JAVA

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.

2：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。

3：什么是 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

4：请写出 Java 的版本分类，以及每种版本的应用方向

三种版本：

J2ME：是面向内存有限的移动终端. 为消费性产品设计的

J2SE: 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

J2EE: 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

5：描述 Java 技术的主要特性

java 虚拟机
垃圾回收
代码安全性

6：描述 Java 虚拟机的功能

提供硬件平台规范
解读独立于平台的已编译的字节码
可当作硬件或软件来实现
可以在 JAVA 技术开发工具上或在 WEB 浏览器上实现

7：简述 Java 代码安全性

1.字节码校验
2.沙箱机制,提供的安全机制

8：简述 Java 代码的运行过程

加载代码　通过类装载器装载 CLASS 文件
校验代码　由字节码校验
执行代码　由解释器将字节码转换成机器码

9：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

10：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路径
JAVA_HOME　供需要运行 JAVA 的程序使用

二：

11：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

12：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
　　*/

13：什么是 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(){}

14：什么是标示符？

标识符：是赋予变量、类、方法的名称。

15：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
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

16：什么是关键字？

关键字就是编程语言与机器语言的编码约定

17：true、false、null、sizeof、goto、const 那些是 Java 关键字

true 、false 、null　　为 JAVA 的关键字

18：Java 的基本数据类型有几种？分别是？

short int long　boolean float double char byte

19：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
字节型 ：	0	-2^7----2^7-1
字符型 ：	' \u0000'	0----2^16-1
short :	0	-2^15----2^15-1
int :	0	-2^31----2^31-1
long :	0	-2^63----2^63-1
float :	0.0f	-2^31----2^31-1

double ： 0.0d -2^63----2^63-1

boolean: false true\false

20：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

21：如何定义 float 类型和 long 型

float f = 1.2f

long l =1.2L

22：什么是变量？

变量：一种在程序中可以改变的标识符

23：变量的定义规则？

1.首字母小写中间用大写字母隔开 其后可以跟字母‘下划线、$和数字

2.具有一定含义

3.首字母必须是字母、$、下划线,

24：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

25：什么是 Javadoc?

按照一定格式生成程序的文档的工具

26：什么是引用类型？

一个用户定义类型, 它可引用类和数组。

27：什么是按值传递？什么是按引用传递？

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）

28：那些是按值传递？那些是按引用传递？

基本数据类型+String 类型 按值传递

除基本数据类型+String 类型以外，比如 Model s[] 按引用传递

29：如何创建一个新对象？如何使用对象中的属性和方法？

public class S{}

通过对象的实例 用(.)来调用属性和方法；

静态的方法和属性，也可以直接用类(.)来调用；

30：简述 new 关键字所做的工作

new 关键字就是为事先声明的对象分配一块内存区域；

31：简述”=” 和”==”的功能和区别

"="赋值,"= ="判断= =前后两个值得内存地址是否相等；

区别：

= ：为赋值表达式

= = ：为逻辑表达式

32：什么是实例变量？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 可修饰的属性；

final 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；

33：简述上述各种变量的定义规则和使用规则？

实例变量：它不需要 static 关键字来声明,只要对象被当作引用，实例变量就将存在；

局部变量：在方法内任意定义变量即为局部变量；

类变量：必须用 static 修饰；

final 变量：不可以在修改的

34：a++和++a 的区别？

a++ : 先使用，后加 1

++a : 先加 1，后使用

34：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

35：请描述>>、<<、>>>的功能

>> ：算术或符号右移位运算符

<< ：算术或符号右移位左移运算符

>>> ：逻辑或非符号右移位运算符

36：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;
```

37：请写出完整的 if 条件控制语句的结构

```
if(布表达式){
      语
}else{
}
```

38：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
}
```

39：请写出完整的 for 语句的结构

```
for(初始语句;条件语句;步长){
}
```

40: 请写出完整的 while 语句的结构

```
while(boolean 语句){
}
```

41：请写出完整的 do while 语句的结构

```
do{
}while(boolean 语句);
```

42：请描述 break 和 continue 的功能和用法

break:终止最近的循环

continue:跳出本次循环,执行下一次循环

//以上不完全

43：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]);
  }
}
```

44：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]);
```

```
    }
  }
```

