1. **简述Java的基本历史**

java起源于SUN公司的一个GREEN的项目，其原先目的是：为家用消费电子产品发送一个信息的分布式代码系统，通过发送信息控制电视机、冰箱等

1. **简单写出Java特点，写出5个以上，越多越好**

简单的、面向对象的、分布式的、安全的、稳定的、与平台无关的、可解释的、多线的、动态的语言。

1. **什么是Java?**

JAVA：一种编程语言

一种开发环境

一种应用环境

一种部署环境

1. **请写出Java的版本分类，以及每种版本的应用方向**

三种版本：

JME：是面向内存有限的移动终端. 为消费性产品设计的；

JSE： 是SUN公司针对桌面开发和低端商务计算方案开发的版本。为笔记本电脑、PC机设计的；

JEE： 是一种利用JAVA 2平台来简化企业解决方案的开发、部署和管理相关的复杂问题的体系结构。 为企业级的、服务器端的高端应用而设计的；

1. **描述Java技术的主要特性**

java虚拟机

垃圾回收

代码安全性

1. **描述 Java虚拟机的功能**

（1）通过 ClassLoader 寻找和装载 class 文件

（2）解释字节码成为指令并执行，提供 class 文件的运行环境

（3）进行运行期间垃圾回收

（4）提供与硬件交互的平台

1. **简述Java代码安全性**

（1）编译时的错误检验

（2）类装载检查，把本机上的类和网络资源类相分离，在调入类的时候进行检查，因而可以限制任何“特洛伊木马“的应用

（3）字节码校验

（4）沙箱机制，限定访问权限

1. **描述Java垃圾回收机制**

Java 编程语言解除了程序员释放内存的责任。它可提供一种系统级线程以跟踪每一次内存的分配情况。在 Java 虚拟机的空闲周期，垃圾收集线程检查并释放那些可被释放的内存。垃圾收集在 Java 技术程序的生命周期中自动进行，它解除了释放内存的要求，这样能够有效避免内存漏洞和内存泄露（内存泄露就是程序运行期间，所占用的内存一直往上涨， 很容易造成系统资源耗尽而降低性能或崩溃 。

垃圾回收有以下特点：

（1）垃圾回收是一个自动的系统行为，程序员不能控制垃圾回收的功能和行为。

（2）有一些跟垃圾回收相关的方法，比如：System.gc( ) ,调用这些方法，仅仅是在通知垃圾回收程序，至于垃圾回收程序运不运行，什么时候运行，都是无法控制的。

（3）程序员可以通过设置对象为 null，来标示某个对象不再被需要了， 这只是表示这个对象可以被回收了，并不是马上被回收。

1. **简述Java代码的运行过程**

（1）加载代码 通过类装载器装载CLASS文件

（2）校验代码 由字节码校验

（3）执行代码 由解释器将字节码转换成机器码

1. **简述如何搭建Java开发环境**

首先下载安装JDK 然后配置环境

（1）配置PATH，

（2）配置CLASSPATH

（3）配置JAVA\_HOME

1. **简述classpath，path，JAVA\_HOME的意义和配置方法**

path 操作系统运行环境的路径

classpath JAVA运行应用程序时所需要的类包的路径

JAVA\_HOME 供需要运行JAVA的程序使用

1. **请描述：一个完整的Java文件的代码分几个部分，每部分都起什么作用，大致的写法**

package 当前类所在位置

import 当前类运行时所需要的包或类

public class 类名 {

属性定义;

方法定义:

构造方法定义;

public static void main(String args []) 例外{ 程序入口

语句;

}

}

//注释

1. **Java代码的注释有几种？分别举例说明**

（1） // 单行注解

（2） /\* \*/ 块注释

（3） /\* \*

\* 文档注释

\*/

1. **什么是Java代码的语句，什么是Java代码的块，举例说明**

语句 是一行以分号终止的代码,例：int a;

块 是以{ }为边界的一些语句的集合 例：public void tt( ){}

1. **什么是标示符？**

标识符：是赋予变量、类、方法的名称。

1. **标示符定义的规则？**
   1. 首字母只能以字母、下划线、$开头,其后可以跟字母’下划线、$和数字配置CLASSPATH
   2. 首字母小写中间用大写字母隔开
   3. 标识符具有一定的含义

1. **什么是关键字？**

关键字就是编程语言与机器语言的编码约定

1. **true、false、null、sizeof、goto、const那些是Java关键字**

true 、false 、null 为JAVA的关键字

1. **Java的基本数据类型有几种？分别是？**

short int long boolean float double char byte

1. **请写出每种基本类型的默认值？取值范围？分别给出一个定义的例子**

默认值         取值范围 示例

字节型 ： 0 -2^7—-2^7-1 byte b=10;

字符型 ： ‘ \u0000′ 0—-2^16-1         char c=’c’ ;

short : 0 -2^15—-2^15-1 short s=10;

int : 0 -2^31—-2^31-1 int i=10;

long : 0 -2^63—-2^63-1     long o=10L;

float : 0.0f -2^31—-2^31-1 float f=10.0F

double : 0.0d -2^63—-2^63-1 double d=10.0;

boolean: false true\false boolean flag=true;

1. **在基本类型中，如果不明确指定，整数型的默认是什么类型？带小数的默认是什么类型？**

整数类型 默认为 int

带小数的默认为 double

1. **如何定义float类型和long型**

float f = 1.2f

long l = 1.2L

1. **什么是变量？**

变量是用来引用一块内存区域或一个值，用标识符来表示，可以通过操作变量来操作变量所对应的内存区域或值块的值。

1. **变量的定义规则？**
   1. 以字母、$、下划线开头，其后可以跟字母、下划线、$和数字；
   2. 首字母小写，中间单词用大写字母隔开；
   3. 名称具有一定含义；

1. **请写出Java编码约定中对下列部分的要求：类、属性、方法、包、文件名、变量、常量、控制结构、语句行、注释**

类： 一个类文件中类名要和文件名相同，类名一定要以大写字母开头，单词之间用大写字母分隔

属性： 属性名首字母小写，中间用大写字母隔开。

方法： 方法名首字母小写，中间用大写字母隔开。

包： 引用包必须写在类文件的开头,有且只能有一个包，全部用小写字母。

控制结构：当语句是控制结构的一部分时，即使是单个语句，也应使用大括号将语句封闭。

语句行：每行只写一个语句，并使用四个缩进的空格，使代码更易读，注意用分号结束。

注释： 用注释来说明那些不明显代码的段落。

常量： 常量名一般用大写字母，单词之间用下划线分隔，一旦赋值不允许修改。

1. **什么是Javadoc?**

按照一定格式生成程序的文档的工具。

1. **什么是引用类型？**

用户定义类型, 它可引用类和数组。

除去基本数据类型的其他类型都是引用数据类型。

1. **什么是按值传递？什么是按引用传递？**

按值传递：就是将该值的副本传过去（基本数据类型+String类型的传递，就是按值传递）

按引用传递：就是将值的内存地址传过去（除基本数据类型+String以外类型的传递，就是引用传递）

1. **如何创建一个新对象？如何使用对象中的属性和方法？**

使用new 关键字来创建一个新的对象；

通过对象的实例用”**.**“(点)来调用对象中的属性和方法；

静态的方法和属性，也可以直接用类名”**.**“（点）来调用；

1. **简述new关键字所做的工作**

new 创建一个对象，并为对象分配一块内存。

1. **简述”=” 和”= =”的功能和区别**

“=” 赋值,

“= =”当比较基本数据类型的时候，判断前后两个值是否相等；

当比较引用数据类型的时候，判断= =前后两个值的内存地址是否相等；

区别：

= ：为赋值表达式

= = ：为逻辑表达式

1. **什么是实例变量？什么是局部变量？什么是类变量？什么是final变量？**

实例变量：     类中定义的变量，即类成员变量，如果没有初始化，会有默认值；

局部变量：     在方法中定义的变量，必须初始化；

类变量：     用static修饰的属性；

final变量：     用final 修饰的变量，

1. **简述上述各种变量的定义规则和使用规则？**

实例变量：    不需要static关键字来声明，需要类的实例（对象）调用（用”**.**“）；

类变量：        用static修饰，可以用类名调用，也可以用类的实例调用；

局部变量：    在方法内任意定义变量即为局部变量；

final变量：    一旦赋值，不可以再修改的变量；

final属性只允许赋值一次，且只能通过构造方法赋值，定义后也就是一个常量；

final修饰的变量，只允许赋值一次

1. **a++和++a的区别？**

a++ ： 先使用，后加1

++a ： 先加1，后使用

1. **请描述instanceof、?:、&、&&各自的功能**

instanceof ：用来判断某个实例变量是否属于某种类的类型。

? **:** 三目运算符：

表达式1?表达式2**:**表达式3

如果表达式1为true，执行表达式2，否则执行表达式3

&： 位运算：按位与 |

&&： 逻辑运算：逻辑与

1. **请描述>>、<<、>>>的功能**

10>>4 ：算术或符号右移位运算符

<< ：算术或符号左移位运算符

>>> ：逻辑或无符号右移位运算符

1. **请描述什么是强制类型转换？什么是自动类型转换？什么是向上造型？并分别举例说明**

强制类型转换：在一个类型前面加( )，来强制转换

long l = 9L;

int i = (int)l;

自动类型转换：

int i = 5;

String str = “”+i;

向上造型：把范围小的造型为范围大的类型：

int i = 2;

long l = i;

1. **请写出完整的if条件控制语句的结构**

if(布尔表达式){

语句

}else{

语句

}

1. **请写出完整的switch语句的结构**

switch(字符){

case 字符: 语句

break;

case 字符: 语句

break;

default:语句

}

1. **请写出完整的for语句的结构**

for(初始语句;条件语句;步长){

}

1. **请写出完整的while语句的结构**

while(boolean语句){

}

1. **请写出完整的do while语句的结构**

do{

}while(boolean语句);

1. **请描述break 和 continue的功能和用法**

break：终止（跳出）最近的循环

continue：跳出本次循环，执行下一次循环

1. **定义一个一维的int数组，先创建它，并初始化它，给它赋值，然后输出其中的一个值**

public class Arr{

public static void main(String args[]){

int a[] = new int[5];

a={1,2,3,4,5};//错误 , 只能在初始化时这样做

a[0]=1;

a[1]=2;

System.out.println(a[0]);

}

}

1. **定义一个一维的A类型数组，直接定义并赋值，然后输出其中的一个值**

public class A{

public static int i;

public static void main(String args[]){

A aa = new A( );

A bb = new A( );

A a[] = {aa,bb};

a[0].i=2;

System.out.println(a[0]);

}

}

1. **把上面的数组改成2维的数组**

public class A{

public static int i;

public static void main(String args[]){

A a[ ][ ] = new A[5][5];

a[0][0].i=2;

System.out.println(a[0][0]);

}

}

1. **举例说明数组拷贝方法的使用：arraycopy方法**

public class A{

public static void main(String args[]){

int a[] = new int[5];

int b[] = new int[5];

System.arraycopy(a[5],0,b[5],0,a.length)

System.out.println(b[0][0]);

}

}

1. **什么是构造和析构方法？功能是？**

构造方法：每个类至少有一个构造方法，类初始化时调用的方法

1．方法名和类名相同

2．无返回值类型

格式：访问权限 类名(参数列表) {}；

1．自己定义构造后，就没有默认的构造方法

2．无构造方法时，默认为空参的构造方法(无参数,方法体为空)  
析构方法：finalize

类销毁时，自动调用方法

当对象在内存中被删除时,自动调用该方法

在此方法中可以写回收对象内部的动态空间的代码

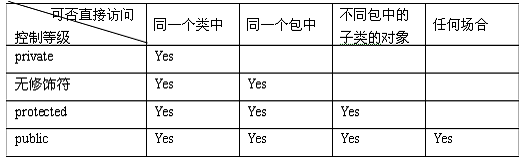
1. **简述Java的访问修饰符类型？分别有什么功能？**

public 公开的，任何一个类都有可以访问

protected 同包同类,只可被它的子类访问

default 只能被同一个包中类访问

private 私有的，只能在本类中被访问



1. **分别说明：在类上、在属性上、在方法上能使用哪些访问修饰符**

在类上：public default final

在方法上： 访问权限：public protected private default

方法修饰符：static

final

返回类型：void

在属性上：public protected private default

static

final

1. **简述什么是包？包的功能**

包：对所定义的多个JAVA类进行分组，将多个功能相关的类定义到一个包(文件)中。

功能：1.易于查找和使用适当的类

2.包不止包含类和接口,还可以包含其它包,形成层次的包空间

3.有助于避免命名冲突

1. **请写出5个以上的JDK中的包，以及他们的基本功能**

java.awt： 包含构成抽象窗口工具集的多个类，用来构建和管理应用程序的图形用户界面

java.lang： 提供java编成语言的程序设计的基础类

java.io：　 包含提供多种输出输入功能的类，

java.net：　 包含执行与网络有关的类，如URL，SCOKET，SEVERSOCKET，

java.applet： 包含java小应用程序的类

java.util：　 包含一些实用性的类

1. **什么是包装类？Java中都有哪些包装类**

在JDK中针对各种基本类型分别定义相应的引用类型 ————–称为封装类

Boolean Byte Short Integer Long Float Double Character

1. **OOP（面向对象）语言的三大特征是？**

封装性，继承性，多态性

1. **分别表述类的三大特性及其他们的功能**

封装：隐藏类的实现细节、迫使用户去使用一个接口去访问数据、使代码更好维护

继承：子类可以直接继承使用父类的方法，程序员只需要做的是定义额外特征或规定将适用的变化

多态性：同一个行为具有多个不同表现形式或形态的能力

1. **如何实现继承？继承的规则？**

public class A extends B{

}

1. 单继承性
2. 构造方法不能继承
3. super引用父类，调用父类的属性，方法
4. 当子类中出现与父类的方法名，返回类型，参数列表相同的方法时要覆盖此方法
5. **简述this和super的功能和用法**

this ：

1. 能访问除构造方法以外所有的属性、方法，通过this**.**　来调用方法和属性
2. 不可以在静态方法中使用，
3. 在调用其它构造方法时，必须放在该构造方法的第一行
4. 在构造方法中使用this(参数列表)，相当于调用本类的其它构造方法，它必须作为构造方法的第一句

super ：访问父类

1. super**.** 点取父类中被子类隐藏的方法和属性，
2. 通过 super(参数列表) 来调用父类的属性和方法，在调用父类的构造方法时必须放在子类构造方法里的第一行；

1. **如何确定在多态的调用中，究竟是调用的那个方法？**

new的是哪一个类就是调用的哪个类的方法。

1. **请描述方法定义的完整的语法结构**

权限修饰符 修饰符 返回类型 方法名(参数) 例外{

方法体;

}

1. **什么是重载？**

在一个类中出现方法名相同，但参数列表不同时，这种情况就是重载。

1. **简述重载的规则**
   1. 方法名称必须相同；
   2. 参数列表必须不同（个数不同，或类型不同，或参数排列顺序不同）；
   3. 与返回类型无关；
2. **什么是方法的覆盖？什么情况下出现？**

方法覆盖：

子类可以修改从父类继承过来的行为,也就是说子类可以创建一个与父类方法有不同功能的方法，但具有相同的：名称、返回类型名和参数列表。

在父子类的情况下出现时。

1. **方法覆盖的规则？**
   1. 方法名要和父类中被覆盖的方法名相同,返回类型相同，参数列表相同。
   2. 访问权限要大于等于被覆盖方法的权限。
   3. 例外列表要小于等于被覆盖方法的例外列表。

1. **static修饰符的功能是？可以用在什么地方？怎么访问？**

static 修饰符功能:

* 1. 共用一块内存区域，也就是用static修饰的变量或成员方法对所有类的实例都是相同的
  2. 静态成员方法和静态变量的优点在于他们能在没有创建类的任何一个实例的情况下被引用。
  3. 可以用在方法或属性上

访问方式：

1. 直接使用类名调用
2. new 一个实例，用实例调用

1. **static的基本规则**
   1. 类中的静态方法只可以调用静态的属性和方法，不允许调用非静态的属性和方法。
   2. 静态变量为类变量，可以用类名直接调用静态的属性和方法。
   3. 静态方法中不允许出现this和super
   4. 构造方法不允许声明static
   5. 非静态变量为实例变量，通过实例调用
   6. 静态方法不能被非静态方法覆盖

1. **final修饰符的功能是？可以用在什么地方？**

功能：final标记所有通用的功能,不能随意更改

可以用在类、属性和方法上

1. **final的基本规则**
   1. final修饰的类不能被继承
   2. final修饰的成员变量只允许赋值一次，且只能通过构造方法里赋值。
   3. final修饰的局部变量即为常量，只能赋值一次。
   4. final修饰的方法不允许被子类覆盖。

1. **什么是抽象方法，什么是抽象类**

用abstract修饰的方法，只有方法的声明，而没有方法的实现，就叫抽象方法。

用abstract修饰的类，就叫抽象类。

1. **抽象类的规则**
   1. 抽象类内部可以没有抽象方法
   2. 必须用abstract修饰
   3. 不能够直接使用抽象类，必须通过子类继承并且实现
   4. 抽象方法不允许被private修饰

1. **什么情况下使用抽象类**
   1. 当一个类的一个或多个方法是抽象方法时；
   2. 当类是一个抽象类的子类，并且不能为任何抽象方法提供任何实现细节或方法体时；
   3. 当一个类实现一个接口，并且不能为任何抽象方法提供实现细节或方法体时；

