name="connection.username">用户名</property>

<property name="connection.password">密码</property>

<property name="connection.url">连接数据库的url地址</property>

<property name="dialect">指定连接的数据库名称（相关信息在hibernate文件中的project\etc \hibernate.properties配置文件中）</property>

<property name="show\_sql">true</property>（对数据库操作时控制台输出显示hibernate对应的sql语句）

<property name="current\_session\_context\_class">thread/jta</property>（配置当前session与thread/jta绑定，thread表示本地事务，jta表示全局事务（即多个数据库的事务），可以使用getCurrentSession()方法获得session）

<property name="format\_sql">true</property>（对数据库操作时控制台输出显示hibernate对应的格式化后的sql语句）

<property name="hbm2ddl.auto">create/update/create-drop/validate</property>

（对象映射文件和相应的类指定完成时，hibernate会根据这些文件自动创建对应的数据库，create表示没有对应的表则创建对应的表，有则覆盖创建，update表示没有对应的表则创建对应的表，有表且表结构发生变化则改变表结构，没有变化则不改变表结构，create-drop表示创建新表，当sessionFactory设置关闭后将删除schema，validate表示每添加数据前会验证数据库表的结构与配置文件中的结构是否一致，一般建议使用update）

<mapping resource="对象映射文件的路径"/>（指定管理的对象映射文件）

</session-factory>

</hibernate-configuration>

Configuration 配置名=new Configuration().configure(“配置文件名（默认hibernate.cfg.xml）”); 读取配置文件，并完成初始化

SessionFactory 会话工厂名=configuration.buildSessionFactory(); 创建会话工厂对象，一般一个会话工厂对象对应一个数据库，通常设置会话工厂为单态的，

Session 会话名=会话工厂名.openSession(); 创建会话对象，相当于Connection对象

会话名.save(对象名); 将对象信息通过会话保存到对象对应的数据库表中即对应的数据库表插入一条相应对象信息的记录

会话名.close(); 关闭会话

hibernate强制统一要求对数据库的除查询以外的操作为事务，需要进行事务提交

Transaction 事务名=会话名.beginTransaction(); 创建会话对应的事务

事务名.commit(); 提交事务

hibernate数据库表对应的类对象一般规范需要序列化，实现Serializable接口即class 类名 implements java.io.Serializable

类名 记录对象名=(类名)会话.load(类名.class, 主键值); 通过主键值获取对应数据库表中记录的对象信息

类名 记录对象名=(类名)会话.get(类名.class, 主键值); 通过主键值获取对应数据库表中记录的对象信息

类名.class获取类的class对象

记录对象名.设置函数("值"); 通过load加载后的记录对象可以通过设置对象的属性值对数据库表中对应的记录进行修改，hibernate默认修改是对对象所有成员属性进行修改，未通过设置函数修改的属性默认修改为原属性值

会话名.delete(记录对象名); 删除记录对象对应的数据库表中的记录

hql语句（hibernate query language）

ORM框架除了hibernate外还有Apache OJB，Cayenne，Jaxor，iBatis，jRelationalFramework ，mirage，SMYLE，TopLink，ejb cmp

hibernate的核心配置文件hibernate.cfg.xml文件可以用hibernate.properties配置文件来代替

hibernate框架的configuration类负责管理hibernate的配置信息，读取加载hibernate.cfg.xml配置文件中的信息已经配置的驱动，管理\*.hbm.xml对象关系文件

SessionFactory会话工厂可以缓存sql语句和数据（称为session级缓存）

通过SessionFactory获取Session的两个方法的区别：

openSession()是获取一个新的session，需要进行session资源关闭操作

getCurrentSession()是获取和当前进程绑定的session，即在同一个进程中的session是同一个，这样有利于事务控制，并且在事务提交或者回滚后系统会自动关闭session资源，但是对数据库的任何操作都要封装成事务包括查询操作

Session默认线程不同步

Session的两个对数据库查询操作的方法get()和load()的区别

get()查询数据会先去缓存（session缓存/二级缓存）查找，未查到就立即向数据库发出对应的sql执行语句，查不到数据会返回null

load()查询数据会先去缓存（session缓存/二级缓存）查找，未查到会返回一个代理对象，当使用到查询结果内容的时候才会向数据库发出对应的sql执行语句（这种现象称为延时加载，可以通过修改配置文件中<class name="数据域对象名（类名）" lazy="true/false" table="数据库表名">的lazy属性值（默认true）修改加载方式），查询不到数据会报错ObjectNotFoundException