45：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]);
   }
  }
```

46：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]);
   }
  }
```

二、面向对象

1：OOP（面向对象）语言的三大特征是？

封装性，继承性，多态性

2：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;}

访问权限＋访问修饰符＋返回值＋方法名＋参数列表＋例外列表＋(块)方法内部代码

3：什么是重载？什么情况下出现？

在一个类中出现相同的方法名但参数列表不同时，这种情况就是重载

其本质是创建一个新的成员方法

此方法在同一个类中出现

4：简述重载的规则

1．方法名相同,参数列表不同

2．返回类型可以不同

3．重载父类的方法时访问权限只能放大，例外列表只能少

5：什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，类构成实例化时调用的方法

1．方法名和类名相同

2．无返回值类型

格式：访问权限 类名 {}；

1．有构造时，则无默认

2．无构造方法时，默认为空(无参数,方法体为空)

jc 垃圾回收

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

构造方法：类初始化时调用的方法。可写可不写，如果不写就会有一个默认的构造方法

析构方法：当对象被从内存中删除时调用的方法，在析构成员方法内，可以填写用来回收对象内部的动态空间的代码

6：简述 this 和 super 的功能和用法

this : 访问当前类的属性或方法,不可以在静态方法中使用，在调用其它构造方法时，必须放在该构造方法的第一行

能调用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

super :访问父类

super 只能调用父类的方法和属性，

在调用父类的构造方法时必须放在子类构造方法下面的第一行；

通过 Super (.) 来调用父类的属性和方法

7：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

private 私有的，只能在本类中被访问

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

8：分别说明：在类上、在属性上、在方法上等能使用那些访问修饰符

在类上：public

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

9：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

10：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

11：什么是包装类？Java 中都有哪些包装类

Boolean Byte Short Integer Long Float Double Charactor

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

12：分别表述类的三大特性及其他们的功能

封装性 继承性 多态性

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：同一种功能拥有许多不同的实现方式

13：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

14：什么是方法的覆盖？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,子类可以创建一个与父类方法有不同功能的方法

在同类型中出现 相同的名称、返回类型 方法名和参数列表的方法时

在父子类时

15：方法覆盖的规则？

方法名要和父类中被覆盖的方法名相同,返回类型相同

参数列表要和父类中被覆盖方法的参数列表相同

访问权限要大于等于被覆盖方法的权限

例外列表要小于等于被覆盖方法的例外列表

16：如何调用父类的构造方法？如何调用自己的构造方法？

要调用父类的构造方法通过 在子类的构造方法中第一行写 super(参数)能给父类传参;

要调用自己的构造方法通过 this(参数)或者直接调;

17：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

18：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,也就是一个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

2．new 一个实例，用实例调

19：static 的基本规则

1.类中的静态方法不允许调用非静态的属性和方法,只可以调用静态的属性和方法

2.可以用类名直接调用静态类中的属性和方法

3.静态方法中不允许出现 this 和 super

4.静态方法不能被非静态方法覆盖

5.构造方法不允许声明 static

6.静态变量为类变量，可以直接调

7.非静态变量为实例变量，通过实例调用

20：final 修饰符的功能是？可以用在什么地方？

功能: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

21：final 的基本规则

final 类不能被继承

final 成员变量只允许赋值一次,且只能通过构造方法里赋值

final 局部变量即为常量，只能赋值一次

final 方法不允许被子类覆盖

final 一般用于标记那些通用性的功能不能随意修改

22：什么是抽象类？

抽象类： 一个类中声明方法存在而不是实现，以及带有对已知行为的方法的实现，这样的类通常被称做抽象类

23：抽象类的规则

类内部至少包含一个没有实现体的方法

用 abstract 修饰

不能够直接使用抽象类，必须通过子类继承并且实现

24：什么情况下使用抽象类

当一个类的一个或多个方法是抽象类时；

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；

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

25：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＝＝比较的也是值

26：toString 方法的功能和基本写法

返回一个 String 类型

```
public String toString(){
}
```

30：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("-");