1. **equals方法和”= =”的功能和区别**

功能：判断对象是否相等

区别：

equals方法比较的是对象的值

= =:比较的是对象值的内存地址，对基本数据类型来说**= =**比较的是数值

1. **toString方法的功能和基本写法**

返回一个String类型

public String toString(){

return “”;

}

1. **String的方法的功能和基本使用方法，请描述5个以上**

substring(参数1,参数2); 功能:取字符串中参数1到参数2的的所有字符; “String”.subString(0,1);

replace(参数1,参数2); 功能:用参数1中的值替换字符串中所有参数2的值 “String”.replace(ing,tt);

equals( ); 功能:判断两个字符串是否相等 “String”.equals(“tt”);

trim( ); 　 功能:去掉字符串两边的空格 “String”.trim();

indexof( ); 功能:查找字符串中参数所在位置,并返回字符串第一个出该参数的下标

“String”.indexOf(“ing”);

split( ); 功能:根据参数分割该字符串 “String”.split(“-”);

1. **为什么使用StringBuffer类？有什么功能和优点？**

–只创建一个对象

–StringBuffer 对象的内容是可以被修改的

–除了字符的长度之外，还有容量的概念

–通过动态改变容量的大小，加速字符管理

1. **举例说明如何使用StringBuffer**

StringBuffer sb = new StringBuffer();

sb.append(“aaa”);

1. **如何使用Java读写系统属性？**

**读：**

Properties props = System.*getProperties*();

Enumeration prop\_names = props.propertyNames();

**while** (prop\_names.hasMoreElements()) {

String prop\_name = (String) prop\_names.nextElement();

String property = props.getProperty(prop\_name);

System.*out*.println(“Property ‘” + prop\_name + “‘ is ‘”+ property + “‘”);

}

**写：**

System.setProperties(props);

1. **简述properties文件的结构和基本用法**

结构：扩展名为properties的文件，内容为key、value的映射，例如”a=2″

用法：

**public class** Test {

**public static void** main(String args[]) {

**try** {

String name = “test.properties”;

InputStream in = **new** BufferedInputStream(**new** FileInputStream(name));

Properties p = **new** Properties();

p.load(in);

System.*out*.println(“a的值==” + p.getProperty(“a”));

} **catch** (Exception err) {

err.printStackTrace();

}

}

}

1. **什么是接口？接口的定义规则？**

接口是抽象方法和常量值的定义的集合。从本质上讲，接口是一种特殊的抽象类，这种抽象类中只包含常量和方法的定义，而没有变量和方法的实现。

public interface Tt {

public void outPut( );

public int tt( );

}

1. **接口的作用？为什么使用接口？**

多重继承

封装、隔离

功能,实现的分离

多态

便于后期维护

接口是可插入性的保证。

1. **什么是多重接口**

就是一个类实现多个接口

1. **描述接口的基本思想？**

封装　隔离

接口及相关机制的最基本作用在于：通过接口可以实现不相关类的相同行为，

而不需考虑这些类之间的层次关系。

    根据接口可以了解对象的交互界面，而不需了解对象所属的类。

面向对象程序设计讲究”提高内聚，降低耦合”。

1. **如何在代码中使用接口？**

public class MyCast implements Tt{

public void outPut( ){}

public int tt( ){

return 0;

}

}

1. **举例说明如何把接口当作类型使用**

public interface Animal(){

public void tt( );

}

public class Dog implements Animal{

public void tt ( ){}

}

Animal ani = new Dog();

1. **如何选择接口和抽象类？为什么？**

优先选用接口，尽量少用抽象类

在下列情况下选择抽象类：需要定义子类的行为，又要为子类提供共性的功能。

1. **什么是异常，异常的分类？**

程序中导致程序中断的一些情况叫做异常，一般程序员可以处理。

异常分为运行时异常和非运行时异常（一般异常）

运行时异常是指因设计或实现方式不当导致的问题。也可以说，是程序员的原因导致的，本来可以避免发生的情况。例如：BufferOverflowException, ClassCastException，IndexOutOfBoundsException，ArithmeticException。出现这些异常的代码不需要try块包围。

非运行时异常是指除运行时异常以外所有的异常，出现这些异常的代码必须用try块包围。例如：FileNotFoundException,EOFException,

1. **简述处理异常的两种方式？**

抛出（throws）和catch语句处理

1. **简述try块的功能和规则**

try块内部一般写一些编程人员认为可能会出现异常的代码,使程序运行时不会因为出现异常而中断。

1. **简述catch块的功能和规则**

功能：可以截获所声明的异常，并在语句块内对其进行处理

规则：catch(Exeption e){

e.printStackTrace( );

     }

1. **简述finally块的功能和规则**

finally块一般写一些不论是否发生异常都必须执行一次的代码

例如关闭与数据库的连接等

1. **简述throw和throws的功能和使用方法**

throw 指编程人员主动抛出一个异常

throw new NullPointerExeption( );

throws指程序遇到异常情况自动的被动抛出一个异常

    public void test( ) throws Exeption{}

1. **异常的分类？**

错误（Error）： JVM 系统内部错误、资源耗尽等严重情况，程序员不可控制

例外（Exception）：其它因编程错误或偶然的外在因素导致的一般性问题，程序可以控制

1. **什么是预定义异常**

java程序中预先定义好的异常叫做预定义异常

1. **简述自定义异常的规则**

写一个类继承Exception

用户自定义异常通常属Exception范畴，依据惯例，应以Exception结尾，应该由人工创建并抛出。

1. **什么是断言？**

用来证明和测试程序的假设。

1. **如何使用断言？举例说明**

一种是assert<<布尔表达式>>；另一种是assert<<布尔表达式>>：<<细节描述>>。

assert a= =10 : “这里a等于10″;

**集合框架**

1. **什么是集合？什么是元素？**

可以包含其他对象的简单对象就叫集合

集合框架中所包含的对象就叫做元素

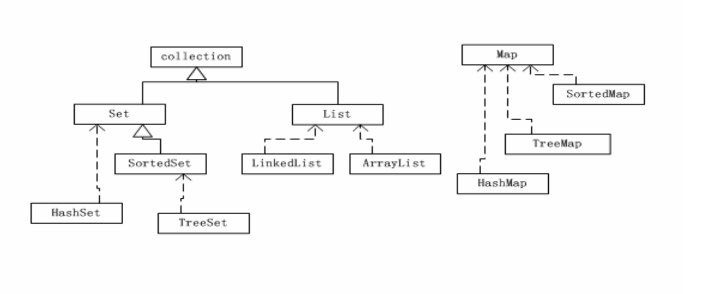
1. **描述出Java集合框架中集合的接口关系**

Collection—-Set和List

Set—-HashSet和SortedSet

List—-ArrayList和LinkedList

Map—-HashMap、SortedMap和TreeMap



1. **代码示例：Collection接口的使用，包括如何定义、初始化、赋值、取值、修改值除值**

Collection col = new ArrayList();

String oldValue = “abcd”;

String newValue = “1234″;

//增加

col.add(oldValue);

Iterator it = col.iterator();

while(it.hasNext()){

//取值

String str = (String)it.next();

if(str.equals(oldValue)){

//删除

col.remove(oldValue);

//修改

col.add(newValue);

}

}

1. **代码示例：List接口的使用，包括如何定义、初始化、赋值、取值、修改值、删值**

List col = new ArrayList();

String oldValue = “abcd”;

String newValue = “1234″;

col.add(oldValue);

Iterator it = col.iterator();

while(it.hasNext()){

String str = (String)it.next();

if(str.equals(oldValue)){

col.remove(oldValue);

col.add(newValue);

}

}

1. **代码示例：Set接口的使用，包括如何定义、初始化、赋值、取值、修改值、删值**

Set col = new HashSet();

String oldValue = “abcd”;

String newValue = “1234″;

col.add(oldValue);

Iterator it = col.iterator();

while(it.hasNext( )){

String str = (String)it.next();

if(str.equals(oldValue)){

col.remove(oldValue);

col.add(newValue);

}

}

1. **代码示例：Map接口的使用，包括如何定义、初始化、赋值、取值、修改值、删**

Map map = new HashMap();

String oldValue = “abcd”;

String newValue = “1234″;

//增加

map.put(“1″,oldValue);

Set set = map.keySet( );

Iterator it = set.iterator( );

while(it.hasNext( )){

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

//取值

String value = map.get(key);

}

//修改

map.put(“1″,newValue);

//删除

map.remove(“1″);

}

1. **描述List接口、Set接口和Map接口的特点**

List接口中的对象按一定顺序排列，允许重复

Set接口中的对象没有顺序，但是不允许重复

Map接口中的对象是key、value的映射关系，key不允许重复

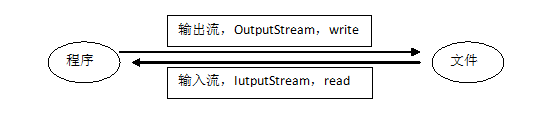
1. **如何给集合排序？**

实现comparable接口

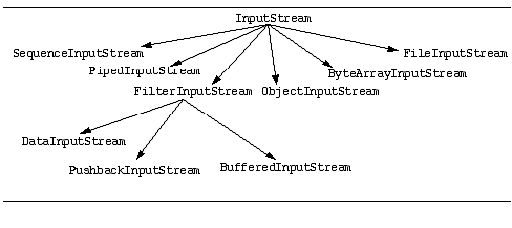
**IO**

1. **什么是流？可画图说明**

流是要传输的数据在源和目的地之间运行的轨迹。



1. **描述I/O流的基本接口和类的结构**



1. **代码示例：使用URL流，读取网页代码**

import java.io.\*;  
import java.net.URL;  
import java.net.URLConnection;  
public class ReadURL {  
public static void main(String[] args) throws Exception {  
URL url = new URL(“http://bbs.tarena.com.cn”); //建立封装统一资源定位符的类  
URLConnection uc = url.openConnection(); //打开连接  
InputStreamReader isr = new InputStreamReader(uc.getInputStream());  
BufferedReader br = new BufferedReader(isr);

//传输数据  
while(true){  
String temp = br.readLine();  
if(temp==null){  
break;  
}  
System.out.println(temp);  
}  
br.close();  
}  
}

1. **什么是Unicode？**

是一种字符的编码方式，占16位字节，Java 技术使用 Unicode 来表示字符串和字符。

1. **代码示例：如何使用Reader和Writer来进行输入输出**

**//Writer写入操作**

        FileOutputStream fos=new FileOutputStream(“a.txt”);

        OutputStreamWriter osw=new OutputStreamWriter(fos);

        BufferedWriter bw=new BufferedWriter(osw);

        bw.write(“欢迎来到java快车”);

        bw.close();

**//Reader读取操作**

        FileInputStream fis=new FileInputStream(“a.txt”);

        InputStreamReader isr=new InputStreamReader(fis);

        BufferedReader br=new BufferedReader(isr);

        String str=br.readLine();

        System.out.println(str);

1. **什么叫保持？什么是可序列化？如何实现可序列化？**

将一个对象存放在某种类型的永久存储器上（比如硬盘，光盘，软盘等），称为保持。

标识一个类可以被持久化，叫序列化，也叫串行化或持久化。表示这个类中的数据可以按流式输出。

实现可序列化的方法：实现java.io.Serializable接口。

1. **代码示例：如何读写对象流**

//读

try {

String str = “123″;

FileOutputStream f = new FileOutputStream(“test.txt”);

ObjectOutputStream s = new ObjectOutputStream(f);

s.writeObject(str);

f.close( );

}catch(Exception e) {

e.printStackTrace( );

}

//写

try {

FileInputStream f = new    FileInputStream(“test.txt”);

ObjectInputStream s = new    ObjectInputStream(f);

str =(String)s.readObject( );

f.close( );

}catch(Exception e){

e.printStackTrace( );

}

1. **简述File类的基本功能**

处理文件和获取文件信息；文件或文件夹的管理。

概括一句话就是：除了读写文件内容其他的都可以做。

1. **代码示例：如何使用随机文件读写类来读写文件内容**

RW表示文件时可读写的

读：

try{

RandomAccessFile f = new RandomAccessFile(“test.txt”, “rw”);

long len = 0L;

long allLen = f.length();

int i = 0;

while (len < allLen) {

String s = f.readLine();

if (i > 0) {

col.add(s);

}

i++;

//游标

len = f.getFilePointer();

}

}catch(Exception err){

err.printStackTrace();

}

写：

try{

RandomAccessFile f = new RandomAccessFile(“test.txt”, “rw”);

StringBuffer buffer = new StringBuffer(“\n”);

Iterator it = col.iterator();

while (it.hasNext()) {

buffer.append(it.next() + “\n”);

}

f.writeUTF(buffer.toString());

}catch(Exception err){

err.printStackTrace();

}

1. **代码示例：如何使用流的基本接口来读写文件内容**

try{

DataInputStream in =

new DataInputStream(

new BufferedInputStream(

new FileInputStream(“Test.java”)

)

);

String currentLine=null;

while ((currentLine = in.readLine()) != null){

System.out.println(currentLine);

}

}catch (IOException e){

System.err.println(“Error: ” + e);

}

**多线程**

1. **什么是线程？**

轻量级的进程

1. **线程的三个部分是？**

处理机

代码

数据

1. **为什么使用多线程**

使UI响应更快

利用多处理器系统

简化建模

1. **代码示例：Java中实现多线程的两种方式，包括如何定义多线程，如何使用多线程**

**实现Runnable接口**

    class Thread1 implements Runnable{

        public void run( ){

            //run里一般写一个while(true)循环

            System.out.println(“Runnable“);

        }

    }

public class Test{

        public static void main(String[] a){

            Thread1 r = new Thread1();

            Thread t1 = new Thread(r);

            Thread t2 = new Thread(r);

            t1.start();

            t2.start();

        }

    }

**继承Thread**

    class Thread2 extends Thread{

        public void run(){

            System.out.println(“extends“);

        }

    }

    public class Test{

        public static void main(String[] a){

            Thread t = new Thread2();

            t.start();

        }

        }

1. **如何实现线程的调度？如何暂停一个线程的运行**

调度用wait和notify，

—-

sleep()

1. **什么是线程的优先级**

判断哪个线程先执行的级别

1. **简述sleep方法和wait方法的功能和区别**

sleep是让线程休眠一段时间

wait是让线程挂起

1. **什么是守候线程**

隐藏在后台持续运行的线程

1. **什么是临界资源**

指多个线程共享的资源

1. **什么是互斥锁，Java中如何实现**

用来保证在任一时刻只能有一个线程来访问临界资源的那个标记

在Java中使用

用在对象前面限制一段代码的执行

用在方法声明中，表示整个方法为同步方法。

1. **什么是死锁？如何避免？**

如果程序中有多个线程竞争多个资源，就可能会产生死锁。当一个线程等待由另一个线程持有的锁，而后者正在等待已被第一个线程持有的锁时，就会发生死锁。

要避免死锁，应该确保在获取多个锁时，在所有的线程中都以相同的顺序获取锁。尽量少用临界资源。

1. **简述wait和notify,notifyAll的使用**

被锁定的对象可以调用wait( )方法，这将导致当前线程被阻塞并放弃该对象的互斥锁，即解除了wait( )方法的当前对象的锁定状态，其他的线程就有机会访问该对象。

notify使等待队列上的一个线程离开阻塞状态

notifyAll使等待队列上的所有线程离开阻塞状态

**Socket编程**

1. **什么是url？基本的格式是？**

统一资源定位器

Http://www.163.com:port

1. **简述IP,Port,TCP的基本功能**

IP代表网络位置

Port代表端口号

TCP可保证不同厂家生产的计算机能在共同网络环境下运行，解决异构网通信问题，是目前网络通信的基本协议

1. **简述Java网络模型的基本功能**

描述服务端和客户端的连接过程

1. **简述Java网络编程究竟做些什么？如何做？**

1.建立连接

2.准备输出的数据，流式输出

3.流式输入，编程业务需要的格式

4.关闭连接

服务器分配一个端口号。如果客户请求一个连接，服务器使用accept( )方法打开socket连接。

客户在host的port端口建立连接。

服务器和客户使用InputStream和OutputStream进行通信。

1. **代码示例：基于Socket编程**

try {

ServerSocket s = new ServerSocket(8888);

while (true) {

Socket s1 = s.accept();

OutputStream os = s1.getOutputStream();

DataOutputStream dos = new DataOutputStream(os);

dos.writeUTF(“Hello,” +s1.getInetAddress() + “port#” + s1.getPort() + “\nbye!”);

            dos.close();

s1.close();

}

}catch (IOException e) {

System.out.println(“程序运行出错:” + e);