建议确定数据库中有要查找的数据时用load()方法，不确定时用get()方法

ThreadLocal<类型> 名称 定义线程局部模式，将指定类型的对象与线程绑定即在线程作用域中指定数据类型的对象时同一个

Session 会话名=threadLocal.get(); 从线程中获得绑定的会话对象

threadLocal.set(会话名); 将会话绑定到线程中

HQL（Hibernate Query Language）是面向对象的查询语言，与SQL不同，对象名java类和属性区分大小写其他不区分大小写，查询针对数据库表映射的数据域类不是数据库表，支持多态，语句主要通过Query来实现

List<domain类名> 队列名=会话名.createQuery("（hql查询语句）from（数据域类名）Employee where （条件）id=3").list(); 查询结果会自动封装成对应的domain对象存入队列中，Query是对数据对象进行操作不是对数据库表，查询的数据域类名是数据库表对应的类名，条件中可以是数据库表对应的字段名也可以是类名对应的属性名

Criteria是比HQL更面向对象的查询方式

List<domain类名> 队列名=会话名.createCriteria(查询条件可以是类名.class查询整个类对应的数据库表所有内容).setMaxResults(显示最大结果个数).add(过滤条件).list(); 通过设置相应信息查询相应结果自动封装成对应的数据对象存入队列中

sql语句一般建议查询语句只查需要的字段，hql语句一般建议查询所有，因为hql语句的优势是面向对象的，查询结果会被封装成对象，方便关联查询等操作

from 类名 [where 条件] hql语句查询所有属性

select 属性名[,属性名] from 类名 [where 条件] hql语句查询部分属性

类名 对象名=会话名.createQuery(“hql语句”).uniqueResult; 确定要查询的数据最多只有一条记录存在，指定查询语句查询出一条记录即可停止查询，减少查询时间提高效率，存在多条记录会报错

hql语句类属性名前加distinct表示对该属性值进行过滤去重

hql语句条件（属性名 between 值 and 值）表示查询指定两个值之间的的属性名的属性值

hql语句条件（属性名 in/not in）表示查询在/不在指定值内的属性名的属性值

hql语句条件（group by 属性名）表示查询结果按照属性名进行分组

属性名 as 别名 hql语句指定属性名的别名

hql语句的having条件和聚合函数与数据库中用法相同，但是对类和属性名操作的

hql语句查询一列数据时返回的是对象不是对象数组

hql语句类名.属性名表示对应数据库表.字段名

List<domain类名> 队列名=会话名.createQuery("hql语句").setFirstResult(开始记录数（第一条记录数值为0）).setMaxResult(显示记录个数); 查询结果分页显示从指定开始记录数开始显示指定个数条记录

Query 查询名=会话名.createQuery("hql语句（输入参数用?/:参数名表示）"); 通过输入参数的方式防止sql注入漏洞

查询名.setString(参数序号/”参数名”,参数值); 通过输入参数的方式防止sql注入漏洞

hql查询语句也可以配置到对象关系映射文件中：

<query name="查询名"><![CDATA[hql语句]]></query>

List<类名> 队列名=会话名.getNameQuery("查询名").list(); 程序中调用对象关系映射文件中的hql查询语句

List<类名> list=会话名.createCriteria(类名.class).add(Restrictions.gt("属性名",指定数值)); 查询属性值对应的字段大于指定数值的数据记录

hibernate数据对象的三种状态：

瞬时态（transient）：数据库中没有与之相对应的数据存在，超出作用域会被java虚拟机回收器回收，一般是数据对象被创建但没有与session关联的状态

持久态（persistent）：数据库中有与之相对应的数据存在，数据对象已经与session关联且session未关闭的状态，hibernate会自动检测持久态下的数据对象中的数据发生的变化相应的在提交事务时会改变数据库中对应的数据发生对应的变化