31：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

32：举例说明如何使用 StringBuffer

```
StringBuffer sb = new StringBuffer();
sb.append("aaa");
sb.flush();
```

33：如何给 Java 代码设置系统属性？如何在程序中使用它们

设置在一个扩展名为 properties 的文件，内容为 key、value 的映射例如“a=2”；

```
System.getProperties();
System.setProperties();
```

34：简述 properties 文件的结构和基本用法

结构：

key=value

用法：

System.getProperties 方法返回系统的 Properties 对象。

System.getProperty(String propertyName)方法返回对应名字属性的值。

System.getProperty(String name， String value)重载方法当没有 name 指定的属性时，返回 value 指定的缺省值。

35：什么是接口？接口的定义规则？

接口就是定义多种方法,通过实现接口中所有方法的抽象类.

```
public interface  Tt {
  public void outPut();
  public int tt();
}
```

36：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。

37：什么是多重接口

就是一个类实现多个接口

38：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”。

39：如何在代码中使用接口？

```
public class MyCast implements Tt{
  public void outPut(){}
  public int tt(){
  return 0;}}
```

40：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();
```

41：如何选择接口和抽象类？为什么？

1.优先选择接口

2.优先使用对象组合,少用继承

3.抽象类一般用在定义子类的行为而父类又有特定行为的子类情况中

4.在接口和抽象类的选择上，必须遵守这样一个原则：行为模型应该总是通过接口而不是抽象类定义。

42:什么是异常？

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

43：简述处理异常的两种方式？

抛出和 catch 语句处理

43：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,使程序运行不会因为出现异常而中断

44：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){
        System.out.println("this is an Exeption!");
    }
```

45：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

46: 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{}

47：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

48：什么是预定义异常

java 程序中预先定义好的异常叫做预定义异常

49：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。

50：什么是断言？

用来证明和测试程序的假设。

51：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";

52：什么是集合？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

53：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

54：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);
    }
}
```

55：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);
    }
}
```

56：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);
    }
}
```

57：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
col.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");
}
```

58：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

59：如何给集合排序？

实现 comparable 接口

三、IO

1：什么是流？可画图说明

字符串分解==========OutStream==========>write()方法写到文件中

2：描述 I/O 流的基本接口和类的结构

InputStream

OutputStream

3：代码示例：如何使用 URL 流来进行输入输出

```
try {
   imageSource = new URL("http://mysite.com/~info");
} catch (MalformedURLException e) {
}
```

4：什么是 Unicode？

是一种字符的编码方式

5：代码示例：如何使用 Reader 和 Writer 来进行输入输出

```
InputStreamReader   ir = new InputStreamReader(System.in);
OutStreamReader    or = new OutStreamReader(System.in);
```

6：什么是可序列化？如何实现可序列化？

表示一个数据可以按流式输出

实现 java.io.Serializable 接口

7：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();
}
```

8：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

除了读写文件内容其他的都可以做

9：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();
}
```

10：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")
      )
      );
      while ((currentLine = in.readLine()) != null){
            System.out.println(currentLine);
      }
```

```
}catch (IOException e){
    System.err.println("Error: " + e);
}
```

四、线程

1：什么是线程？

轻量级的进程

2：线程的三个部分是？

处理机

代码

数据

3：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

4：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

4.1 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);
    }
}
```

4.2 继承 Thread

```
class Thread2 extends Thread{
    public void run(){
        System.out.println(extends);
    }
}

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();
        Thread t3 = new Thread2();
        t3.start();
    }
}
```

5：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()