}

1. **TCP和UDP区别**

TCP（传输控制协议）和UDP（用户数据报协议）是传输层协议

TCP是面向连接的可靠数据传输协议，能保证传输内容的完整和准确；数据传输的辅助开销大，效率低

UDP: 是无连接的，不可靠地传输协议，不能保证传输内容的完整和准确，对信息的适时要求比较高

**设计模式**

1. **什么是设计模式？**

就是经过实践验证的用来解决特定环境下特定问题的解决方案。

1. **设计模式用来干什么？**

寻找合适的对象

决定对象的粒度

指定对象的接口

描述对象的实现

运用复用机制

重复使用经过实践验证的正确的，用来解决某一类问题的解决方案来达到减少工作量、提高正确率等目的

1. **什么是对象粒度**

对象中方法的大小就是粒度

1. **基本的Java编程设计应遵循的规则？**

面向接口编程，优先使用对象组合

1. **设计模式的应用范围**

所能解决的特定的一类问题中

1. **简述什么是单例模式，以及解决的问题，应用的环境，解决的方案，应用时应注意什么？**

**解决的问题：**在任何时间内只有一个类实例存在的模式。

**应用环境：**需要有一个从中进行全局访问和维护某种类型数据的区域的环境下使用单例模式。

**解决方案：**保证一个类只有一个类实例存在。

**本质：**控制实例的数量。

**注意：**要合理的使用单例，避免单例成为瓶颈。

1. **代码示例：单例模式的两种实现方法，并说明优缺点**

public class Test{

    private static Test test = new Test();

     public Test(){

        }

     public static Test getInstance(){

         return test;

    }

    }

public class Test{

     private static Test test = null;

        private Test(){

        }

        public static Test getInstance(){

            if(test==null){

                test = new Test();

            }

        return test;

        }

    }

第一种方式：装载类的时候就创建类的实例，用空间换时间。

第二种方式：延迟装载，不需每次都创建一个类实例，而只是在第一次使用时创建，有效的利用资源，用时间换资源。

1. **简述什么是工厂模式，以及他解决的问题，应用的环境，解决的方案，模式的本质**

利用工厂来解决接口选择的问题的模式。

**应用环境：**当一个类无法预料要创建哪种类的对象或是一个类需要由子类来指定，创建的对象时，就需要用到工厂模式。

**解决方案：**定义一个创建对象的类，在类的方法中决定具体实例化哪一个类。

**本质：**就是根据不同的情况来选择不同的实现类，重在选择。

1. **代码示例：工厂模式的实现方法**

**单独是使用工厂：**

public class Factory{

     public static Sample creator(int which){

         if (which==1){

             return new SampleA();

        }else if (which==2)

             return new SampleB();

         }

        }

     }

**工厂+单例：**

public class MyFactory{

public static MyFactory f = null;

public MyFactory(){

}

public static MyFactory getInstance(){

if(f==null){

f=new MyFactory();

}

}

public DBDAO getDAO(){

return new DBDAOImpl();

}

}

1. **简述什么是值对象模式，以及他解决的问题，应用的环境，解决的方案，模式的本质**

用来把一组数据封装成一个对象的模式

**解决问题：**在远程方法的调用次数增加的时候，相关的应用程序性能将会有很大的下降

**解决方案：**使用值对象的时候，可以通过仅仅一次方法调用来取得整个对象，而不是使用多次方法调用以得到对象中每个域的数值

**本质：**封装数据。

1. **代码示例：值对象模式的实现方法**

**步骤：**

public class UserModel{

         private String userId;

         private String userName;

            public void setUserId(String id){

             this.userId = id;

         }

         public String getUserId(){

             return userId;

         }

         public void setUserName(String name){

             this.userName = name;

        }

        public String getUserName(){

             return userName;

         }

     }

1. **简述什么是DAO模式，以及他解决的问题，应用的环境，解决的方案，模式的本质**

数据访问对象

**解决问题：**根据数据源不同，数据访问也不同。根据存储的类型（关系数据库、面向对象数据库、纯文件等）和供应商实现不同，持久性存储（如数据库）的访问差别也很大。如何对存储层以外的模块屏蔽这些复杂性，以提供统一的调用存储实现。程序的分布式问题

**解决方案：**将数据访问逻辑抽象为特殊的资源，也就是说将系统资源的接口从其底层访问机制中隔离出来；通过将数据访问的调用打包，数据访问对象可以促进对于不同数据库类型和模式的数据访问。

**本质：**分层，是系统组件和数据源中间的适配器。（一层屏蔽一种变化）

1. **代码示例：DAO模式的实现方法**

1. **什么是框架？框架的优点？**

框架是能够完成一定功能的半成品

１.优点：完成一定的功能、提供一个精良的应用程序架构

２.框架都是以接口的形式出现

３.应该了解框架完成的功能、框架的API接口、框架的功能是怎么实现的

1. **什么是开放-封闭法则(OCP)**

可扩展但是不可以更改已有的模块

对扩展是开放的，对修改是封闭

1. **什么是替换法则(LSP)**

使用指向基类（超类）的引用的函数，必须能够在不知道具体派生类（子类）对象类型的情况下使用。

1. **如何综合使用我们学过的设计模式来构建合理的应用程序结构**

是采用接口进行隔离，然后同时暴露值对象和工厂类，如果是需要数据存储的功能，又会通过DAO 模式去与数据存储层交互。

1. **构建常用的合理的Java应用包结构**

Ui(表现层)

business–factory,ebi,ebo

dao–factory,dao,impl

**awt、swing**

1. **什么是GUI中的容器？什么是GUI中的组件？**

容器：能包含其他container和其他component的component。

组件：在设计上能实现一定功能的封装体；

1. **简述AWT中的Frame、Panel和组件的关系**

组件放在Panel上，而多个Panel可以叠加到Frame上，Frame一般只有一个，各个Panel可以互相在Frame上切换

1. **简述如何使用一个组件**

初始化，加到容器中，注冊事件，实现相应的事件

Panel pan ＝ new Panel();

TextField td = new TextField();

td.setText(“this is in a TextField.”);

pan.add(td);

1. **描述一个GUI界面的基本构成？**

Frame，Panel，組件

Popmenu

1. **如何控制外观，如颜色、字体等？**

可使用setColor()，setFont()方法

例如：

Frame f = new Frame();

Font font = new Font(“TimesRoman”, Font.PLAIN, 14);

f.setColor(Color.red);

f.setFont(f);

1. **什么是布局管理器？**

用来管理GUI界面中组件的分布情況，负责决定布局方针以及其容器的每一个子组件的大小

1. **描述每种布局管理器的基本特点**

FlowLayout 从左到右分布，排满推到下一行

BorderLayout 上下左右中分布

CardLayout 卡片式分布

GridLayout 网格式分布

XYLayout 坐标分布

1. **什么是JFC(Java基础类)？**

是关于GUI组件和服务完整集合，

主要包括5个API：

AWT，JavaD,Accessibility,Drag&Drop,Swing

提供了帮助开发人员设计复杂应用程序的一整套应用程序开发包；

1. **Swing和AWT的区别？**

Swing提供了更完整的组件，引入了许多新的特性和能力。Swing API是围绕着实现AWT 各个部分的API 构筑的。

AWT 采用了与特定平台相关的实现，而绝大多数Swing 组件却不是这样做的，因此Swing 的外观和感觉是可客户化和可插的。

1. **什么是双缓冲？**

在后台进行界面的更新，然后在前台进行界面交换

功能：双缓冲可以改善一个被频繁改变的组件的外观

1. **描述GUI的事件机制**

事件源：是一个事件的产生者，或产生事件的组件对象

事件监听器：调用事件处理方法的对象

事件处理器：就是一个接收事件、解释事件并处理用户交互的方法。

1. **描述Swing应用程序的基本结构？**

组件定义

初始化界面

各种事件处理方法

各种适配类

1. **描述表现层的基本功能？**

展示数据

人机交互

收集参数、调用逻辑层api

1. **描述在开发Swing应用程序中，一般都要写那些代码？都写到什么地方？**

一般要在类的初始化的时候给组件赋值，写在jinit的方法里面

一般要在按扭的事件处理中写收集参数，组织参数，调用业务接口的方法

1. **对于GUI组件而言，最主要的方法是哪些？**

初始化

如何给组件初始化

如何从组件中取值

设计组件的属性

1. **如何学习GUI组件的用法？**

主要学会组件的定义、取值、赋值的方法

类比学习

**JavaScript**

1. **什么是Javascript**

JavaScript是一种基于对象(Object)和事件驱动(Event Driven)并具有安全性能的脚本语言。

1. **Java和Javascript的区别**

1、基于对象和面向对象

Java是一种真正的面向对象的语言，即使是开发简单的程序，必须设计对象。

JavaScript是种脚本语言，它可以用来制作与网络无关的，与用户交互作用的复杂软件。它是一种基于对象（Object Based）和事件驱动（Event Driver）的编程语言。因而它本身提供了非常丰富的内部对象供设计人员使用。

2、解释和编译

两种语言在其浏览器中所执行的方式不一样。Java的源代码在传递到客户端执行之前，必须经过编译，因而客户端上必须具有相应平台    上的仿真器或解释器，它可以通过编译器或解释器实现独立于某个特定的平台编译代码的束缚。

JavaScript是一种解释性编程语言，其源代码在发往客户端执行之前不需经过编译，而是将文本格式的字符代码发送给客户编由浏览器    解释执行。

3、强变量和弱变量

两种语言所采取的变量是不一样的。

Java采用强类型变量检查，即所有变量在编译之前必须作声明。

JavaScript中变量声明，采用其弱类型。即变量在使用前不需作声明，而是解释器在运行时检查其数据类型，

4、代码格式不一样

Java是一种与HTML无关的格式，必须通过像HTML中引用外媒体那么进行装载，其代码以字节代码的形式保存在独立的文档中。

JavaScript的代码是一种文本字符格式，可以直接嵌入HTML文档中，并且可动态装载。编写HTML文档就像编辑文本文件一样方便。

5、嵌入方式不一样

在HTML文档中，两种编程语言的标识不同，JavaScript使用<Script>…</Script>来标识，而Java使用<applet>…</applet>来标识。

6、静态联编和动态联编

Java采用静态联编，即Java的对象引用必须在编译时的进行，以使编译器能够实现强类型检查。

JavaScript采用动态联编，即JavaScript的对象引用在运行时进行检查，如不经编译则就无法实现对象引用的检查。

1. **Javascript的运行环境**

具备javascript运行器的

1. **如何在web页面加入Javascript，请写出两种方式并示例**

<script language=”javascript”>

alert(11);

</script>

或者

<script language=”javascript” src=”/test.js”></script>

1. **写出Javascript基本的数据类型**

整型

实型

布尔

字符型

空值

    特殊字符

1. **Javascript中如何定义变量，有何规则**

必须是一个有效的变量，即变量以字母开头，中间可以出现数字如test1、text2等。除下划线（－）作为连字符外，变量名称不能有空格、（＋）、（－）、（，）或其它符号。

不能使用javascript中的关键字

1. **代码示例：Javascript中的if控制语句的结构**

if(i>4){

alert(11);

}

1. **代码示例：Javascript中的for、while循环语句的结构**

for(var i=0;i<10;i++){

alert(11);

}

while(i<10){

alert(22);

}

1. **简述break和continue的用法和功能**

使用break语句使得循环从For或while中跳出，continue使得跳过循环内剩余的语句而进入下一次循环。

1. **Javascript中如何定义类，如何定义属性，如何定义方法，请代码示例**

function QuaryArgItem(){

this.keys = new Array();

this.values = new Array();

    }

    QuaryArgItem.prototype.push = function(key, value)

    {

     key = (key == null) ? “” : “” + key;

     value = (value == null) ? “” : “” + value;

     this.keys.push(key.toUpperCase());

     this.values.push(value);

    }

    QuaryArgItem是类名

    push相当于方法名

    使用的时候  
：

        a = new QuaryArgItem();

        a.push();

1. **Javascript的function如何定义，有何规则**

Function 方法名 （参数,变元）{

方法体;

Return 表达式;

}

1. **如何触发Javascript的function**

function test(){

alert(11);

}

<input type=”button” onClick=”test();”>

1. **说出下列String对象的方法的功能和用法：toLowerCase、indexOf、subString、toUpperCase**

toLowerCase将指定字符串转化为小写

indexOf判断是否包含某一字符或字符串

substring从字符串中取一段并返回

toUpperCase将指定字符串转化为大写

1. **Javascript的日期对象是？如何初始化日期对象？**

提供一个有关日期和时间的对象Date

date = new Date();

1. **说出下列Javascript系统方法的功能和用法：eval、unEscape、escape、parseFloat**

eval：返回字符串表达式中的值

unEscape：返回字符串ASCI码

escape：返回字符的编码

parseFloat：返回实数

1. **Javascript中如何定义数组？如何初始化？如何取值和赋值**

var arrayName = new Array();

    Function arrayName(size){

        This.length=Size;

        for(var x=; x<=size;x++){

            this[x]=0;

        }

        Reture this;

    }

1. **简要描述Javascript中下列内部对象的功能：Navigator、Window、Location、History、Document**

Navagator：提供有关浏览器的信息

Window：Window对象处于对象层次的最顶层，它提供了处理Navagator窗口的方法和属性

Location：提供了与当前打开的URL一起工作的方法和属性，是一个静态的对象

History：提供了与历史清单有关的信息

Document：包含与文档元素一起工作的对象，它将这些元素封装起来供编程人员使用

1. **如何利用Document来从页面上取值和赋值**

取值：var a = document.all(“text1″).value;

赋值：document.all(“text1″).value = ’123′;

1. **简要描述Javascript对象层次结构**

window–document–组件

1. **说出下列常见事件什么时候被触发：onFocus、onBlur、onSelect、onChange、onClick**

onBlur:当失去输入焦点后产生该事件

onFocus:当输入获得焦点后，产生该文件

onchange:当文字值改变时，产生该事件

onselect:当文字加亮后，产生该事件

onClick：当组件被点击时产生的事件

1. **代码示例：使用Frame作一个基本的三分页面**

<HTML>

<HEAD>

</HEAD>

<Frameset Rows=”10%,90%”>

<frame name=”top” src=”test1.htm”>

<Frameset Cols=”40%,60%”>

<frame name=”left” src=”test2.htm”>

<frame name=”right” src=”test3.htm”>

</Frameset>

</Frameset>

</HTML>

1. **框架如何载入页面**

<frame name=”left” src=”test.htm”>

1. **如何从框架中的一个页面访问另一个页面中的组件**

var value = parent.right.document.all(“text1″);

**CSS**

1. **如何把样式表加入到html页面中**

在文档<HEAD>中用<Style type=”text/css”></style>定义；

使用<LINK>元素链接到外部的样式表单。<LINK REL=”stylesheet” href=”style1.css”>；

1. **如何链接元素和样式，请写出4种方法，并代码示例**

1、直接连接

2、class连接

3、id连接

4、元素的style=””

**Xml**

1. **XML有几种解析方法，有什么区别？**

有XML DOM解析、SAX解析、StAX解析

XML DOM：（XML Document Object Model）处理大型文件时其性能下降的非常厉害。这个问题是由DOM的树结构所造成的，这种结构占用的内存较多，而且DOM必须在解析文件之前把整个文档装入内存,适合对XML的随机访问；

SAX：（Simple API for XML）不同于DOM,SAX是事件驱动型的XML解析方式。它顺序读取XML文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理XML文件，适合对XML的顺序访问；

StAX：（Streaming API for XML）与其他方法的区别就在于应用程序能够把 XML 作为一个事件流来处理，无论从性能还是可用性上都优于其他方法；

DOM解析代码如下：

import java.io.\*;

import java.util.\*;

import javax.xml.parsers.\*;

import javax.xml.transform.\*;

import javax.xml.transform.dom.\*;

import javax.xml.transform.stream.\*;

import org.w3c.dom.\*;

public class Fuxi{

public Fuxi(){}

public void runXml(){

File f = new File(“f:/test/xmltest/student.xml”);

try{

//首先创建一个documentbuilderfoctory的工厂

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

//利用工厂来创建documengbuilder

DocumentBuilder db = dbf.newDocumentBuilder();

//利用db的parse方法来解析xml文件

Document doc = db.parse(f);

//将读出来的文件格式化

doc.normalize();

//定义一个nodelist数组来存放xml文件中的节点（标签）

NodeList students = doc.getElementsByTagName(“student”);

//从xml文件中读取数据

for(int i=0;i<students.getLength();i++){

//定义一个元素

Element student = (Element)students.item(i);

System.out.println(“stu\_id :”+student.getElementsByTagName(“stu\_id”).item(0).getFirstChild().getNodeValue());

System.out.println(“name :”+student.getElementsByTagName(“name”).item(0).getFirstChild().getNodeValue());

System.out.println(“age :”+student.getElementsByTagName(“age”).item(0).getFirstChild().getNodeValue());

}

//向文件中写数据

String stu\_id = “001″;

String name = “xingxing”;

String age = “22″;

Text msg;

//创建元素

Element studentNew = doc.createElement(“student”);

//创建子元素

Element stuid = doc.createElement(“stu\_id”);

//设置元素的值

msg = doc.createTextNode(stu\_id);

//将值添加 给元素

stuid.appendChild(msg);

//将元素添加到节点数组中

studentNew.appendChild(stuid);

Element name1 = doc.createElement(“name”);

msg = doc.createTextNode(name);

name1.appendChild(msg);

studentNew.appendChild(name1);

Element age1 = doc.createElement(“age”);

msg = doc.createTextNode(age);

age1.appendChild(msg);

studentNew.appendChild(age1);

//将元素studentnew添加到document树中

doc.getDocumentElement().appendChild(studentNew);

//进行写操作

TransformerFactory tff = TransformerFactory.newInstance();

Transformer tfor = tff.newTransformer();

DOMSource dos = new DOMSource(doc);

StreamResult result = new StreamResult(f);

tfor.transform(dos,result);

}catch(Exception e){

System.out.println(e.toString());