游离态（detached）：据库中有与之相对应的数据存在，数据对象没有与session关联或者session已经关闭的状态，游离态下的数据对象中的数据发生变化不会影响数据库中与之对应的数据发生相应变化

由于延时加载原因，关联查询关联数据时，hibernate会向数据库发送两次查询，第一次返回结果是没有实际的关联数据的代理对象，当使用到关联数据时才会发出第二次查询关联数据，当使用关联数据在session（会话）结束时会报错误，原因是hibernate发出第二次查询时session已经关闭无法查询

解决延时加载的一种方法：通过修改配置文件中<class name="数据域对象名（类名）" lazy="true/false" table="数据库表名">的lazy属性值（默认true）修改加载方式

解决hibernate延时加载的方法：

在对象映射文件中配置关闭延时加载

Hibernate.initialize(查询的对象名.get关联的类名());初始化代理对象

在session未关闭是，使用关联对象的数据，强制数据库进行访问查询操作

Web项目既解决延时加载又解决hibernate发送多余SQL语句的方法：openSessionInView在web接受请求的位置设置一个过滤器，该过滤器创建session并且在过滤器结束的时候再关闭session，请求响应过程中使用同一个session（过滤器创建的session）并且不进行关闭。

基础接口的参数类型无法确定的情况可以用Serializable接口这个类型来代替大部分常用参数类型，因为大部分常用参数类型都继承了Serializable接口

Web应用一般用<%=this.getServletContext().getContextPath() %>或者${pageContext.request.contextPath}代替web应用名称的路径，提高代码的灵活性

Hibernate实现延时加载需要重写数据域（domain）对象，所以实现延时加载数据域（domain）对象必须是非final对象

Hibernate会向一级缓存中放入数据的数据库操作：save，update，saveOrUpdate，load，get，list，iterate，lock

Hibernate会首先从一级缓存中取出数据的数据库操作： load，get

Session.evict(数据对象名); 清楚hibernate指定数据对象名的一级缓存（Session级缓存）

Session.clear(数据对象名); 清楚hibernate所有一级缓存（session级缓存）

Hibernate一级缓存不需要配置但不能控制缓存数量，缓存大批量数据时会有内存溢出的可能，一级缓存的生命周期与session关联，session关闭时一级缓存会自动清空

Hibernate由于一级缓存缓存的数量有限，所以有二级缓存（SessionFactory缓存），hibernate二级缓存是交给第三方处理的，所以二级缓存需要在hibernate的配置文件hibernate.cfg.xml中配置并且引入第三方的包和相应的缓存配置文件才能使用，常用的二级缓存有Hashtable，EHCache，OSCache

Hibernate二级缓存配置：

<session-factory>

<property name="cache.use\_second\_level\_cache">true</property>（指定使用二级缓存）

<property name="cache.provider\_class">二级缓存的名称（包含包名）</property>（指定使用的二级缓存的名称）

<property name="hibernate.generate\_statistics">true</property>（指定是否使用hibernate二级缓存统计信息）

<class-cache usage="read-only/read-write/nonstrict-read-write/transactional" class="数据域类名（包含包名）"/>（指定使用二级缓存的策略（只读，读写，不严格读写，事务）和使用二级缓存的数据域对象，这个配置可以在对应的对象映射文件中配置，标签名为cache）

</session-factory>

Statistics 名称=SessionFactory.getStatistics(); 获得hibernate二级缓存所有统计信息

主键值产生策略：

increment：自动递增，增量固定为1不能修改，适用于所有数据库系统，不适用于多进程，主键必须为数值类型

identity：适用于支持identity数据库系统，主键必须为数值类型

sequence：适用于支持sequence数据库系统，主键必须为数值类型

native：会根据数据库系统类型自动从identity、sequence和hilo中选择适合的策略，主键必须为数值类型

hilo：由hibernate按照high/low算法产生值，从指定的数据库表和字段中获取值（需要配置相应的表和字段），适用于所有数据库系统，主键必须为数值类型

uuid：由hibernate基于128位bit唯一值产生算法，根据当前设备ip、时间、jvm启动时间、内部自增量4个参数生产16进制数值，一般主键使用字符串String类型

assigned：自定义设计指定主键值

映射复合主键：

<composite-id name="复合主键名" class="复合主键生成类名（包含包名）">（指定复合主键的名称以及复合后的类名）

[<key-property name="数据域属性名" type="数据域属性类型">

<column name="数据域属性对应的数据库字段名"></column>

</key-property>]（指定需要复合的主键）

</composite-id>

foreign：主键值与另一张表的主键相关联

主键类型是数值类型的oracle数据库一般生成策略是sequence、mysql数据库一般是increment/assigned、sql server数据库一般是identity/native/assigned