6：什么是线程的优先级

判断哪个线程先执行的级别

7：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

8：什么是守候线程

隐藏在后台持续运行的线程

9：什么是临界资源

指多个线程共享的资源

10：什么是互斥锁，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

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。

11：什么是死锁？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

12：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

13：什么是 url？基本的格式是？

统一资源定位器

Http://www.163.com:port

14：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

15：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

16：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。

17：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);
}
```

18：代码示例：基于 UDP 编程

19：TCP 和 UDP 区别

TCP 能保证传输内容的完整和准确，UDP 不能

五、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

2：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

3：什么是对象粒度

对象中方法的多少就是粒度

4：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

5：设计模式的应用范围

所能解决的特定的一类问题中

6：简述什么是单例模式，以及他解决的问题，应用的环境，解决的方案，模式的本质

在任何时间内只有一个类实例存在的模式
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
解决方案就是保证一个类只有一个类实例存在
本质就是实例共用同一块内存区域

7：代码示例：单例模式的两种实现方法，并说明优缺点

```
public class Test{
    public Test(){
    }
    private static Test test = new 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;
}
}
```

第二种方式不需每次都创建一个类实例，而只是在第一次创建

8：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接口,让子类来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接口

9：代码示例：工厂模式的实现方法

```
public class Factory{
    public static Sample creator(int which){
        if (which==1){
            return new SampleA();
        }else if (which==2)
            return new SampleB();
        }
    }
}
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();
    }
}
```

10 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：就是把需要传递的多个值封装成一个对象一次性传过去

11：代码示例：值对象模式的实现方法

```
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;
    }
}
```

12：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：式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。

DAO 的本质就是一层屏蔽一种变化

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）

13：代码示例：DAO 模式的实现方法

```
public interface CustomerDAO {
public int insertCustomer(...);
```

public Collection selectCustomersVO(...);

}

14：什么是开放-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

15：什么是替换法则(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

16：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。

17：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

六、awt

1：什么是 GUI 中的容器？什么是 GUI 中的组件？

容器：是 Component 的一个抽象子类，它允许其他的组件被嵌套在里面，它可以容纳组件或其他容器

组件：AWT 提供的具有一定功能类，且带自己的界面，例如 MenuBar、Button 等

2：简述 AWT 中的 Frame、Panel 和组件的关系

组件直接添加在 Panel 上，而多個 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

3：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);
```