}

}

public static void main(String[] args){

Fuxi ff = new Fuxi();

ff.runXml();

}

}

**JEE**

1. **简述JEE**

JEE　： JAVA　 PLATFORM　 ENTERPRISE　 EDITION

JEE是一个规范集；

JEE是一个框架集；

JEE是一个技术集或API集；

适用于创建服务器端的大型的软件服务系统

1. **C/S B/S 模式**

C/S client/server即客户机/服务器模式，客户端也参与程序的运行与计算（富客户端、胖客户端）

B/S border/server即浏览器/服务器模式，又称为瘦客户端，客户端不参与程序的运行与计算

1. **企业级应用（大规模的应用）的特点**

1.生命周期长、稳定、可靠

2.组件往往分布在异构的环境中，能够跨平台

3.维护性、扩展性、重用性

4.有事务、安全、线程

1. **什么是业务逻辑？**

依照业务逻辑划分模块，

所谓判定业务　就是具有以下特征：

1.业务流程

2.业务判断

3.功能组合

1. **平台（角色）的划分**

1.平台供应商。提供满足API集实现的厂商（BEA weblogic IBM webshphere）

2.组件供应商

3.组件装配人员

4.部署人员

5.系统管理人员

6.工具供应商 提供开发组件所使用的工具（Jbuilder、eclipse）

1. **java技术分布（设计架构 模块内部设计）**

1.表现层 Servlet、 Jsp、 JavaBean、 Taglib

2.逻辑层 EJB(SessionBean)

3.数据层 （　JDBC 　EJB（EntityBean））

4.服务 JTA（JAVA事务架构） JTS（JAVA事务服务） JAAS（JAVA授权与验证服务）

5.分布式通讯 RMI（IIOP）+JNDI 、JAXP（JAVA 的 XML 架构）

JAVAMAIL（邮件服务）、JMS（消息服务）、IDLC、JCA（JAVA连接器框架 （成））

1. **JEE的体系结构： 是一种组件的体系结构**

1.组件 ： 能够完成一定功能的封状体（独立的功能的集合）

不能单独运行，必须运行在容器上

分为两类：web组件、ejb组件

2.容器 ： 提供组件的运行环境，并对组件进行管理（一段程序）

管理组件的生命周期

不能单独运行，必须运行在服务器上

**程序（组件）通过上下文来调用容器（context）**

**组件通过web.xml向容器描述自己，使容器能够认识组件**

**容器通过回调方法来调用组件**

分为两类：web容器、ejb容器

3.服务器 ： 提供容器的运行环境，提供大量的JEE基础服务

web服务器（只提供web服务）

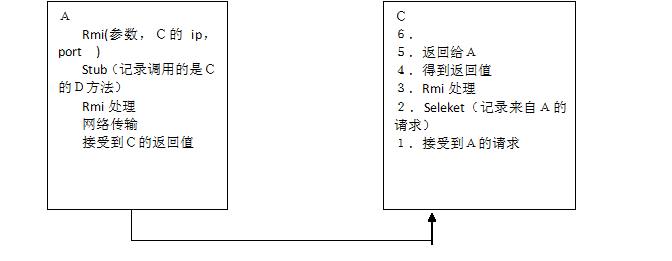
jee服务器（提供web、jee服务）

1. **什么是回调方法**

由容器自动调用的方法，就叫回调方法。

1. **ＲＭＩ远程方法的调用机制**

例如：Ａ主机有Ｂ（）方法，Ｃ主机有Ｄ（）方法，Ａ　想调用Ｃ主机的Ｄ方法



1. **学习方法**

1.规范（熟记） 2.熟悉API 3.多练 4.多想

**Servlet**

* **描述war包、jar包、ear包的结构**

war-                            web文档压缩包

—WEB-INF

—-web.xml            web应用部署文件

—-lib                本web应用需要的jar包（可选）

—-classes            本web应用需要的类文件（可选）

jar-                             java文件压缩包

—META-INF

—-\*.MF java包的启动和描述文件

ear-                            企业资源包/企业应用包

–META-INF

—Application.xml 配置文件

–\*.war

–\*.jar

* **什么是servlet？servlet主要做什么？**

Servlet是运行在服务器端的小应用程序，是接收网络服务的请求和产生响应的一种方式。

servlet的功能：接受http请求，产生动态http响应。

* **servlet 与cgi相比的优点？servlet的缺点**

优点：

性能（线程比进程更快）

可伸缩

Java强健且面向对象

Java平台独立

缺点：

处理代码（业务逻辑）与HTML（表现逻辑）混合在一起

* **常用的servlet包的名称是？**

javax.servlet

javax.servlet.http

* **描述servlet接口的层次结构？**

Servlet

–GenericServlet

–HttpServlet

–自己的servlet

ServletRequest

–HttpServletRequest

ServletResponse

–HttpServletResponse

* **对比get方法和post方法？**

get方法： 请求对服务器没有负面影响，Form数据量小，提交的数据在url中可见；

明文传输，安全度低，不可以传输中文。

post方法：请求的数据过程改变服务器的状态，Form数据量大，提交的数据在url

中不可见，安全度高。可以传输中文；

* **归类描述HttpServletRequest接口都完成那些功能**
  + 读取HTTP头标
  + 读取cookie
  + 读取路径信息
  + 标识HTTP会话。

* **归类描述HttpServletResponse接口都完成那些功能**
  + 设置HTTP头标
  + 设置cookie
  + 设定响应的content类型
  + 输出返回数据

* **描述service方法所完成的基本功能？默认是在那里实现的？**

**基本功能：**service方法是在servlet生命周期中的服务期，根据HTTP请求方法（GET、POST等），将请求分发到doGet、doPost等方法

**实现：**默认在HttpServlet类中实现

* **如何开发自己的Servlet？描述应该做的步骤和每步需要完成的工作**

1.引jar包

1)构建开发环境： 新建工程，把tomcat下lib文件夹下的servlet-api.jar引入到Eclipse的工程中

2.开发servlet类

1)首先继承HttpServlet

2)重写 doGet( ) doPost( )

3.部署

安装web服务器，例如Tomcat

在Tomcat的webapps目录下新建一个文件夹作为web应用的根

    在根下新建一个名为WEB-INF的文件夹，里面建立一个web.xml的文件、一个classes的文件夹、一个lib文件夹

    按照servlet的DTD配置web.xml文件

把编译好的servlet的class文件复制到classes目录下

lib文件存放程序所需要的jar包

4.启动服务器

* **为何servlet需要部署描述？**

servlet需要配置web.xml文件来使容器认识servlet程序（web组件）

* **Servlet基本的描述应该是？请写出来**

<servlet>

<servlet-name>Hello</servlet-name>

<servlet-class>com.javakc.web.action.MyServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Hello</servlet-name>

<url-pattern>/greeting</url-pattern>

</servlet-mapping>

* **如何在html中使用servlet**

FORM标记

ACTION– 指定form信息的目的地（相关的URL）

METHOD – 指定HTTP方法（GET或POST）

语法：

<FORM ACTION=’servlet-URL’ METHOD=’{GET|POST}’>

{HTML form tags and other HTML content}

</FORM>

* **如何接受request中的参数**

String userName = request.getParameter(“userName”)方法

* **如何接受request中header的值**

request.getHeader(name);

request.getIntHeader(name);

request.getDateHeader(name);

request.getHeaderNames();

* **如何输出html**

PrintWriter pw = response.getWriter();

pw.write(“<html>”);

pw.write(“Hello”);

pw.write(“</html>”);

* **如何设置输出的contentType**

response.setContentType(“text/html”)

* **描述servlet的生命周期？**

生命周期是指servlet实例在web容器中从：首次创建调用init方法开始初始化期，经过service方法运行期，一直到destory方法销毁期    结束。

servlet实例的生命周期由web容器来管理。

* **描述init,service,destroy方法的功能和特点**

init方法： 是在servlet实例创建时调用的方法，用于创建或打开任何与servlet相的资源和初始

化servlet的状态，Servlet规范保证调用init方法前不会处理任何请求

    service方法：是servlet真正处理客户端传过来的请求的方法，由web容器调用，

根据HTTP请求方法（GET、POST等），将请求分发到doGet、doPost等方法

destory方法：是在servlet实例被销毁时由web容器调用。Servlet规范确保在destroy方法调用之

前所有请求的处理均完成，需要覆盖destroy方法的情况：释放任何在init方法中

打开的与servlet相关的资源存储servlet的状态

* **什么是回调方法？有什么特点？**

由容器来调用程序的方法

由容器来决定什么时候来调

* **如何设置初始化servlet的参数？**

在<servlet>中添加如下配置

<init-param>

<param-name>greetingText</param-name>

<param-value>Hello</param-value>

</init-param>

* **如何获取servlet初始化的参数**

public void init() {

greetingText = getInitParameter(“greetingText”);

System.out.println(“>> greetingText = ‘” + greetingText + “‘”);

}

* **ServletConfig接口默认实在那里实现的**

GenericServlet类实现ServletConfig接口

* **什么是ServletContext？有什么作用？**

Servlet上下文

ServletContext对象是Web应用的运行时表示，约等于整个web应用；可通过其实现Web应用中的资源共享

* **如何访问ServletContext接口？是在那里实现的？**

在Servlet的实现类中通过this.getServletContext()方法访问。

GenericServlet类实现了ServletContext接口。

* **ServletContext接口的功能包括？分别用代码示例**

只读初始化参数：     getInitParameter(name:String) : String

getInitParameterNames() : Enumeration

读写访问应用级属性：getAttribute(name:String) : Object

setAttribute(name:String, value:Object)

getAttributeNames() : Enumeration

只读访问文件资源：    getResource(path) : URL

getResourceAsStream(path) : InputStream

写web应用日志文件：log(message:String)

log(message:String, Throwable:excp)

* **如何设置ServletContext的参数？**

<context-param>

<param-name>catalogFileName</param-name>

<param-value>/WEB-INF/catalog.txt</param-value>

</context-param>

* **如何获取ServletContext设置的参数值？**

ServletContext context = this.getServletContext();

String catalogFileName = context.getInitParameter(“catalogFileName”);

* **描述Web应用的生命周期？**

Web容器启动时，初始化每个Web应用

Web容器关闭时，销毁每个Web应用

可以创建”监听器”对象触发这些事件；

* **如何用代码实现监控Web应用的生命周期？**

public class Test implements ServletContextListener{

public void contextDestroyed(ServletContextEvent arg0) {

System.out.println(“web 容器销毁了+++++”);

}

public void contextInitialized(ServletContextEvent arg0) {

System.out.println(“web 容器启动了+++++”);

}

}

web.xml文件中的配置：

<listener>

<listener-class>

com.csy.Test

</listener-class>

</listener>

* **web应用中如下错误码示什么意思：400，401，404，500**

400 Bad Request//错误请求

401 Unauthorized//未被认可

404 Not Found//没有创建

500 Internal Server Error//服务器内部错误

* **描述Web应用中用声明方式来进行错误处理的两种方法**

使用error-page元素声明一个给定HTTP状态码的处理器

<error-page>

<error-code>404</error-code>

<location>/error/404.html</location>

</error-page>

可以声明任意数量的错误页面，但一个给定的状态码只能对应一个页面

使用exception-type元素声明给定Java异常的处理器

<error-page>

<exception-type>

java.lang.ArithmeticException

</exception-type>

<location>/error/ExceptionPage</location>

</error-page>

可以声明任意数量的错误页面，但一个给定的异常类型只对应一个页面

不能使用父类捕获多种异常

* **描述记录异常日志的方法，都位于那些接口？**

GenericServlet：

log(message:String)

log(message:String, Throwable:excp)

ServletContext：

log(message:String)

        log(message:String, excp:Throwable)

* **什么是会话？**

Web容器可为每个用户保存一个”会话对象”，用来存储特定用户的会话信息。

* **如何获得会话？**

HttpSession session = request.getSesseion( );

* **会话Api的基本功能？**

getID() :String

    isNew() :boolean

    getAttribute(name):Object

    setAttribute(name,value)

    removeAttribute(name)

* **如何销毁会话？**

1、可使用部署描述符控制所有会话的生命周期

<session-config>

         <session-timeout>10</session-timeout>

     </session-config>

2、可使用控制特定会话对象的生命周期HttpSession接口

invalidate()

     getCreationTime() :long

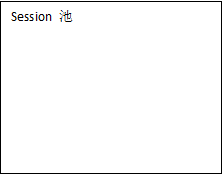
    getLastAccessedTime() :long

    getMaxInactiveInterval() :int

     setMaxInactiveInterval(int)

* **描述会话保持状态的基本原理**

**client server**



**request**

http://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java10.pnghttp://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java11.pnghttp://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java12.pnghttp://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java13.png

**response (sessionid)**

http://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java14.pnghttp://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java15.png

http://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java16.pnghttp://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java17.png

http://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java18.png**request (cookies)**

当客户端第一次请求时，服务器创建一个session与request绑定，用响应对象response来返回sessionid 放到客户端的cookies中存储下来，下次在发送请求时，直接根据sessionid来检索服务器的会话（每次请求都会将所有的cookies带到服务器端）

* **请描述session和cookie的异同之处**

* **如何读写cookie，代码示例**

写：    Cookie c = new Cookie(“yourname”, name);

response.addCookie(c);

读：Cookie[] cookies = request.getCookies();

**for** ( **int** i=0; i < cookies.length; i++ ) {

            String key =cookies[i].getName();

String value = cookies[i].getValue();

System.*out*.println(“cook”+i+“:key=”+key+“,value=”+value);

     }

* **什么是URL重写，如何实现，代码示例**

Cookie不能使用时，可使用URL重写 request.encodeURL();

客户在每个URL的后面追加额外的数据

服务器将此标识符与其存储的有关会话数据相关联

http://host/path/file;jsessionid=123

* **描述web应用的4种认证技术**

BASIC – Web浏览器接收用户名和口令，将其以明码方式发送给Web服务器

DIGEST – Web浏览器接收用户名和口令，使用加密算法将此数据发送给Web服务器

FORM – Web应用提供发送给Web浏览器的HTML form

CLIENT-CERT – Web容器使用SSL验证用户，服务端和客户端的链路保护

* **什么是授权，什么是验证？**

授权是根据用户角色划分web资源的过程，其标识web应用中的安全域 分配权限

web容器使用厂商指定的机制验证用户的角色 匹配权限

* **什么是HTTPS**

HTTPS (Secure Hypertext Transfer Protocol) 是使用SSL协议的HTTP

* **什么是审计？**

也就是访问跟踪，是为web应用的每次访问保留记录的过程

* **如何实现声明性授权**

1、标识web资源集

2、标识角色

3、将web资源集影射到角色

4、标识每个角色中的用户

在 web.xml里配

* **描述servlet并发问题？**

多个同类线程运行，可以共享同一个Servlet实例，共享的数据和资源未合理同步，可能会引起数据的冲突

* **描述Web应用中的六种属性范围**

局部变量（页面范围）

实例变量

类变量

请求属性（请求范围）

会话属性（会话范围）

上下文属性（应用范围）

* **指出上述六种哪些是线程安全的**

局部变量和请求属性

* **什么是STM？如何实现？**

SingleThreadModel接口

可以实现SingleThreadModel接口保证某一时刻只有一个请求执行service方法

* **如何实现并发管理？**

尽可能使用局部和请求属性

使用synchronized语法控制并发

尽可能减少同步块和同步方法的使用

使用正确设置了线程安全的资源类

**Jsp**

1. **什么是Jsp?**

Java Server Page    结合java和html在服务端动态生成web页面的技术

1. **描述Jsp页面的运行过程？**

第一步：

请求进入Web容器，将JSP页面翻译成Servlet代码

第二步：

编译Servlet代码，并将编译过的类文件装入Web容器（JVM）环境

第三步：

Web容器为JSP页面创建一个Servlet类实例，并执行jspInit方法

第四步：

Web容器为该JSP页面调用Servlet实例的\_jspService方法；将结果发送给用户

1. **描述Jsp页面的五类脚本元素的功能、写法、并示例**

**注释** <%– –%>

<HTML>

<%– scripting element –%>

</HTML>

**指令标记** <%@ %> 指令标记影响JSP页面的翻译阶段

<%@ page session=”false” %>

**声明标记** <%! %> 声明标记允许JSP页面开发人员包含类级声明

<%! public static final String DEFAULT\_NAME = “World”; %>

**脚本标记** <% %> 脚本标记允许JSP页面开发人员在\_jspService方法中包含任意的Java代码

    <% int i = 0; %>

**表达式标记** <%= %> 表达式标记封装Java运行时的表达式，其值被送至HTTP响应流 <B>Ten is <%= (2 \* 5) %></B>

1. **描述Jsp页面中的注释种类和写法**

HTML注释

<!– HTML注释显示在响应中 –>

JSP页面注释

<%– JSP注释只在JSP代码中可见，不显示在servlet代码或响应中。–%>

Java注释

<%

/\* Java注释显示在servlet代码中，不显示在响应中

\*/

%>

1. **描述Jsp页面的指令标记的功能、写法、并示例**

指令标记影响JSP页面的翻译阶段

<%@ page session=”false” %>

<%@ include file=”incl/copyright.html” %>

<%@ taglib %>

1. **描述Jsp页面的声明标记的功能、写法、并示例**

声明标记允许JSP页面开发人员包含类级声明

写法：

<%! JavaClassDeclaration %>

例：

<%! public static final String DEFAULT\_NAME = “World”; %>

<%! public String getName(HttpServletRequest request) {

return request.getParameter(“name”);