主键类型使字符串String类型一般生成策略是uuid/assigned

Hibernate不适合OLAP（On-Line Analytical Processing联机分析处理）以查询分析数据为主的系统，适合OLTP（On-Line Transaction Processing联机事务处理），hibernate批量操作数据的效率不高，适合数据量不大，性能要求不是很高，事务操作比较多的系统

Spring是容器框架，用于配置管理bean（java中的任何一个对象）和维护bean之间关系的框架，配置管理项目的各个组件

spring的核心配置文件applicationContext.xml一般放在src目录下，配置时需要引入指定格式的xsd文件，applicationContext.xml文件的一般格式：

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<context:annotation-config></context:annotation-config>（启动注解）

<context:property-placeholder location="classpath: 文件名（包含包名）[,classpath: 文件名（包含包名）]"/>（分散配置引入分散文件，通过${属性名}取出文件中对应的属相值）

[<bean id="实例化对象名" class="管理的类名（包含包名）" parent="继承的父类的实例化对象名" scope="singleton/prototype/request/session/global-session（作用域，默认singleton，单例模式）" init-metho="指定初始化方法名" destroy-metho="指定销毁方法名" autowire="byname/byType/constructor/autodetect/no（bean对应类的关联关系未设置会自动装配，默认值由beans标签中的default-autowire属性的值决定，default-autowire表示所有bean的默认自动装配方式，默认值为no）">（指定bean在实例化对象以及销毁时调用指定的初始化函数和销毁函数进行处理，函数是bean对应类的成员函数）

[(<property name="类的属性名">

<value>属性值</value>

</property>)/(<property name="类的属性名" value="属性值"></property>)]

[<property name="类属性的名称" ref="引用的类的spring实例化对象名"></property>]（类属性的配置，将引用的类的spring实例化对象名对应类设置为类的属性，在类中的名称设置为类属性的名称）

</bean>]（配置spring管理的bean，当spring加载时会自动创建实例化管理的类的对象名称为实例对象名，并将类的属性名的值设置为指定的的属性值）

<bean class="PropertyPlaceholderConfigurer">

<property name="locations">

<list>

[<value>文件名（包含包名）</value>]

</list>

</property>

</bean>（另一分散配置引入分散文件的方式）

</beans>

ApplicationContext 名称=new ClassPathXmlApplicationContext("spring核心配置文件（默认路径是src）"); 获得applicationContext对象（容器对象），从类路径中加载spring配置文件并实例化配置的所有类对象

ApplicationContext 名称=new FileSystemXmlApplicationContext("spring核心配置文件（绝对路径即硬盘中的路径）"); 获得applicationContext对象（容器对象），从文件系统中加载spring配置文件并实例化配置的所有类对象

ApplicationContext 名称=new XmlWebApplicationContext("spring核心配置文件"); 获得applicationContext对象（容器对象），从web系统中加载spring配置文件并实例化配置的所有类对象

类名 对象名=(类名) ApplicationContext.getBean("spring配置文件中的对应类的实例化对象名"); 通过spring的applicationContext对象获得对应类的实例对象，创建实例化类对象的任务由spring完成和管理

ioc(反向控制 inverse of control)：控制权反转，创建对象（bean）和维护对象的关系的权利从程序中转移到spring的容器（applicationContext.xml）

di( dependency injection 依赖注入)：di与ioc是同一个概念，spring设计者认为di更准确表示spring的核心

BeanFactory 工厂对象名=new XmlBeanFactory(new ClassPathResource("spring配置文件名")); 加载spring配置文件不实例化配置类对象，获得BeanFactory对象（工厂对象）