4：描述一个 GUI 界面的基本构成？

Frame，Panel，組件

Popmenu

6：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);
```

7：什么是布局管理器？

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

8：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

四：swing

1：什么是 JFC(Java 基础类)？

是关于 GUI 组件和服务完整集合

2：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。

3：什么是双缓冲？

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

4：描述 GUI 的事件机制

事件源：是一个事件得产生者，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。

5：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

6：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

7：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

8：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

9：如何学习 GUI 组件的用法？

主要学回組件的定义、取值、赋值的方法
类比学习

五、JEE

适用于创建服务器端的大型的软件服务系统

1. JEE ： JAVA PLATFORM ENTERPRISE DEDITON

２.是一个规范集、技术集、框架集（API 集）

一种技术对应一种或多种规范

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、提供一个经良的应用程序架构

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

３.C/S B/S 模式

a. C/S client/server 客户端也参与程序的运行与计算（富客户端、瘦客户端）

b. B/S border/server

4、企业级应用（大规模的应用）

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

5、业务逻辑

划分模块是 依照业务逻辑，所谓判定业务 就是起具有以下特征：

1.业务流程

2.业务判断

6、平台（角色）

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

7、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））

8、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

web 组件 ejb 组件

2.体系结构： 应用组件之间的关系

3.容器 ： 提供组件的运行环境，并对组件进行管理（一段程序）

管理组件的生命后期

不能单独运行必须运行在服务器上

程序通过上下文来调用容器（context）

容器通过回调的方法来调用组件

web 容器 ejb 容器

4.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）

9、部署描述 ：（用 xml 来描述）

组件向容器描述自己，使容器能够认识组件

容器通过回调的方法来调用组件

回调：由程序实现的方法，程序自己不调，由容器自动调用的方法

10、JAVA 关于 XML 的 API

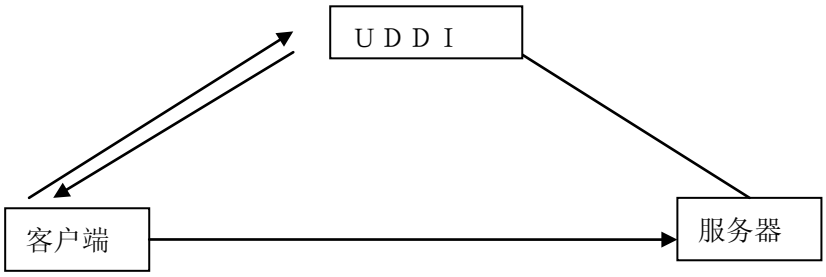
JAXP（Java API for xml prasing ）解析 xml 文件以便读写

JAXB (Java API for xml Binding) 映射技术

JAVA 数据映射到 xml 文件

11、web service ：提供 web 服务 （功能的重用 同样的功能只存在一份）

机制如下：



1．服务器注册 uddi，来展示自己的功能

2．客户端通过 uddi 来寻找自己想要的功能

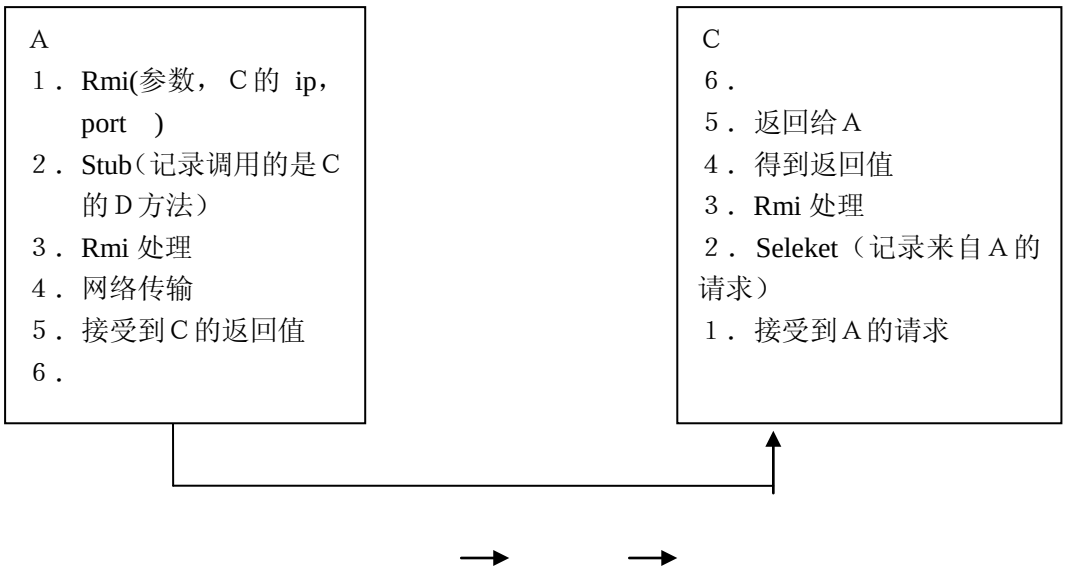
3．uddi 返回信息给客户端

4．客户端通过 web service 来调用服务器的方法，在调用时封装了ＲＰＣ、ＲＭＩ、ＪＮＤＩ（ＳＯＡＰ）

5． 调用是屏蔽了客户端与服务器端的差异，通过统一的 rmi 来进行通讯

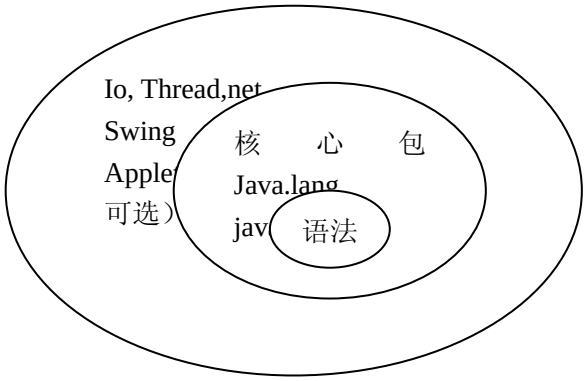
12、ＲＭＩ远程方法的调用机制

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



web　service　又出现不同的版本→集成技术→S O A

13、Ｊ ＡＶＡ的体系结构



14、学习方法

1.规范（熟记） 2.熟悉 API　3.多练　4.多想

六、servlet

1：什么是 B/S 和 C/S

Browser/Server 浏览器/服务器（瘦客户端）　Custom/Server 客户端/服务器（胖客户端）

2：描述 war 包、jar 包、ear 包的结构

war-
--myweb
---WEB-INF
----web.xml
----lib
----classes

jar-
--myjar
---META-INF
----.MF

ear-
--META-INF
---Application.xml
--.war
--.jar

3：什么是 servlet？servlet 主要做什么？

网络服务的请求和响应方式

通过 WEB 浏览器调用 servlet 的方法，主要是在服务端写动态代码与服务器端交流；

4：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

5：常用的 servlet 包的名称是？

javax.servlet

javax.servlet.http

6：描述 servlet 接口的层次结构？

Servlet

--genericServlet

--HttpServlet

--自己的 servlet

ServletRequest

ServletResponse

7：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，数据的内部应在 url 中可见；明文传输，安全度低

post 方法：请求的数据过程改变服务器的状态，Form 数据量大，数据的内部应在 url 中不可见，安全度高；

8：归类描述 HttpServletRequest 接口都完成那些功能

1.读取和写入 HTTP 头标

2.取得和设置 cookies

3.取得路径信息

4.标识 HTTP 会话。

9：归类描述 HttpServletResponse 接口都完成那些功能

HttpServletResponse 加入表示状态码、状态信息和响应头标的方法,它还负责对 URL 中写入一 Web 页面的 HTTP 会话 ID 进行解码。

10：描述 Service 方法所完成的基本功能？默认是在那里实现的？

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

HttpServlet 类实现

11：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 common 包-->lib 包-->servlet--->api.jar

2.开发 servlet 类

1)首先继承 HttpServlet

2)实现 doGet() doPost()

3)定义 doGet() doPost()

3.建个 web 应用

4.部署

安装 web 容器，例如 Tomcat

在 Tomcat 的 webapps 目录下新建一个文件夹作为 web 程序的根

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

12：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

13：Servlet 基本的描述应该是？请写出来