}

%>

<%! int counter = 0; %>

1. **描述Jsp页面翻译成Servlet的规则**

jsp中的注释标记被翻译成Servlet类中的注释

jsp中的指令标记被翻译成Servlet类中的import语句等

jsp中的声明标记被翻译成Servlet类中的属性

jsp中的脚本标记被转移到Servlet类中service方法中的代码

jsp中的表达式标记被翻译成Serlvet类中的write()或者print()方法括号中的代码

1. **描述Jsp页面的九个预定义变量的功能、用法、并示例**

request 与请求相关的HttpServletRequest对象

response 与送回浏览器的响应相关的HttpServletResponse对象

out 与响应的输出流相关的JspWriter对象

    session 与给定用户请求会话相关的HttpSession对象，该变量只在JSP页面参

与一个HTTP会话时有意义

    application 用于Web应用的ServletContext对象

config 与该JSP页面的servlet相关的ServletConfig对象

pageContext 该对象封装了一个JSP页面请求的环境

page 该变量与Java编程语言中的this变量等价

    exception 由其它JSP页面抛出的Throwable对象，该变量只在”JSP错误页面”中可用

1. **page指令的功能，写法、并示例，并描述它的如下属性的功能和用法：import、session、buffer、errorPage、isErrorPage、ContentType、pageEncoding**

import import 定义了一组servlet类定义必须导入的类和包，值是一个由

逗号分隔的完全类名或包的列表。

session session 定义JSP页面是否参与HTTP会话，值可以为true（缺省）

或false。

buffer buffer 定义用于输出流（JspWriter对象）的缓冲区大小，值可以为

none或Nkb，缺省为8KB或更大。

errorPage 用来指定由另一个jsp页面来处理所有该页面抛出的异常

isErrorPage 定义JSP页面为其它JSP页面errorPage属性的目标，值为true或

false（缺省）。

ContentType 定义输出流的MIME类型，缺省为text/html。

pageEncoding 定义输出流的字符编码，缺省为ISO-8859-1

1. **描述MVC各部分的功能？MVC的优点？MVC的缺点？**

**MVC各部分的功能：**

Model（模型表示企业数据和业务逻辑）

封装应用状态

响应状态查询

暴露应用的功能

Controller（控制器接受用户的输入并调用模型和视图去完成用户的需求）

验证HTTP请求的数据

将用户数据与模型的更新相映射

选择用于响应的视图

View（视图是用户看到并与之交互的界面）

产生HTML响应

请求模型的更新

提供HTML form用于用户请求

**MVC的优点：**

低耦合性：视图层和业务层分离

高重用性和可适用性

较低的生命周期成本

快速的部署

可维护性

有利于软件工程化管理

提高软件的健壮性

**MVC的缺点：**

工作量大，增加工作的复杂性，MVC不适合小型甚至中等规模的应用程序

1. **什么是Model 1结构，以及结构中各部分的功能**

**结构：**jsp+javabean

Model1中使用jsp来处理web应用中的视图控制部分

Javabean收集参数

1. **什么是JavaBean?**

用户可以使用JavaBean将功能、处理、值、数据库访问和其他任何可以用java代码创造的对象进行打包，并且其他的开发者可以通过内部的JSP页面、Servlet、其他JavaBean、applet程序或者应用来使用这些对象。

1. **JavaBean的规则？**

使用get和set方法定义属性

一个无参构造方法

无public实例变量( 所有属性私有化)

1. **什么是jsp标准动作？包含那些？分别都是什么功能？如何使用？**

JSP页面中使用类似于XML的标记表示运行时的动作

jsp:useBean

jsp:setProperty

jsp:getProperty

jsp:param

jsp:include

jsp:forward

1. **用代码示例如下标准动作的使用：useBean、getProperty、setProperty**

<jsp:useBean

id=”myForms”

class=”com.base.mystruts.forms.MyActionForm” scope=”session” />

    <jsp:setProperty name=”myForms” property=”name” />

<jsp:getProperty name=”myForms” property=”id” />

1. **描述说明Bean的四种scope**

page

request

session

application

1. **描述说明页面上的字段和Bean中属性的对应规则**

id 指javabean的变量名

class指javabean类的全路径

scope指javabean的应用范围

name指所用到的javabean的变量名

    property指javabean中的属性

1. **描述useBean动作的处理过程**

使用id声明变量

试图在指定的范围内查找对象

如果没找到

创建一个类的实例

执行useBean标记体初始化对象

如果找到

     将对象转换为类指定的类型

1. **描述forward动作的功能**

使用脚本代码处理请求时，可用jsp:forward 动作产生一个不同的视图，

使用同一个request：

1. **什么是Model 2结构，以及结构中各部分的功能**

jsp+model+servlet

Model 2架构使用MVC模式，JSP页面充当视图，Servlet充当控制器

Servlet控制器：

验证HTML form数据

调用模型中的业务服务

存储请求（或会话）范围内的域对象

选择下一个用户的视图

JSP页面视图：

使用用户界面（在HTML中）

        访问域对象

1. **如何获得分发器？分发器的功能？**

上下文对象中的分发器：

ServletContext context = this.getServletContext();

RequestDispatcher servlet = context.getNamedDispatcher(“MyServlet”);

servlet.forward(request, response);

请求对象中的分发器：

RequestDispatcher view = request.getRequestDispatcher(“tools/nails.jsp”);

view.forward(request, response);

可以将请求转发到另一个jsp页面，并保持请求的传递性

1. **Jsp中页面跳转的两种方式？**

* **使用分发器跳转页面**

RequestDispatcher dispatcher = request.getRequestDispatcher(“url”);

dispatcher.forword(request);

    特点：保持请求的传递（上一次请求中的数据，在下一个请求中仍然可以得到）；

        转发的页面只能是同一个Web应用程序的其他Web组件

* **使用重定向跳转页面**

response.sendRedirect(“url”);

    特点：产生全新的请求，（上一次请求中的数据，在下一个请求中不能得到）；

        重新定向的页面可以是任意的URL

1. **描述Jsp中引入另外页面的两种方法，并比较两种方法的特点？**

<%@ include file=””%> 在编译期间把另一个页面完全嵌入这个页面,可以说是一个页面

在这里可以共享变量

<jsp:include page=””> 动态的再运行期间把另一个页面加入这个页面,可以说是两个

页面，不可以共享变量

1. **什么是自定义标记**

自定义标记库是一个Web组件，

包括：

一个标记库描述符文件（\*.tld）

所有相关的标记处理器类(\*.jar)

1. **描述使用自定义标记的步骤，以及每步详细的工作**

使用自定义标记前，要拿到.tld和.jar两类文件

1. 把.jar文件放到WEB-INF的lib里面。
2. 把.tld文件放到WEB-INF根目录下。
3. 在web.xml中配置。

<taglib>

<taglib-uri>http://java.sun.com/jsp/jstl/core</taglib-uri>

     <taglib-location>/WEB-INF/c.tld</taglib-location>

</taglib>

1. 在页面中引用。

<%@ taglib uri=“http://java.sun.com/jsp/jstl/core” prefix=“s”%>

uri必须与web.xml中的uri对应

prefix是前缀，用来区分是哪一个taglib

使用标记  
格式：<prefix:tag名称 属性>

<s:forEach >

**在tomcat下配置数据源**

**1、服务器与数据库的连接**

配置server.xml文件

**1.oracle**

<Resource name=”jdbc/company” scope=”Shareable” auth=”Container” type=”javax.sql.DataSource”

                        factory=”org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory”

url=”jdbc:oracle:thin:@127.0.0.1:1521:orcl”

driverClassName=”oracle.jdbc.driver.OracleDriver”

username=”aa”

password=”aa”

maxActive=”50″

maxIdle=”10″

maxWait=”-1″

/>

**2.sqlserver**

<Resource name=”jdbc/webpage” scope=”Shareable” auth=”Container” type=”javax.sql.DataSource”

        factory=”org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory”

url=”jdbc:jtds:sqlserver://localhost:1433;SelectMethod=cursor;

DatabaseName=webpagetest”

driverClassName=”net.sourceforge.jtds.jdbc.Driver”

username=”aa”

password=”aa”

maxActive=”50″

maxIdle=”10″

maxWait=”-1″

/>

**2、配置自己的web应用的xml文件**

<Context path=”/eb03web” docBase=”F:/workweb/eb03web”

privileged=”true” antiResourceLocking=”false” antiJARLocking=”false”>

<ResourceLink global=”jdbc/company” name=”jdbc/company” type=”javax.sql.DataSource”/>

</Context>

**3、配置web.xml文件**

与服务器建立连接

<resource-ref>

<res-ref-name>jdbc/company</res-ref-name>

<res-type>javax.sql.DataSource</res-type>

<res-auth>Container</res-auth>

</resource-ref>

**4、在应用程序中不直接与数据库连接，通过DataSource连接池建立连接**

//1.在程序单独用时设置系统的属性，在tomcat下可以不用配置，

System.setProperty(Context.PROVIDER\_URL,””);

System.setProperty(Context.INITIAL\_CONTEXT\_FACTORY,””);

//2.创建一个上下文对象

InitialContext context = new InitialContext();

//3.通过上下文对象在连接池中查找DataSource

DataSource ds = (DataSource)context.lookUp(“java:comp/env/jdbc/company”);

//4.通过数据源建立连接

ds.getConnection();

**webwork**

1. **webwork是什么**

    webwork是一种基于MVC（Model View Controller ）的web应用框架。

    Webwork包括两个部分：webwork2和xwork，webwork2处理web相关的部分，xwork处理web无关的部分。

1. **描述基本的webwork应用的组成部分以及每部分的功能，并指出哪些是需要开发人员作的**

1）    WebWork ActionContext Clear Up ：负责清空ActionContext，确保不会出现内存泄露，有时会引起一些集成上的问题。

    2）    other Filters ：其他的过滤器

    3）    Filter Dispatcher ：webwork的总调度类，通过http请求到ActionMapper询问请求的Action是否执行

    4）    ActionMapper ：决定这个action需要被执行

    5）    ActionProxy：找ConfigurationManager要Action的配置，然后产生Action Invocation对象

    6）    ConfigurationManager：响应ActionProxy的请求找到xwok.xml

    7）    Xwork.xml ：通过配置让 Action 调用器 调用Action

    8）    Action Invocation：调用对应的Action

    9）    Interceptor ：拦截在Action和result执行前后完成一些功能（比如将request中的数据映射到值栈中）

    10）    Action ：处理客户端请求，调用逻辑层API，通过Result选择下一界面

    11）    Result：常量字符串

    12）    页面：通过Taglib的标签 得到值栈中的数据并显示

    13）    Taglib： 得到封装对象中的数据

    需要开发的有：Action，页面（配置上Taglib的前缀，用OGNL得到数据），xwork.xml，如果要解决中文问题还要配置 webwork.properties文件

1. **描述webwork开发的步骤，以及每步具体的工作**

1（导包）、把webwork2.4.4.jar和lib下搜出的所有jar包导到eclipse和web应用的lib下

    2（创建web应用）、在tomcat的webapp下新建一个自己的web应用，把webwork包中blank/webapp下的所有东西拷贝到自己的web应用下

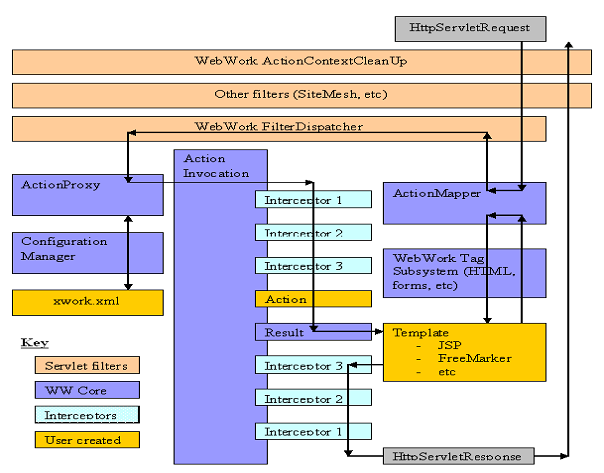
    3（复制Taglib）、在Webwork-2.2.4发行包中找到taglib.tld，在/src/java/META-INF下面，拷贝到自己web应用的WEB-INF下面

    4（编译blank下的源文件）、把blank应用下src放到工程中并编译，把类文件放到WEB-INF的classes下

    5、在地址栏输入http://localhost:8080/自己web应用的名字

1. **描述从页面点击一个按钮开始，完整的Webwork应用运行过程**
   1. 用户点击页面的提交按钮，即客户端发送http请求到web容器。
   2. web容器通过web.xml找到web容器的Filter；通过url-pattern 找到filter-name，再通过filter-name找到filter-class，即找到FilterDispatcher。
   3. Action代理调用ConfigurationManager读取xwork.xml文件，根据提交的xx.action,找到对用的action的全路径，通过Action调用器调用Action类。
   4. 在调用Action类之前，首先被配置的拦截器拦截到，进行相关的处理，如将request中的数据映射到值栈中。
   5. 回调Action中的execute方法，执行后返回Result。
   6. 得到Reslut后，在xwork.xml中匹配下一个页面。
   7. 执行Result调用下一个页面。

1. **图示描述webwork的架构**



1. **为什么推荐使用ActionSupport**

因为ActionSupport不仅实现了Action接口，而且实现了其他的接口，例如：validate()方法-数据检验，getLocal()获取本地信息等功能

1. **Webwork中数据检测的方式有几种，详细写出分别怎么做**

1、在validate()方法中进行数据检验

    2、在execute()方法中进行数据检验，但一定要有返回值为INPUT

1. **Webwork中如何使用本地信息**

1、和Action类同级新建一个与Action同名的.properties文件，并用key，value定义好组件对应的信息（国际化信息）

2、在Action类中使用：this.getText(“key”)

1. **详细写出webwork的Action中三种数据的对应方式，并比较优缺点**

1、域模型：灵活，封装数据， public 或 private+get/set

    2、字段驱动：把字段直接封装在Action中，适用于字段少的程序

    3、模型驱动：Action类实现ModelDriver接口，并实现getModel方法，页面上直接用返回对象的字段就可以，但一个Action只能对应一个Model

1. **什么是Result**

result是一个字符串常量，他表示Action执行完成的状态，xworlk.xml根据返回的result选择下一个界面

1. **写出通用Result中dispatch,redirect和chain的特点**

1、dispatch：保持同一请求，保持数据的传递

    2、redirect：重定向，一个全新的请求，丢失数据

    3、chain；保持数据的传递，但不是同一请求（一个全新的请求）

1. **Interceptor的功能和运行特点**

能在Action和Result执行前后完成一些功能；也可以把Action的共性功能分离出来，以便以后发生变化便于修改，不用动很多的类

1. **配置Interceptor要注意什么，什么是Interceptor栈**

注意：以后自定义的Intercepter要追加到defaultStack后——即自己定义一个栈把默认栈和自定义的Intercepter追加上

    多个Interceptor构成Interceptor栈

1. 什么是OGNL

OGNL是Object-Graph Navigation Language的缩写，它是一种功能强大的表达式语言（Expression Language，简称为EL），通过它简单一致的表达式语法，可以存取对象的任意属性，调用对象的方法，遍历整个对象的结构图，实现字段类型转化等功能。它使用相同的表达式去存取对象的属性。

1. **代码示例：在webwork中向session中取值和赋值**

Action向session中取值：

ServletActionContext.getRequest().getSession().getAttribute(“nowPage”);

赋值：

ServletActionContext.getRequest().getSession().setAttribute(“nowPage”,nowPage);

页面向session中取值： <ww:property value=”#session['pageNum']“/>

1. **代码示例：在webwork中定义集合，并取值和赋值**

<ww:set name=”col” value=”{1,2,3}”/>

    <ww:property value=”#col[1]“/>

1. **简述值栈的原理**

1、值栈与request绑定，而且是线程安全的

    2、值栈相当于一个小型的数据库

    3、webwork所有的值存取都通过值栈

1. **如何在webwork中取得HttpSession的数据**

（1）取得HttpRequest对象：

HttpServletRequest request = ServletActionContext. getRequest();//这个方法返回了与当前线程的ActionContext相关联的request

（2）取得HttpSession对象：

HttpSession session = ServletActionContext. getRequest().getSession();

1. **描述如何使用验证框架的步骤和每步应做的工作**

使用验证框架的方法特别简单：

    （1）写validatiors.xml文件，注册要用到的验证器

    （2）写类名-validation.xml文件，并放到相应的地方

    （3）开启Action的验证功能，在xwork.xml文件中指定拦截器”validation”或者”validationWorkflowStack”，示例：

        <interceptor-ref name=”validationWorkflowStack”/>

    注意：不管验证是否通过，我们的Action都会执行，但如果验证没有通过，它不会调用Action的execute()方法

1. **set标签是用来做什么的，描述set标签支持的作用域**

功能：对值栈中的表达式进行求值，并将结果赋给特定作用域中的某个变量

    下面来看个例子：

    每次都要重复#session['user']不仅是麻烦，而且容易出错，较好的方式是：定义一个临时变量来代替它。

            <ww:set name=”test” value=”#session['user']“>

            <ww:property value=”#test.username”/>

            <ww:property value=”#test.age”/>

            <ww:property value=”#test.address”/>

    作用域的说明： Set标签支持5种作用域：default(也是默认的，ActionContext)，page,request,session,application。通常我们不需要管，除非你要把一个标签或者servlet集成到webwork中。