工厂对象名.getBean("spring配置文件中的类名"); 实例化spring配置文件中指定类名的对象

Bean的作用域

Singleton：单例，一个Spring容器中指定bean的所有对象是同一个，加载配置文件时对象会被实例化

Prototype：bean定义对应多个对象实例，加载配置文件时对象不会被实例化

Request：指定bean对象为单例对象，作用域是http的一次请求

Session：指定bean对象为单例对象，作用域是http的会话Session

global-session：指定bean对象为单例对象，作用域是web的Spring ApplicationContext

Spring配置文件配置的实现BeanPostProcessor接口的类的对应bean：

<bean id="实例化对象名" class="实现BeanPostProcessor接口的类（包含包名）" />（实现BeanPostProcessor接口的类对所有Spring配置文件配置的bean对应的类在实例化时进行后置处理）

Spring配置文件配置的bean使用应用环境（ClassPathXmlApplicationContext()函数）调用的生命周期（从开始到结束的顺序）：

1.bean实例化函数（对应类的构造函数）

2.根据配置文件设置bean对应类实例化对象的属性值

3.调用BeanNameAware接口的setBeanName()函数：Spring配置文件配置的bean对应的类实现BeanNameAware接口，调用setBeanName(String 名称)函数通过名称获得类在加载配置文件实例化对象的名称

4.调用BeanFactoryAware接口的setBeanFactory ()函数：Spring配置文件配置的bean对应的类实现BeanFactoryAware接口，调用setBeanFactory (BeanFactory 名称)函数通过名称获得工厂信息

5.调用ApplicationContextAware接口的setApplicationContext()函数：Spring配置文件配置的bean对应的类实现ApplicationContextAware接口，调用setApplicationContext(ApplicationContext 名称)函数通过名称获得ApplicationContext信息

aop（面向切面编程，对所有对象编程）

6.调用BeanPostProcessor接口的postProcessorBeforeInitialization()函数：Spring配置文件配置的bean对应的类实现BeanPostProcessor接口，调用postProcessorBeforeInitialization()函数对Spring配置文件配置的bean对应的类实例化时做前置处理，调用postProcessorAfterInitialization()函数对Spring配置文件配置的bean对应的类实例化时做后置处理

7.调用InitializingBean接口的afterrPropertiesSet()函数：Spring配置文件配置的bean对应的类实现InitializingBean接口，调用afterrPropertiesSet()函数对Spring配置文件配置的bean对应的类实例化的对象做处理

8.调用配置文件中bean标签中配置的初始化函数

9.调用BeanPostProcessor接口的postProcessorAfterInitialization ()函数

10.调用Bean对应类的成员函数

11.Spring容器框架关闭

12.调用DisposableBean接口的destroy()函数：Spring配置文件配置的bean对应的类实现DisposableBean接口，调用destroy()函数对Spring配置文件配置的bean对应的类实例化的对象销毁时做相应处理

13.调用配置文件中bean标签中配置的销毁函数

Spring配置文件配置的bean使用工厂方法（XmlBeanFactory ()函数）调用的生命周期（从开始到结束的顺序）：1,2,3,4 ,7,8,10,11,12,13

Spring配置文件bean对应类的方法可以通过注解方式进行配置对应函数上一行为（@对应注解函数名称）表示对应函数是注解函数

一般在配置文件中进行配置不使用接口

Spring配置bean对应类的（list或者数组）/set/map/properties集合类型的属性的属相值：

<property name="类的属性名">

<list/set/map/props>

[<value>属性值</value>]/[<ref bean="引用的类的spring实例化对象名"></ref>（类中有类的list/set关联关系，对象名类的对应bean在配置文件中需要配置）]/[<entry key="键值" value-ref="属性值"></entry>]/[<prop key=""></prop>]

</list/set/map/props>

</property>

Map集合的取出值方法：

for (Entry<String, 值类型> 名称 :对象的map集合.entrySet()) {

名称.getValue(); 获得值

}

迭代器方法

Map<String, 值类型> 名称=对象的map集合

Iterator iterator=名称.keySet().iterator();

while (iterator.hasNext()) {

String key = (String) iterator.next();

类型名 对象名=名称.get(key);

}

Properties集合的取出值方法：

Enumeration 枚举名=Properties对象名.keys();

while (枚举名.hasMoreElements()) {

String key = (String) 枚举名.nextElement();