```
<servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>sl314.web.FormBasedHello</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/greeting</url-pattern>
</servlet-mapping>
```

14：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>
```

15：如何接受 request 中的参数

String userName = (String)request.getParameter("userName")方法

16：如何接受 request 中 header 的值

request.getHeader(name);

request.getHeaders(names);等

17：如何输出 html

```
PrintWriter pw = response.getWriter();
pw.write("<html>");
pw.write("Hello");
pw.write("</html>");
```

18：如何设置输出的 contentType

response.getContentType()

19：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

20：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 实例被销毁时有 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

21：什么是回调方法？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

22：如何设置初始化 servlet 的参数？

```
<init-param>
    <param-name>greetingText</param-name>
    <param-value>Hello</param-value>
</init-param>
```

23：如何获取 servlet 初始化的参数

```
public void init() {
  greetingText = getInitParameter("greetingText");
  System.out.println(">> greetingText = ''' + greetingText + '''');
}
```

24：ServletConfig 接口默认在那里实现的

GenericServlet 类实现 ServletConfig 接口

25：什么是 ServletContext？有什么作用？

Servlet 上下文

ServletContext 对象是 Web 应用的运行时表示，可通过其实现 Web 应用中的资源共享

26：如何访问 ServletContext 接口？是在那里实现的？

Servlet 通过 getServletContext()方法访问

GenericServlet 类实现

27：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)

28：如何设置 Context 的参数？

```
<context-param>
    <param-name>catalogFileName</param-name>
    <param-value>/WEB-INF/catalog.txt</param-value>
</context-param>
```

29：如何获取 Context 设置的参数值？

```
ServletContext context = sce.getServletContext();
String catalogFileName = context.getInitParameter(“catalogFileName”);
```

30：描述 Web 应用的生命周期？

Web 容器启动时，初始化每个 Web 应用

可以创建"监听器"对象触发这些事件；

Web 容器关闭时，销毁每个 Web 应用

31：如何用代码实现监控 Web 应用的生命周期？

```
public class Test implements ServletContextListener{
    public void contextInitialized(ServletContextEvent sce) {
        //
    }
}
<listener>
    <listener-class>
        com.csy.Test
    </listener-class>
</listener>
```