1. **代码示例：property、set、Iterator、bean、param、if、else标签的用法**

    <ww:property value=”#session['user'].username”/>

<ww:property value=”#session['user'].username”/>

    <ww:property value=”#session['user'].age”/>

<ww:property value=”#session['user'].address”/>

    <ww:set name=”test” value=”#session['user']“>

    <ww:iterator id=”kc” value=”ewm.col”>

        <td><ww:property id=”kc” value=”id”/></td>

    </ww:iterator>

    <ww:bean id=”kc” name=”com.opensymphony.webwork.util.Counter”>

        <ww:param name=”last” value=”100″/>

    </ww:bean>

    <ww:if test=”#s.even”>

        <ww:property />偶数行

    </ww:if>

    <ww:else>

        <ww:property />奇数行

    </ww:else>

**ORM对象关系型数据库映射**

(Object Relation Mapping)

一、映射的特点

1.一个对象可以对应多个表

2.一个表可以对应多个对象

3.对象和表的关系不用完全对应

4.对象的属性的名称和表字段的名称可以不同

5.类型可以不同，但数据类型之间可以转换

6.对象中必须有主键，数据库的表对主键无所谓

7.数据库中有主外键关系时，对象中可以有，也可以没有

二 、有一个映射的描述文件————>xml

三 、怎样实现描述文件————>一段程序

对象 映射（mapping） 数据库

user

id=11

name=”张三”———— > (1) —————> tbl\_user(id,name,age,tel,address)

age=25 <————(2) <————

(1)从对象映射到数据库

1.JDBC

2.根据描述文件来动态拼接sql

3.执行，添加到数据库中

(2)从数据库中映射到对象

1.JDBC

2.拼接查询sql

3.ResultSet

4.Model

**hibernate**

**开发流程：**

1. **是什么？**

hibernate是一种基于orm 的轻量级的框架

1. **有什么？**

1. session 保持连接状态（依赖事务和连接池）

2. Transaction事务

3.Connection Provider 连接池

1. **能干什么？**

1.orm

2.提供操作数据库的接口，简化数据持久化的编程任务

1. **怎么做？**

1.搭建hibernate的运行环境，将hibernate的包和hibernate所需要的包拷贝到lib 文件夹下

2.O（vo）

3.R （关系型数据库中的表）

4.配置文件

**4.1 cfg.xml 放在classes根目录下，默认名字为hibernate.cfg.xml**

a.与数据库的连接

b.可选配置

c.映射资源的注册

**4.2 hbm.xml文件 名字与类名相同 并且与类放在一起**

a. 对象与数据库表之间的映射

b. 对象的属性与数据库表的字段之间的映射

c.组件之间的映射

d.对象与对象之间的关系映射

5、客户端

1.得到SessionFactory

2.通过SessionFactory 工厂来创建Session实例

3.打开事务

4.操作数据库

5.事务提交

6.关闭连接

**运行流程：**

1. **整体流程**

1.通过configuration来读cfg.xml文件

2.得到SessionFactory 工厂（根据数据库的连接来创建sessionfactory）

3.通过SessionFactory 工厂来创建Session实例

4.打开事务

5.通过session的api操作数据库

6.事务提交

7.关闭连接

1. **save**

1.to—>po

2.根据model和cfg.xml中映射文件的注册来找到hbm.xml文件

3.根据hbm.xml文件和model来动态的拼sql

4.客户端提交或者刷新内存

5.执行sql，值放到数据库

1. **update、delete**

1.根据model 的id在内存hibernate的缓存中查找该对象

如果内存中没有就到数据库中查找来保证对象的存在

2.根据model和cfg.xml文件中映射文件的注册来找到hbm.xml文件

3.根据model和hbm.xml 文件来动态拼sql

4.客户端提交或者刷新内存

5.执行sql

1. **Query**

**load**

1.根据model的类型来找到hbm.xml文件

2.判断id对应的对象在缓存中是否存在，如果存在，就直接使用，如果不存在就进行下一步。

3.用id做为查询条件来动态拼sql

4.执行sql 查询（先在内存中查找，如果没有找到（会抛出例外），就去数据库中查找）

5.返回一个model类型的对象

**get**

1.根据model的类型来找到hbm.xml文件

2.用id做为查询条件来动态拼sql

3.执行sql 直接在数据库中查找，如果没有查到就会返回null

**query**

1.分析hql语句，得到model的类型

2.根据moedel和cfg.xml文件中映射文件的注册找到hbm.xml文件

3.根据model和hbm.xml文件来动态拼sql

4.执行sql查询数据库

5.返回一个resultset

6.循环resultset的值，放到model中在放到集合（List）中

**谈谈hibernate缓存机制**

缓存其实很直观的理解为放到内存或者硬盘上的一块区域，缓存就是一个缓冲的作用，有很多时候，我们的数据需要经常访问，如果放到数据库中，不是不可，只是效率上低，如果有了中间产物来代替数据库，就省很多事情了。打个比方，去买酱油，不一定一定要去酱油的生产工厂去买。代码中需要数据的时候，可以先到缓存中查找，如果存在，就直接拿数据，不存在就去数据库中取，也省了很多事。

hibernate使用的是二级缓存，有两个级别，分别是Session，SessionFactroy级别，一级缓存是Session级别的缓存，Hiberante来管理的，这个我们一般不用管，二级缓存是SessionFactory级别的，可以进行配置和更改，加载卸载，就是我们平时用的hibernate配置文件中的配置SessionFactory的属性。

对一级缓存中的session事务进行数据的预存取，数据在各个session的上下文中存在（单例），当session关闭时，就是session.close( )，整个所有session范围内的数据全部清空；二级缓存默认的情况下是不开启的，如果开启，数据库汇总的所有数据都先copy到二级缓存中，这里要注意的是，一级缓存是每个线程下创建的session缓存区域，可以是多个，二级缓存只有一个区域 ，画个图比较容易理解。

| 一级session缓存| | 一级session缓存| |一级session缓存| |一级session缓存|

| | | |

| | | |

—————————————————————————————————

sessionFactroy级别的二级缓存

——————————————————————————————————————-

|

|

|

——————————————————————————————————————

DB

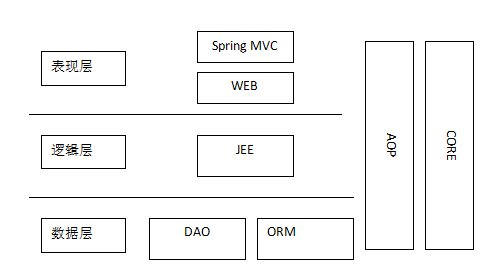
—————————————————————————————————————–

**Spring**

1. **是什么？**

Spring是基于JEE的轻量级的应用框架

1. **有什么？**



每个包的功能：

Spring MVC：spring 本身提供的web 框架

WEB：集成其他的web 应用的框架，或者说是对其他web应用框架的支持。如struts，webwork

JEE ：集成一系列的jee的技术（对JEE技术的支持）

DAO：封装了JDBC；（对JDBC的支持）

ORM：提供了对ＯＲＭ工具的集成（支持）

AOP ：面向切面编成

CORE：spring的核心包，提供bean的工厂和IOC容器

1. **能干什么？**

把一系列的jee的技术有效的组合在一起形成以良好的系统

1. **怎么用？**
   1. **搭建web工程，引入spring的jar包**
   2. **在web.xml中添加如下配置**

<context-param>

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

            <param-value>classpath\*:applicationContext\*.xml</param-value>

    </context-param>

        <filter>

            <filter-name>struts2</filter-name>

            <filter-class>

                org.apache.struts2.dispatcher.FilterDispatcher

            </filter-class>

        </filter>

        <filter-mapping>

            <filter-name>struts2</filter-name>

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

</filter-mapping>

<listener>

            <listener-class>

                org.springframework.web.context.ContextLoaderListener

            </listener-class>

        </listener>

1. **部署**

1. **容器和bean**

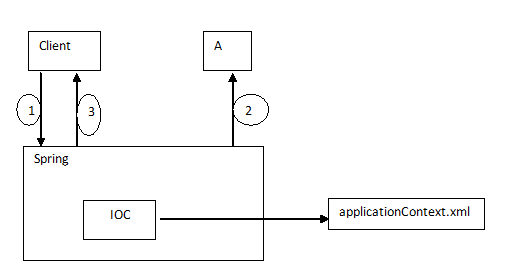
Bean ：是指受spring的ioc管理的对象称为bean

容器 ：（与jee的容器类比）

Jee ：提供组件的运行环境和管理组件的生命周期（不能单独存在）

Spring ：提供bean的运行环境和管理bean的生命周期（可以单独存在）

1. **IOC控制反转**

****

从前的应用程序想要得到A的实例他会直接主动的去拿，当用了sping后，应用程序由主动的去取变成被动的等待，由spring来控制应用程序所需要的对象

* 1. IOC的优点

1.可以通过IOC来获得实例

2.可以通过DI来获取注入的资源

* 1. 配置IOC的原则
     1. 一定是可变的资源才采用依赖注入
     2. 一定是层间的接口或者是模块之间的相互调用才采用依赖注入

1. **DI依赖注入**

1. 应用程序依赖spring注入所需要的对象

IOC和DI是对同一种事情的不同描述

2.setter注入： 在配置文件中将接口的实现配置为bean在应用程序中注入bean

例如：

在配置文件中

<bean name=”ebi” class=”com.javakc.business.ebo”>

<property name=”dao” ref=”dao”>

</bean>

在应用程序中

Public DBDAO dao ;

Public void setDao(DBDAO dao){

This.dao = dao;

}

3．构造器注入

<constructor-arg>

4．ref 表示参照其它的bean

在参照的过程中一定要注意死循环

5.自动装配———–〉no

自动装配根据名字来匹配相应的bean 尽量不要去自动装配

6.lookup注入

7.singleton

1.单例模式是整个的jvm中只有一个实例

2.spring的singleton是指在spring的容器中只有一个实例

一个生命周期中只有一个实例

8.DI的优点：

1.程序被动等待，强化面向接口编成

2.切断了对象或组件之间的联系，使程序的结构更加松散，运行和维护更加简单

1. **Aop面向切面编程**

**1.AOP面向切面编程 一些较好的模式或者是示例—-范式**

切面：一个切面代表我们所关注的一系列的共同的功能点（模块之间的共同的功能点）

**2.AOP的思想： 主动—->被动（追加功能）**

**3.AOP 的概念**

1.切面 ：我们所关注的功能点

2.连接点 ：事件的触发点（方法的执行）

3.通知 ：连接点触发时执行的动作（方法）

4.切入点 ：一系列的连接点的集合 （连接点的模块化）

5.引入 ：扩展的功能

6.目标对象 ：包含连接点的对象

7.aop代理 ：实现机制

8.织入 ：把advice和目标对象连接起来

**4.AOP的事件机制**

1.通过切面找出一系列共同的功能点

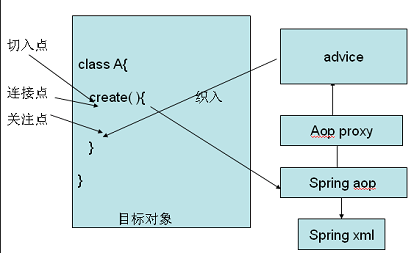
2.找到目标对象(在标准的spring 中 接口的实现类为target)

3.找到切入点

4.确定连接点

5.通知spring AOP，查询xml文件，通过代理找到advice。

6.由aop代理来实现动态织入



**5. AspectJ**

**5.1.在xml中配置比较烦琐**

所有的入口必须从一个代理（ProxyFactoryBean）开始

<bean id=“myTarget” class=“com.javakc.aop.MyTarget”> </bean>

         <bean id=“myAdvice” class=“com.javakc.aop.MyAdvice”>  
</bean>  
  
         <aop:config>

        <aop:aspect ref=“myAdvice”>

        <aop:pointcut id=“kc”

expression=“execution(\* com.javakc.aop.MyTarget.t\*())”/>

        <aop:before pointcut-ref=“kc” method=“abc”/>

        </aop:aspect>

    </aop:config>

**5.3.使用注解的方法相对简单**

@AspectJ的基本语法

1.@Aspect声明一个切面，将一系列的共同的功能定义成一个切面

直接在类上定义@Aspect

2.@Pointcut声明切入点

2.1、用一个专门的类来定义pointcut，类中的方法名就是该pointcut的名字

2.2、可以使用匿名的pointcut

2.3、执行切点的几种方法

2.3.1 execute（public \* 包结构.\*.\*.（..）） 可以指定到具体的方法

2.3.2 within 指定到包，不能指定到类

within（”com.javakc.spring..\*”）

2.3.3 this 指定到实现接口的所有的实现类

2.3.4 target 指定具体的实现类

**5.4.advice的五种类型的示例**

**客户端必须从接口走才能得到监控，实现想要追加的功能**

**5.4.1.@AfterReturning(pointcut=”” returning=”retVal”)**

追加的方法的参数名字一定要与retrning的名字相同

在注解@AfterReturning中必须加上pointcut和returning两个参数

pointcut指所要监控的目标对象的方法

得到目标对象的方法的返回值，来作为参数，进行下一步的处理,参数没有顺序，按参数的名字进行匹配

完成追加的功能

1 定义一个pointcut，通过方法名来作为pointcut的名称来引用

(1).@AfterReturning(“com.javakc.spring.schemaaop.TestPointcut.t4()”)

(2).

     2.直接引用匿名的pointcut

     (1).@AfterReturning(“execution(

\* com.javakc.spring.schemaaop.Api.test4())”)

     (2).@AfterReturning(pointcut=

“com.javakc.spring.schemaaop.TestPointcut.t4() &&

args(str)”, returning=”retVal”)

@AfterReturning (pointcut=”com.javakc.spring.schemaaop.TestPointcut.t4() && args(str)”,returning=”retVal”)

public void testAfterReturning(String str,Object retVal){

    System.out.println(“afterReturning1=>”+retVal+”=>”+str);

    }

**5.4.2.@Aronud**

     注解@Around环绕追加功能；

     在执行目标对象的方法的前、后追加功能；

     必须有参数；第一个参数的类型必须为ProceedingJoinPoint；

     通过ProceedingJoinPoint的实例的proceed来调用所监控的

目标对象的方法

     1 定义一个pointcut，通过方法名来作为pointcut的名称来引用

     (1).@Around(“com.javakc.spring.schemaaop.TestPointcut.t1()”)

     (2).@Around(“com.javakc.spring.schemaaop.TestPointcut.t2()

&& args(str)”)

2.直接引用匿名的pointcut

     (1).@Around(“execution(

\* com.javakc.spring.schemaaop.Api.test1())”)

     (2).@Around(“execution(

\* com.javakc.spring.schemaaop.Api.test2(..)) && args(str)”)

// @Around(“com.javakc.spring.schemaaop.TestPointcut.t2() && args(str)”)

     @Around(“execution(\* com.javakc.spring.schemaaop.Api.test2(..)) && args(str)”)

public void testAround(ProceedingJoinPoint prj,String str) throws Throwable{

    System.out.println(“around1==========before1pointcut==>”+str)

        Object obj = prj.proceed();

        System.out.println(“around1==========after1pointcut==>”+str);

    }

**5.4.3.@Before**

注解@Before在执行目标对象的方法前追加相应的功能

1 定义一个pointcut，通过方法名来作为pointcut的名称来引用

(1).@Before(“com.javakc.spring.schemaaop.TestPointcut.t1()”)

(2).@Before(“com.javakc.spring.schemaaop.TestPointcut.t2() && args(str)”)

注意args后的名称与参数名相同

2.直接引用匿名的pointcut

(1).@Before(“execution(\* com.javakc.spring.schemaaop.Api.test1())”)

(2).@Before(“execution(\* com.javakc.spring.schemaaop.Api.test2(..)) && args(str)”)

注意args后的名称与参数名相同

// @Before(“com.javakc.spring.schemaaop.TestPointcut.t2() && args(str)”)

@Before(“execution(\* com.javakc.spring.schemaaop.Api.test2(..)) && args(str)”)

public void testBeforeParam(String str){

    System.out.println(“before1=param=>”+str);

}

**5.4.4.@After**

注解@After在执行目标对象的方法后追加相应的功能

1 定义一个pointcut，通过方法名来作为pointcut的名称来引用

     (1).@After(“com.javakc.spring.schemaaop.TestPointcut.t1()”)

     2.直接引用匿名的pointcut

     (1).@After(“execution(\* com.javakc.spring.schemaaop.Api.test1())”)

@After(“com.javakc.spring.schemaaop.TestPointcut.t1()”)

    public void testAfter(){

    System.out.println(“after1== >pointcut”);

    }

**5.4.5.@AfterThorwing**

1. **描述一下spring中BeanFactory和ApplicationContext的差别**

BeanFactory是一个Interface,这是Spring的核心。它仅仅只是提供的一些基本功能。ApplicaionContext 也是一个interface,这是从BeanFactory继承过来的所以它具备了BeanFactory的所有功能。但它也从其它的类或interface得到了一些特性。比如提供国际化的消息访问，资源访问，事件传播。

但其主要区别在于BeanFactory是延迟加载,如果Bean的某一个属性没有注入，BeanFactory加载后，直至第一次使用getBean方法调用此Bean时才会抛出异常；而ApplicationContext则在初始化自身时检验，这样有利于检查所依赖属性是否注入；所以通常情况下我们选择使用ApplicationContext.