类型名 对象名= Properties对象名.get Property(key);

}

内部bean：

<bean id="实例化对象名" class="管理的类名（包含包名）">

<property name="实例化对象名称对应的类在实例化对象名对应的类中的属性名">

<bean id="实例化对象名称" class="管理的类名（包含包名）"></bean>

</property>

</bean>

Spring配置bean对应类的集合类型的属性的属相值设置为空（null）：

<property name="类的属性名">

<null/>

</property>

Spring配置bean对应类的属性的属相值通过类的构造函数进行设置：

<bean>

[<constructor-arg index="属性序列号（从0开始）" type="属性类型" value="属性值"></constructor-arg>]

</bean>

aop（aspect oriented programming）面向切面（方面）编程

汇编语言面向机器

C语言面向过程，一般系统软件的语言（操作系统，数据库，杀毒软件，防火墙，驱动）

Java语言面向对象

Spring面向方面（切面）

把实现的接口的引入关联的源码文件后生成的相应的需要实现的函数的参数名称会相对标准容易理解

Spring配置文件中进行aop配置

<bean id="实例化通知的对象名" class="通知的类名（包含包名）"></bean>（配置通知即实现通知接口的类，通知接口有前置/后置/环绕/异常通知MethodBeforeAdvice/AfterReturningAdvice/ MethodInterceptor/ ThrowsAdvice）

<bean id="实例化的目标对象的名" class="目标对象的类名（包含包名）">

<property name="属性名" value="属性值"></property>

</bean>（配置目标对象即实现相应接口的类）

<bean id="实例化切入的通知的对象名" class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">

<property name="advice" ref="需要切入的通知对象名"></property>

<property name="mappedNames">

<list>

[<value>切入通知的函数名（名称支持使用正则表达式表示）</value>]

</list>

</property>

</bean>

<bean id="proxyFactoryBean" class="org.springframework.aop.framework.ProxyFactoryBean">（代理对象，Spring的类）

<property name="proxyInterfaces">（代理的接口集合）

<list>

[<value>目标对象的类实现的接口名（包含包名）</value>]

</list>

</property>

<property name="interceptorNames">（通知集合）

<list>

[<value>通知的对象名/切入的通知的对象名</value>]

</list>

</property>

<property name="target" ref="代理的目标对象的名称"></property>（代理的目标对象）

</bean>

实现多个接口的类，定义对象的类型为接口，所有接口类型之间可以互相转换

Spring配置文件中bean标签中的property标签配置的属性名以及属性值只会根据对应类的set方法进行设置，与对应类中有无配置的属性无关

实现MethodInterceptor环绕通知接口的类需要实现的方法：

public Object invoke(MethodInvocation arg0) throws Throwable {

目标对象方法函数内执行前的通知

Object object=arg0.proceed(); 调用目标对象方法体返回的对象需要返回

目标对象方法函数内执行后的通知

return object; 返回调用目标对象方法体返回的对象

}

ThrowsAdvice异常通知接口不提供方法，实现异常通知接口的类需要添加实现的方法：

public void afterThrowing(Throwable throwable) {}或者

public void afterThrowing(Method method,Object[] objects,Object target,Exception exception) {}

ProxyFactoryBean是BeanFactory中显示创建代理对象的中心类，ProxyFactoryBean会根据实现的接口，代理的对象，植入的通知使用jdk动态代理技术创建新的代理目标对象类型的子类，代理目标对象没有实现接口，ProxyFactoryBean使用CGLIB库生产代理目标对象类型的子类，所以代理目标类以及类中方法是final类型的不能被代理，Spring只支持方法连接点，不提供属性接入点，Spring认为属性拦截破坏了类的封装

Ssh

ssh（struts+spring+hibernate）框架组合一般步骤顺序：  
1.创建web项目引入相应的Spring开发包，创建Spring配置文件applicationContext.xml文件，测试Spring配置是否成功

2.在项目中引入相应的hibernate开发包，hibernate配置文件hibernate.cfg.xml文件、对象映射文件\*.hbm.xml文件以及SessionFactory在Spring配置文件applicationContext.xml中配置