32：web 应用中如下错误码示什么意思：400，401，404，500

400 Bad Request

401 Unauthorized

404 Not Found

500 Internal Server Error

33：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

34：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)

35：什么是会话？

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

36：如何获得会话？

HttpSession session = request.getSesseion();

37：会话 Api 的基本功能？

getID() :String

isNew() :boolean

getAttribute(name):Object

setAttribute(name,value)

removeAttribute(name)

38：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()

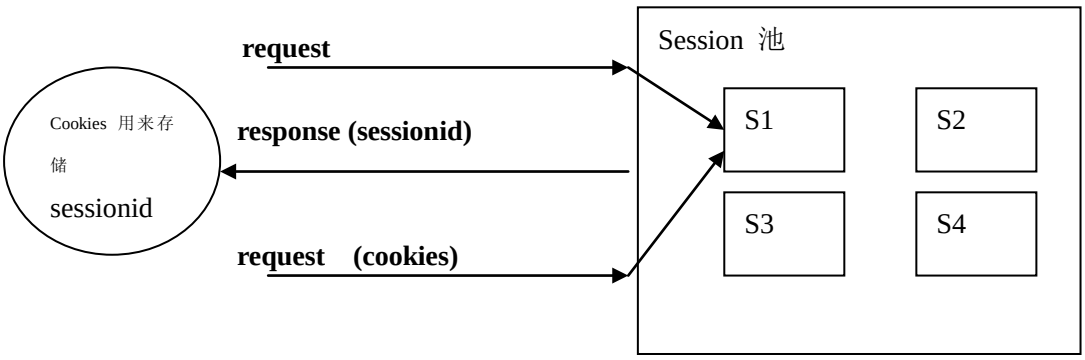
getCreationTime() :long

getLastAccessedTime() :long

getMaxInactiveInterval() :int

setMaxInactiveInterval(int)

39：描述会话保持状态的基本原理



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）

40：如何读写 cookie，代码示例

写：String name = request.getParameter("firstName");

```
Cookie c = new Cookie("yourname", name);
response.addCookie(c);
```

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

```
for ( int i=0; i < allCookies.length; i++ ) {
  if ( allCookies[i].getName().equals(“yourname”) ) {
    name = allCookies[i].getValue();
```

```
        }
    }
```

41：什么是 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

41：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

42：什么是授权，什么是验证？

授权是分局用户角色划分 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

43：什么是 HTTPS

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

44：什么是审计？

也就是访问跟踪，是为 web 应用的每次访问保留记录的过程

45：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

46：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

47：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）

48：支出上述六种那些是线程安全的

局部变量和请求属性

49：什么是 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

50：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

六、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

2：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

3：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>

4：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

*/

%>

5：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　%>

6：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; %>
```

7：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

8：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
Applicationn	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

9：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

10：描述 MVC 各部分的功能？

Model

封装应用状态
响应状态查询
暴露应用的功能

Controller

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

产生 HTML 响应
请求模型的更新
提供 HTML form 用于用户请求

11：什么是 Model 1 结构，以及结构中各部分的功能

Model1 中使用 jsp 来处理 web 应用中的视图控制部分

jsp+javabean

12：什么是 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。

13：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

14：什么是 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:userBean

jsp:setProperty

jsp:getProperty

jsp:parameter

jsp:include

jsp:forward

15：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" />
```