代码示例：

BeanFactory beanFactory = new XmlBeanFactory(new FileSystemResource(“beans.xml”));

ApplicationContext cxt = new ClassPathXmlApplicationContext(“beans.xml”);

1. **谈谈spring对DAO的支持**

Spring提供的DAO(数据访问对象)支持主要的目的是便于以标准的方式使用不同的数据访问技术。

简化 DAO 组件的开发。  
Spring提供了一套抽象DAO类供你扩展。这些抽象类提供了一些方法，用来简化代码开发。

IoC 容器的使用，提供了 DAO 组件与业务逻辑组件之间的解耦。所有的 DAO 组件，都由容器负责注入到业务逻辑组件中，其业务组件无须关心 DAO 组件的实现。

面向接口编程及 DAO 模式的使用，提高了系统组件之间的解耦，降低了系统重构的成本。

方便的事务管理: Spring的声明式事务管理力度是方法级。

异常包装:Spring能够包装Hibernate异常，把它们从CheckedException变为RuntimeException; 开发者可选择在恰当的层处理数据中不可恢复的异常，从而避免烦琐的 catch/throw 及异常声明。

1. **谈谈spring对hibernate的支持**

在所有的 ORM 框架中， Sping 对 Hibernate 的支持最好。如 SessionFactory 的注入、HibernateTemplate 的简化操作及 DAO 支持等。另外， Spring 还提供了统一的异常体系及声明式事务管理等。

一旦 Hibernate 处于 Spring 的管理下， Hibernate 所需要的基础资源，都由 Spring 提供注入。 Hibernate 创建 SessionFactory 必需的 DataSource ，执行持久化必需的 Session 及持久层访问必需的事务控制等，这些原本必须通过代码控制的逻辑，都将由Spring 接管:DataSource, SessionFactory, TransactionManager等，都将作为 Spring 容器中的 bean。将这些bean 放在配置文件中管理。

1、通用的资源管理: Spring 的 ApplicationContext 能管理 SessionFactory，使得配置值很容易被管理和修改，无须使用Hibernate 的配置文件。详细配置如下：

<bean id=“myDataSource”

        class=“org.apache.commons.dbcp.BasicDataSource”>

        <property name=“driverClassName”>

            <value>oracle.jdbc.driver.OracleDriver</value>

        </property>

        <property name=“url”>

            <value>jdbc:oracle:thin:@localhost:1521:orcl</value>

        </property>

        <property name=“username”>

            <value>javakc2</value>

        </property>

        <property name=“password”>

            <value>javakc2</value>

        </property>

    </bean>

    <bean id=“mySessionFactory”

        class=“org.springframework.orm.hibernate3.LocalSessionFactoryBean”>

        <property name=“dataSource” ref=“myDataSource” />

        <property name=“mappingResources”>

            <list>

                <value>com/javakc/spring/h3/UserModel.hbm.xml</value>

            </list>

        </property>

        <property name=“hibernateProperties”>

            <value>

                hibernate.dialect=org.hibernate.dialect.OracleDialect

            </value>

        </property>

    </bean>

    <bean id=“h3″ class=“com.javakc.spring.h3.H3Test”>

        <property name=“sessionFactory” ref=“mySessionFactory”></property>

    </bean>

如果需要使用容器管理的数据源，则无须提供数据驱动等信息，只需要提供数据源的JNDI 即可。对上文的SessionFactory只需将dataSource的配置替换成 JNDI 数据源，并将原有的 myDataSource Bean 替换成如下所示:

    <bean id=“myDataSource”

        class=“org.springframework.jndi.JndiObjectFactoryBean”>

        <property name=“jndiName”><!–指定数据源的JNDI–>

            <value>java:comp/env/jdbc/myds</value>

        </property>

    </bean>

2、有效的 Session 管理: Dao类继承HibernateDaoSurport后，Spring 提供了 HibernateTemplate，用于持久层访问，该模板类无须显示打开 Session及关闭 Session。它只要获得 SessionFactory 的引用，将可以智能打开 Session，并在持久化访问结束后关闭 Session ，程序开发只需完成持久层逻辑，通用的操作则由HibernateTemplate 完成。

3、统一的事务管理。无论是编程式事务，还是声明式事务， Spring 都提供一致的编程模型，无须烦琐的开始事务、显式提交及回滚。

建议使用声明式事务管理，Spring 的声明式事务以 Spring 的 AOP 为基础。可将事务管理逻辑与代码分离。代码中无须实现任何事务逻辑，程序开发者可以更专注于业务逻辑的实现。声明式事务不与任何事务策略藕合，采用声明式事务可以方便地在全局事务和局部事务之间切换。

Spring声明性事务有两种配置方法，一种是xml配置：

    <bean id=“txManager”    class=“org.springframework.orm.hibernate3.HibernateTransactionManager”>

        <property name=“sessionFactory” ref=“mySessionFactory” />

    </bean>

    <tx:advice id=“txAdvice” transaction-manager=“txManager”>

        <tx:attributes>

            <tx:method name=“\*” propagation=“REQUIRED” />

        </tx:attributes>

    </tx:advice>

    <aop:config proxy-target-class=“true”>

        <aop:pointcut id=“test” expression=“execution(\* com.javakc.spring.h3.JdbcTest.\*(..))” />

        <aop:advisor advice-ref=“txAdvice” pointcut-ref=“test”/>

    </aop:config>

另一种是通过注解，在需要添加事务的类上注明 @Transactional

|  |
| --- |
| **Spring2.0之前事务的写法**  **<bean id=”txProxyTemplate”**  **class=”org.springframework.transaction.interceptor.TransactionProxyFactoryBean”**  **abstract=”true”>**  **<property name=”transactionManager” ref=”transactionManager” />**  **<property name=”transactionAttributes”>**  **<props>**  **<prop key=”get\*”>PROPAGATION\_REQUIRED,readOnly</prop>**  **<prop key=”\*”>PROPAGATION\_REQUIRED</prop>**  **</props>**  **</property>**  **</bean>** |

1. **谈谈Spring对事务的支持**

1、声明式事务管理：

1.流程：由客户端访问—-aop监控—-调用advice来追加事务

2.做法：

2.1 在配置文件的头中引入xmlns：tx 和schema的文件

2.2 <aop:aspectj-autoproxy/>

2.3 注入数据源

<bean id=”dataSource” class=”org.apache.commons.dbcp.BasicDataSource” destroy-method=”close”>

<property name=”driverClassName” value=”oracle.jdbc.driver.OracleDriver”/>

<property name=”url” value=”jdbc:oracle:thin:@127.0.0.1:1521:orcl”/>

<property name=”username” value=”aa”/>

<property name=”password” value=”aa”/>

</bean>

2.4 由spring实现的事务管理，但需要注入数据源

<bean id=”txManager” class=”org.springframework.jdbc.datasource.DataSourceTransactionManager”>

<property name=”dataSource” ref=”dataSource”/>

</bean>

2.5 事务监控所有的方法（事务加强）

<tx:advice id=”txAdvice” transaction-manager=”txManager”>

<tx:attributes>

<tx:method name=”create\*” propagation=”REQUIREDNEW”/>

</tx:attributes>

</tx:advice>

2.6 定义切入点

<aop:config>

<aop:pointcut id=”my”

expression=”execution(\* com.javakc.spring.schemaaop.api.\*.\*(..))”/>

<aop:advisor advice-ref=”txAdvice” pointcut-ref=”my”/>

</aop:config>

2、使用注解实现事务管理：

1.注解@Transcational

（可以在类上，也可以在方法上）

2.在配置文件中同样需要注入dataSource和spring的事务管理

3.使用注解的方法来追加事务 注解驱动

<tx:annotation-driven transaction-manager=”txManager”/>

**如何在Spring中使用Hibernate的事务：**

<bean id=”txManager” class=”org.springframework.orm.hibernate.HibernateTransactionManager”>

<property name=”sessionFactory” ref=”sessionFactory” />

</bean>

**如何在Spring中使用JTA的事务：**

<bean id=”txManager” class=”org.springframework.transaction.jta.JtaTransactionManager” />

**事务**

1. **ACID**

（1）原子性（A）：一系列的操作具有不可分割性，要么成功，要么失败

（2）一致性 ：操作的前后满足同样的规则，保持平衡。

（3）隔离性 ：在事务运行的过程中。事务内的数据对外的可见性和可操作性（必需由数据库的支持）

（4）稳定性 ：保证数据在事务的操作的过程中不会遭到破坏，而且是永久性存储的

1. **事务模型**

（1）经典事务模型又叫平面事务模型（JDBC） 事务不允许嵌套

（2）嵌套事务模型

1. **事务的两种方式**

**（1）声明式**

在配置文件中设置以下6项

（１）.required

如果客户端没有事务　在bean中新起一个事务

如果客户端有事务bean 中就加进去

(2)、 requiresNew

不管客户端有没有事务服务器段都新起一个事务

如果客户端有事务就将事务挂起

(3)、supports

如果客户端没有事务服务端也没有事务

如果客户端有事务服务端就加一个事务

(4)、mandatcry

如果客户端没有事务服务端就会报错

如果客户端有事务服务端就加事务

(5)、notSupported

不管客户端有没有事务服务端都没有事务

如果客户端有事务服务端就挂起

(6)、never

不管客户端有没有事务服务端都没有事务

如果客户端有事务就报错

**（2）编程式事务**

Javax.transaction.UserTranscation

JTA 事务可以精确到事务的开始和结束

1. **事务的隔离性**

Read-uncommited

读取未提交的数据（基本上没有锁）脏读，存在不当读取的问题。

Read-commited

读取已提交的数据（行级锁）

无法读取已经写入，但没有被提交的数据。

Repeatable\_read

读取的数据不允许修改或删除

不可重复读（行级锁+不允许修改/删除）

Serializable

幻影读（表级锁）

**struts**

**开发流程**

1、创建表单

2、从表单中获得输入

3、处理输入（业务逻辑）

4、根据动态输入改变输入流

**要完成上述四步，我们需要创建**

1、一个ActionForm

2、一个action

3、一个配置文件struts-config.xml

4、创建页面

**创建ActionForm**

1、写一个类继承ActionForm

2、私有的属性

3、相应的get、set方法

4、重写tostring、equals、hashcode三个方法

**创建action**

1、写一个类继承Action

2、实现一个公有的方法（回调方法）

public ActionForward execute(ActionMapping mapping,

ActionForm form,

HttpServletRequest request,

HttpServletResponse response)

{

//1.收集参数

MyActionForm myForm = (MyActionForm)form;

//2.组织参数

//3.调用逻辑层

boolean flag = true;

//4.根据返回值来跳转到相应的页面

ActionForward af = new ActionForward();

if(flag){

af = mapping.findForward(“1″);

}else{

af = mapping.findForward(“2″);

}

return af;

}

**配置struts-config.xml文件**

1.<form-beans>

<form-bean name=”myActionForm” type=”全路径.MyActionForm”/>

<form-bean />

</form-beans>

2.<action-mappings>

<action path=”/sll”

name=”myActionForm”

type=”全路径.MyAction”

scope=”session”

input=”错误返回的页面”>

<forward name=”1″ path=”/1.jsp”>

<forward name=”2″ path=”/2.jsp”>

</action>

</action-mappings>

**步骤:**

1.创建一个空的web应用

2.将struts的包放到lib文件夹下

3.将.tld文件文件和struts-config.xml、web.xml放到WEB-INF的跟目录下

4.配置struts-config.xml文件和web.xml文件

5.在页面引入tag文件uri

**STRUTS运行机制**

1、界面点击产生请求

2、容器接到请求

3、匹配web.xml文件中的\*.do来调用ActionServlet

4、ActionServlet的处理

4.1 读struts-congfig.xml文件形成ActionMapping

4.2 通过path 来匹配Action类，通过Action的name属性来匹配ActionForm类

4.3 通过反射机制来给Form添数据

4.4 由ActionServlet转调Action的execute方法

4.5 得到execute方法的返回值，跳转页面

4.5.1 RequestDispatcher

4.5.2 response.sendRedirect(“list.jsp”);

进入execute方法

1 收集参数

2 组织参数

3 调用，逻辑层

4 返回值

4.1 选择下一个页面 （ActionForward）

4.2 把值传给下一个页面

**比较struts和struts2的不同**

|  |  |  |
| --- | --- | --- |
|  | Struts | Struts2 |
| 组成 | 五个部分：ActionServlet，ActionMapping，AtcionForward，ActionForm，Action | 十个部分：AcionContext，FilterDispatcher，ActionMapper，ActionProxy，Configuration Manager，Action Invocation，Interceptot，Action，Result，Taglib |
| url | .do | .action |
| 分发控制 | AtcionServlet | FilterDispatcher |
| 收集参数 | 利用JavaBean技术，将数据封装到ActionForm的子类中；  一个Action对应一个ActionForm的子类 | 使用拦截器将数据封装到值栈中。  使用域模型，一个Action对应多个值栈中的Model |
| 读取配置文件 | ActionMapping | Configuration Manager |
| 拦截器 | 无拦截器 | 执行Action前后有拦截器 |
| 返回页面 | 将字符串封装到AtcionForward，进行跳转页面 | Action中直接返回字符串，通过Result跳转页面 |
| 页面跳转方式 | 两种：dispatch，redirect | 三种：dispatch，redirect，chain |
| 值栈 | 无值栈，Action通过在request范围内设置属性，向jsp页面传递数据 | 有值栈，Action和jsp页面通过值栈传递数据，jsp页面通过OGNL取得值栈中的数据 |
| 配置文件 | struts-config.xml | struts.xml |
| 中文问题 | 不能解决（需要使用filter） | 可以解决中文问题：  1、在jsp页面添加Meta；  2、修改struts.properties中的设置。 |

**UML**

1. **是什么？**

一种标准的图形化的建模语言

1. **能干什么？**

1. **有什么？**

1.视图

1.1、用例视图

1.2、逻辑视图

1.3、并发视图

1.4、组件视图

1.5、展开视图（部署视图）

2.图

2.1、用例图 ：指用户、系统、用例之间的关系

2.2、类图 ：表示系统中的类和类之间的关系

2.3、对象图 ：表示类的实例图

2.4、状态图 ：用来描述对象的状态和引起状态变化的事件

2.5、时序图 ：按着时间的顺序描述对象之间的交互关系

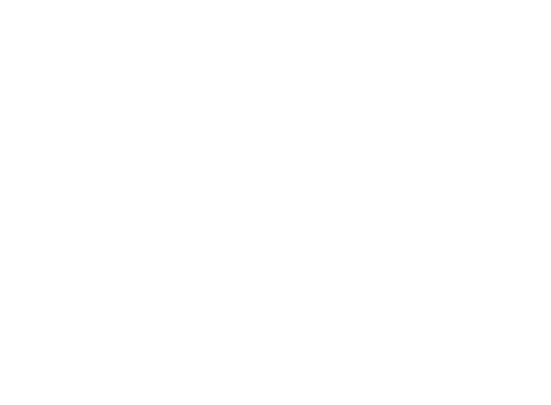
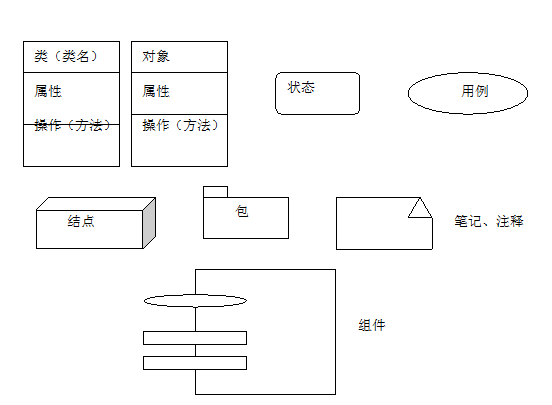
2.6、协作图 ：按照空间的顺序描述对象之间的协作（调用）的关系

2.7、组件图 ：描述组件及其组件之间的关系

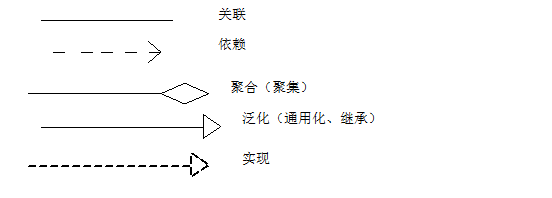
2.8、展开图 ：描述系统中软、硬件的物理架构（软件在硬件上的分布）

2.9、活动图 ：描述活动的流程（流程图）

3.模型元素：



类之间的五种关系：



关联：类A与类B之间存在特定的对应关系。

依赖：类A访问类B提供的服务。

聚集：类A为整体类，类B为局部类，类A的对象由类B的对象组合而成。

泛化：类A继承类B。

实现：类A实现B接口。

    聚集是关联的一种特殊形式。

4.通用机制

5.扩展机制

用例视图————————–用例图——————————需求分析

逻辑视图（静态）——————-类图—————————-详细设计

————————-活动图————————–需求分析

并发视图（动态）——————-对象图————————-详细设计

——————-状态图————————-详细设计

——————-时序图————————-详细设计

——————-协作图————————-详细设计

组件视图 —————————–组件图————————–概要设计

展开视图—————————- –展开图————————–概要设计（雏形）

————————详细设计

—————————测试

**用例：**

1. 用例：代表系统的一个完整的功能
2. 系统：在这里并不是指一个完整的系统，是一部分功能的集合
3. 角色：指与系统交互的人或事

**思维方式：**

1. 角度 ：站在系统的角度来看角色（用户）、再站在用户的角度来看系统
2. 度 ：对于任何事情都要把握度
3. 分层 ：

**消息：**

1. 简单消息：
   1. 同步消息（调用）：在调用过程中一定要等待返回
   2. 异步消息（调用）：在调用过程中不需要等待返回，可以处理其他的事情
2. 状态图的要素
   1. 状态
   2. 转移
   3. 转移的条件
3. 活动图的要素
   1. 活动
   2. 条件
   3. 变迁（转移）
4. 泳道：活动图的一种，是按照一定的规则来划分