3.在applicationContext.xml文件中配置数据源：

<!-- 配置数据源 -->

<bean id="实例化数据源的名" class="org.apache.commons.dbcp.BasicDataSource"

destroy-method="数据源销毁方法名">

<property name="driverClassName" value="数据库驱动" />

<property name="url" value="数据库连接的url" />

<property name="username" value="用户名" />

<property name="password" value="密码" />

<!-- 连接池启动时的初始值 -->

<property name="initialSize" value="连接池启动时的初始连接数" />

<!-- 连接池的最大值 -->

<property name="maxActive" value="连接池最大连接数（最大并发量一般设置为500，在线人数一般是并发量乘以50）" />

<!-- 最大空闲值.当经过一个高峰时间后，连接池可以慢慢将已经用不到的连接慢慢释放一部分，一直减少到maxIdle为止 -->

<property name="maxIdle" value="最大空闲数" />

<!-- 最小空闲值.当空闲的连接数少于阀值时，连接池就会预申请去一些连接，以免洪峰来时来不及申请 -->

<property name="minIdle" value="最小空闲数" />

</bean>

4. 在applicationContext.xml文件中配置会话工厂SessionFactory：

<bean id="实例化会话工厂名" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="dataSource" ref="实例化的数据源的名" />

<property name="mappingResources">

<list>

[<value>\*.hbm.xml（对象映射文件，包含包名路径）</value>]

</list>

</property>

<property name="hibernateProperties">（hibernate.cfg.xml中对hibernate的配置）

<value>[hibernate.属性=属性值]</value>

</property>

</bean>

5.创建数据域类（domain）和映射文件\*.hbm.xml测试Spring和hibernate的结合配置是否成功

6.配置事务管理器统一管理SessionFactory的事务和二级缓存

<bean id="实例化事务处理名" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="实例化会话工厂名" />

</bean>

<tx:annotation-driven transaction-manager="实例化事务处理名"/>（启用事务注解，在类或者方法名前一行添加@Transactional，Spring会管理类中所有方法的事务或者指定方法的事务）

7.引入struts开发包，创建struts配置文件struts-config.xml并进行相应配置，一般放在/WEB-INF目录下，在web.xml文件中指定struts的配置文件和指定spring的配置文件，tomcat服务器在加载web.xml文件时会根据web.xml文件中的配置实例化struts和spring即tomcat服务器管理struts和spring容器

web.xml文件中指定Spring的配置文件：

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:Spring配置文件名（包含路径，默认web根目录即src目录）</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>（对Spring容器进行实例化）

在web.xml文件中指定了Spring的配置文件，但Spring没有管理struts时，在struts控制器中可以通过WebApplicationContext 名称=WebApplicationContextUtils.getWebApplicationContext(this.getServlet().getServletContext());直接获得Spring容器即Spring的ApplicationContext

8.在spring配置文件applicationContext.xml中配置struts即控制器action的配置：

<bean name="网址访问的资源名称一般是（/资源名）" class="指定控制器类的路径（包含包名）"></bean>

在struts配置文件中加入与spring关联的配置：

<controller>

<set-property property="processorClass" value="org.springframework.web.struts.DelegatingRequestProcessor" />

</controller>

Spring管理struts后可以通过设置spring配置文件applicationContext.xml中bean标签的scope属性解决action是单例的问题

9. 配置过滤器解决中文乱码问题：

在web.xml文件中指定自定义过滤器或者在web.xml文件中指定Spring框架提供的处理中文乱码的过滤器：

<filter>

<filter-name>encoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encoding</filter-name>

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

</filter-mapping>

在applicationContext.xml文件中配置启用注解扫描：

<context:annotation-config></context:annotation-config>

在指定的类属性的前一行添加@Resource注解表示Spring会根据名称匹配方式给指定的类属性赋值初始化

在web.xml文件中配置Spring提供的OpenSessionInViewFilter过滤器的方法解决延时加载

<filter>

<filter-name>OpenSessionInViewFilter</filter-name>

<filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>OpenSessionInViewFilter</filter-name>

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

</filter-mapping>

Sns系统（社会化网络系统）

struts（1.3.8）+spring（2.5.6）+hibernate（3.3.1）编译环境和运行环境一致