如：按部门、阶段等；

**其他开发工具和技术**

**Ant**

Ant是基于java的批处理工具

一、配置ant的运行环境

1.将ant的bin目录添加到path中

2.配置JAVA\_HOME

3.配置ANT\_HOME

二、配制build.xml文件，该文件放在应用程序的根目录下

编译java文件

执行java文件

copy

删除

打jar包

根据注解生成配置文件

**Xdoclet**

通过注释生成一系列文件的工具（txt、xml、java、html等）

xdoclet本质是摸板技术+字符串的替换

1.在java文件中写符合要求的注释（将hbm.xml文件的内容搬到java文件中去写）

1.1 在类的上面 写与表的对应

1.2 将每一个属性的注释都写到get方法的上面

2.在ant中引入相应的task来驱动xdoclet

**log4j日志管理**

**1、是什么？**

在开发期间用来测试、对整个日志信息进行管理的工具

**2、功能**

1.控制输出的目的地

2. 控制输出的格式

3. 控制输出的级别

**3、日志的级别**

1. debug 调试

2. info 给用户的提示信息

3. warn 给用户的警告信息

4.error 给程序员用来调试

Debug—-〉info—-〉warn——–〉error

**4、配置**

1.配置级别

2.输入源 （控制台和文件）

3.可以进行分包控制

Log4f.logger.包结构 = 级别

**5、使用**

Logger log=Logger.*getLogger*(Test.**class**);

**public**  
**void** t3(){

        log.debug(“this is debug”);

        log.info(“this is info”);

        log.warn(“this is warn “);

        log.error(“this is error”);

    }

**Junit**

**1.是什么？**

单元测试的框架

**2.怎么做？**

写一个类继承TestCase

测试的方法一般都以test开头并且没有参数

在测试方法前执行的方法public void setUp( )

在测试方法后执行的方法public void tearDown( )

**error和故障的区别**

error ：代码有问题

故障 ：逻辑有问题与祈望的值不相符合

**生命周期**

测试—〉SetUp（）—〉testXX（）—〉tearDown（）—〉结束；

**TestCase套件**

public class MyTest{

        Public static Test suite(){

TestSuite suite = new TestSuite();

suite.addTestCase(Test1.class);

suite.addTestCase(Test2.class);

}

}

**Ajax**

1. 是什么？

AJAX即“Asynchronous JavaScript and XML”（异步JavaScript和XML)。是一种创建交互式网页应用的网页开发技术。

无刷新页面。

1. 有什么？

AJAX是多种技术的综合，包括JavaScript、XHTML和CSS、DOM、XML 和 XSLT、XMLHttpRequest。

1. 能干什么？

使用XHTML+CSS标准化呈现；  
使用DOM进行动态显示及交互；  
使用 XML 和 XSLT 进行数据交换及相关操作；  
使用 XMLHttpRequest 进行异步数据查询、检索；  
使用JavaScript绑定和处理有所得数据；

传统的web应用允许用户填写表单(form)，当提交表单时就向web服务器发送一个请求。服务器接收并处理传来的表单，然後返回一个新的网页。这个做法浪费了许多带宽，因为在前後两个页面中的大部分HTML代码往往是相同的。由于每次应用的交互都需要向服务器发送请求，应用的响应时间就依赖于服务器的响应时间。这导致了用户界面的响应比本地应用慢得多。

与此不同，AJAX应用可以仅向服务器发送并取回必需的数据，它使用SOAP或其它一些基于XML的web service接口，并在客户端采用JavaScript处理来自服务器的响应。因为在服务器和浏览器之间交换的数据大量减少，结果我们就能看到响应更快的应用。同时很多的处理工作可以在发出请求的客户端机器上完成，所以Web服务器的处理时间也减少了。

使用Ajax的最大优点，就是能在不更新整个页面的前提下维护数据。这使得Web应用程序更为迅捷地回应用户动作，并避免了在网络上发送那些没有改变过的信息。

对应用Ajax最主要的批评就是：  
1 它破坏了浏览器后退按钮的正常行为，就是用户点击后退按钮后不能看到之前的数据；  
2 一些手持设备（如手机、PDA等）现在还不能很好的支持Ajax；  
3 用JavaScript作的Ajax引擎，JavaScript的兼容性和DeBug都是让人头痛的事；  
4 Ajax的无刷新重载，由于页面的变化没有刷新重载那么明显，所以容易给用户带来困扰――用户不太清楚现在的数据是新的还是已经更新过的；现有的解决有：在相关位置提示、数据更新的区域设计得比较明显、数据更新后给用户提示等;

1. 怎么做？

<script language=”javascript”>

var req;

function add(){

//1:创建用于传输的对象

req=new ActiveXObject(“Microsoft.XMLHTTP”);

// 声明返回后，处理数据的方法

req.onreadystatechange=manage;

//2:准备传输的内容

var str=”test1111″;

//3:发出请求

req.open(“GET” ,”/others/ajax/test/t2.jsp?str=”+str);

req.send(null);

}

//4:接受返回并处理

function manage(){

if(req.readyState==4){

alert(“r==”+req.responseText);

}

}

</script>

readyState的取值如下：

　　0 (未初始化)

　　1 (正在装载)

　　2 (装载完毕)

　　3 (交互中)

　　4 (完成)

1. 原理

Ajax的原理简单来说通过XmlHttpRequest对象来向服务器发异步请求，从服务器获得数据，可以是xml格式，也可以是一个字符串，然后用javascript来操作DOM而更新页面。

**Dwr**

1. 是什么？

DWR（Direct Web Remoting）是一个开源的类库，可以帮助开发人员开发包含AJAX技术的页面。它可以允许在页面中使用运行在WEB服务器上的JAVA函数,被调用的JAVA函数就像是在页面中JavaScript方法一样。

1. 有什么？

它包含两个主要的部分：允许JavaScript从WEB服务器上一个遵循了AJAX原则的Servlet中获取数据；另外一方面提供了一个JavaScript库可以帮助web开发人员轻松地利用获取的数据来动态改变网页的内容。

1. 能干什么？

动态地把java类生成为JavaScript，感觉调用就像发生在浏览器端，实际上代码调用发生在服务器端，Dwr负责数据的传递和转换。这种从Java到JavaScript的远程调用功能的方式使DWR用起来有种非常像RMI或者SOAP的常规RPC机制。

1. 怎么做？

① 编写配置文件

**web.xml**

<servlet>

<servlet-name>dwr-invoker</servlet-name>

<servlet-class>

org.directwebremoting.servlet.DwrServlet

</servlet-class>

<init-param>

<param-name>debug</param-name>

<param-value>true</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>dwr-invoker</servlet-name>

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

</servlet-mapping>

**dwr.xml**

<allow>

<create creator=”new” javascript=”kc2″>

<param name=”class” value=”dwr.Service” />

</create>

</allow>

②编写**service**

public class Service {

public String sayHello(String yourName) {

//可以是访问数据库的复杂代码

return “Hello World ” + yourName;

}

}

③测试**DWR**

将代码放入应用服务器（比如Tomcat），启动。

然后在地址栏输入http://localhost:8080/你的工程/dwr

④编写jsp页面

1） 要引用3个dwr的js，其中engine.js必须要，如果需要用到dwr提供的一些方便的工具要引用util.js

2） 然后还要引用dwr自动生成的js，dwr/interface/service.js，注意js名字要和dwr.xml配置的一样

3） js里的service.sayHello和java类的那个有一点区别，多了一个参数，用来得到返回的数据

**SQL与Oracle**

1. **SQL分类：**

DDL—数据定义语言(CREATE，ALTER，DROP，DECLARE) 用于建立，修改，删除数据库中的各种对象—–表、视图

DQL—数据查询语言（SELECT）用于检索数据库

基本结构是由SELECT子句，FROM子句，WHERE子句组成的查询块：

SELECT <字段名表>

FROM <表或视图名>

WHERE <查询条件>

ORDER BY字段名表>

DML—数据操纵语言(DELETE，UPDATE，INSERT)用于改变数据库数据

主要有三种形式：

插入：INSERT

更新：UPDATE

删除：DELETE

DCL—数据(事务)控制语言(GRANT，REVOKE，COMMIT，ROLLBACK) 用于维护数据的一致性

包含五条语句：

提交COMMIT;

回滚ROLLBACK;

标记SAVEPOINT

授权GRANT

撤回REVOKE：。

**二、基本语法**

**1、创建数据库**create database database-name

**2、删除数据库**drop database dbname

**4、创建新表**create table tabname(  
col1 type1 [not null] [primary key],  
col2 type2 [not null],  
..  
)

根据已有的表创建新表：

A：create table tab\_new like tab\_old (使用旧表创建新表)(在orcale中不能用)  
B：create table tab\_new as select col1,col2… from tab\_old

**5、删除新表**

drop table tabname  
**6、增加一个列**Alter **table** tabname add column col type

注：列增加后将不能删除。DB2中列加上后数据类型也不能改变，唯一能改变的

是增加varchar类型的长度。  
**7、添加主键**

Alter table tabname add primary key(col)

* 1. **创建索引**

create [unique] index idxname on tabname(col….)  
删除索引

drop index idxname  
注：索引是不可更改的，想更改必须删除重新建。

* 1. **创建视图**

create view viewname as select statement

删除视图：

drop view viewname

* 1. **几个简单的基本的sql语句**

选择：select \* from table1 where 范围

插入：insert into table1(field1,field2) values(value1,value2)

Insert into table1 values(’001′,’sll’)

删除：delete from table1 where 范围  
更新：update table1 set field1=value1 where 范围  
查找：select \* from table1 where field1 like ‘%value1%’

表示模糊查询（匹配字符串）  
排序：select \* from table1 order by field1,field2 [desc]  
总数：select count（\*） as totalcount from table1  
求和：select sum(field1) as sumvalue from table1  
平均：select avg(field1) as avgvalue from table1  
最大：select max(field1) as maxvalue from table1  
最小：select min(field1) as minvalue from table1

* 1. **使用外连接**

A、left join：

左外连接（左连接）：结果集几包括连接表的匹配行，也包括左连接表的所有行。

sql: select a.a, a.b, a.c, b.c, b.d, b.f from a LEFT JOIN b ON a.a

=b.c  
B：right join:  
右外连接(右连接)：结果集既包括连接表的匹配连接行，也包括右连接表的所有行。

C：full join：

全外连接：不仅包括符号连接表的匹配行，还包括两个连接表中的所有记录。

D：等值连接

无条件连接，取两个表的笛卡尔积

* 1. **in 的使用方法**

select \* from table1 where a **[not] in** (‘值1′,’值2′,’值4′,’值6′)

* 1. **两张关联表，删除主表中已经在副表中没有的信息**

delete from table1 where **not exists** ( select \* from table2

Where table1.field1=table2.field1 )

* 1. **日程安排提前五分钟提醒**

sql: select \* from 日程安排 where datediff(‘minute’,f开始

时间,getdate())>5  
**15、一条sql 语句搞定数据库分页**

asc 按升序排列

desc 按降序排列 **select** uuid, userid, username

**from** (**select**  
**rownum** r, uuid, userid, username **from** tbl\_sysuser)

**where** r >= 2

**and** r < 4  
  
**16、前10条记录**

**select** uuid, userid, username

**from** (**select**  
**rownum** r, uuid, userid, username **from** tbl\_sysuser)

**where** r >= 1

**and** r < 11

**17、说明：选择从10到15的记录  
select** uuid, userid, username

**from** (**select**  
**rownum** r, uuid, userid, username **from** tbl\_sysuser)

**where** r >= 10

**and** r <= 15

**一些示例、**

员工工资表

Tbl\_s

|  |  |  |
| --- | --- | --- |
| id | sallary | pid |
| 100 | 8300 | 100 |
| 200 | 9000 | 200 |
| 101 | 9500 | 100 |
| 201 | 9100 | 200 |
| 202 | 8355 | 100 |
| 102 | 8600 | 200 |

查询员工的工资大于所属部门的部门经理的工资的员工信息

参考答案：

**select \* from tbl\_s t where pid in (select pid from tbl\_s where id = pid)**  
**and sallary > (select sallary from tbl\_s where id=t.pid)**

查询亲属的数量大于1的员工的姓名

**select t1.name (select count(\*) from tbl\_r t2 where t1.id=t2.id) 亲属数量 from**

**tbl\_emp t1 Where (select count(\*) from tbl\_r t2 where t1.id=t2.id)>1**

1 使用SQL语句建立员工信息数据表：员工编号（PK）、员工姓名、员工性别、联系电话、备注。员工亲属信息表：员工编码（PK）、亲属编码（PK）、亲属姓名、联系电话、备注。

2使用SQL语句在员工信息表备注前增加住址字段  
Alter table tbl\_employee add address varchar2(50);

3.创建两个表

Create table tbl\_employee (

Uuid varchar2(20) not null primary key,

Name varchar2(20),

Sex varchar2(2),

Tel varchar2(20),

Descripton varchar2(100)

)

Create table tbl\_relation(

Empid varchar2(20) not null,

Uuid varchar2(20) not null,

Name varchar2(20),

Tel varchar2(20)

Description varchar2(100),

Primary key (empid,uuid) 双主键

)

4.使用SQL向员工信息表中插入一条数据：（001，张三，男，010-62570007，北京市海淀区）

　Insert into tbl\_employee values(’001′，’张三’，’男’，’010-62570007′，’北京市海淀区’)

5. 使用SQL变更001员工地址为：北京市朝阳区。  
update tbl\_employee set address=’chaoyanqu’ where id=’001′

6. 由于两张表之间没有建立外键引用，导致亲属信息表中存在垃圾数据，这部分数据的特点是员工编码在员工信息表中不存在，请用SQL语句将这部分数据查询出来

Select \* from tbl\_relation where empid not in(select uuid from tbl\_employee)

7. 为防止再次出现垃圾数据情况，使用SQL语句在亲属表和员工表之间建立外键约束

ALTER TABLE “JAVAKCOA”.”TBL\_REL”

ADD (CONSTRAINT “SF” FOREIGN KEY(“empid”)

REFERENCES “JAVAKCOA”.”tbl\_emp”(“uuid”))

8. 用SQL查询出亲属数量大于1的员工编码、员工姓名，员工亲属数量  
select t1.uuid,t1.name (select count(\*) from tbl\_relation t2 where t2.empid=t1.uuid) ‘亲属数量’ from tbl\_employee t1 where (select count(\*) from tbl\_relation t2 where t2.empid=t1.uuid)>1  
9. 有部分员工的亲属信息录入重复，例如父亲信息录入两次，这类信息的特征是亲属名称相同，但是亲属编码不同，请用SQL语句将这类情况的员工编码、重复的亲属编码，亲属姓名查询出来

Select t1.empid,t1.uuid,t1.name from tbl\_relation t1 where (select count(\*) from tbl\_relation t2 where t2.empid = t1.uuid and t2.name = t1.name )>1

* 1. 有两个表：

TableX有两个字段Code和Name，其中Code为主键，已有记录如下：

|  |  |
| --- | --- |
| http://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java29.pnghttp://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java30.pngCode | Name |
| Zhang | 张三 |
| http://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java31.pngZhao | 赵四 |
| Li | 李五 |

TableY有三个字段Code、Class、Score，其中Code＋Class为主键，已有记录如下：

|  |  |  |
| --- | --- | --- |
| http://www.mianwww.com/wp-content/uploads/2014/05/051314_1254_Java32.pngCode | Class | Score |
| Zhang | C1 | 80 |
| Zhang | C2 | 90 |
| Li | C1 | 85 |

请写SQL：

* 1. 取出学生个数
  2. 取出各个学生的Code及Score合计
  3. 取出没有考试成绩的学生Code及Name
  4. 在两个表作等值联接的情况下，取出Name、Class、Score，请写出SQL语句及输出结果；
  5. 在以TableX为主，两个表作外联接的情况下，取出Name、Class、Score，请写出SQL语句及输出结果；
  6. 在以TableY为主，两个表作外联接的情况下，取出Name、Class、Score，请写出SQL语句及输出结果；

A,Select count(\*) from tableX

B.select x.code,sum(score) from tableX x ,tableY y where x.code = y.code

select code,sum(score) ‘总成绩’ from tbl\_y group by code

C.select \* from tableX where code not in(select code from tableY)

select code,name from tbl\_x where code not in(select code from tbl\_y)

D. select tbl\_x.code,tbl\_x.name,tbl\_y.score from tbl\_x,tbl\_y where tbl\_x.code =tbl\_y.code

用别名：select x.code,x.name,y.score from tbl\_x as x,tbl\_y as y where x.code =y.code

E. select \* from tbl\_employee t1 left outer join tbl\_emprelation t2 on t1.id=t2.empid