16：描述说明 Bean 的四种 scope

Request

Session

Application

Page

17：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

18：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

19：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：

20：什么是 Model 2 结构，以及结构中各部分的功能

jsp+javabean+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

21：如何获得分发器？分发器的功能？

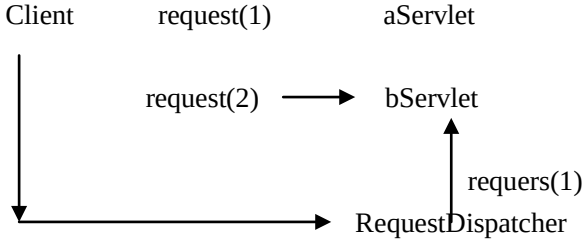
上下文对象中的分发器：

```
ServletContext context = 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



当从客户端发出请求到 a 是 request（1），a 再请求 b 是 request（2）就变成两个不同的请求，用 RequestDispatcher 来保持原来的请求
（在不同的 servlet 之间保持同一个请求时用 RequestDispatcher）

RequestDispatcher rd = request.getRequestDispatcher(“string”);
Rd.forword(request);//前往下一个请求 到下一个 servlet

22：描述 Jsp 中引入另外页面的两种方法？

<%@ include%>
<jsp:include>

23：比较上述两种方法的特点

<%@ include%> 再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> 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

24：什么是自定义标记

自定义标记库是一个 Web 组件，
包括：
一个标记库描述符文件
所有相关的标记处理器类

25：描述使用自定义标记的步骤，以及每步详细的工作

1.自定义标记遵循 XML 标记规则
自定义标记使用 XML 规则；
标记名属性，以及前缀都是大小写敏感；
标记嵌套规则；
2.需要在 JSP 页面和 Web 应用程序的部署描述符中声明标记库
taglib
在部署描述符中使用 taglib 元素声明 web 应用程序使用一个标记库；
在 jsp 页面中使用 taglib 指令表明要使用哪个标记库及相应的自定义标记所使用的前缀；
使用自定义的空标记
使用自定义的条件标记
使用自定义的迭代标记
3.在 JSP 页面中可使用自定义的空标记
getPepParam()标记:将命名的请求标志插入的输出中；
4.在 JSP 页面中使用自定义标记，可有条件地执行 HTML 响应的某部分
heading 标记：生成一个隐藏的 Html table 创建一个彩色且格式化过的页表头
5.在 JSP 页面中使用自定义标记，可迭代执行 HTML 响应中的某部分
IterateOverErrors 标记:对 stutas 中的所有异常现象进行迭代；
taglib
1.把类烤过去
2.把 Taglib 的描述文件(*.tld)放在 web-inf 根目录下
3.在 web.xml 里配置
4.回到页面去，定义 Taglib 使用的前缀
5.直接使用，一定要结尾；

七、Javascript 部分

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。

2：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。

3：Javascript 的运行环境

具备 javascript 运行器的

4：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>
```

5：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

6：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

可以用 var 声明

7：代码示例：Javascript 中的 if 控制语句的结构

```
if(i>4){
    alert(11);
}
```

8：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);
}
```

9：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。

10：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();
```

11：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;
```

}

12：如何触发 Javascript 的 function

function test(){

alert(11);

}

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

13：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

14：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();

15：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

16:：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;

}

17：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

18：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';

19：简要描述 Javascript 对象层次结构

window--document--组件

20：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

21：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>
```

22：框架如何载入页面

```
<frame name="left" src="test.htm">
```

23：如何从框架中的一个页面访问另一个页面

```
var value = parent.right.document.all("text1");
```

CSS 部分

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">；

2：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 读写示例

```
package com.javawebv.addresslist.baseinfo.valueobject;
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").i
                        tem(0).getFirstChild().getNodeValue());
  System.out.println("name   :"+student.getElementsByTagName("name").it
                        em(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();
          }
}
```

八、 Struts

一、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 来生成 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");

5、进入 execute 方法
5.1 收集参数
5.2 组织参数
5.3 调用，逻辑层
5.4 返回值
5.4.1 选择下一个页面 （actionforward）
5.4.2 把值传给下一个页面 （session）

九、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:eb"
        driverClassName="oracle.jdbc.driver.OracleDriver"
        username="sll"
        password="sll"
        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="sa"
        password=""
        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();
```

十、ORM 对象关系型数据库映射

(Object Relation Mapping)

一、一个对象对应一个表（可以是多对多的关系）

1.对象的属性的名称和表字段的名称可以不同

2.对象的属性的数量和表字段的数量可以不同

3.类型可以不同，但数据类型之间可以转换

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

cc

id=11

name=cc ———— > (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

十一、Hibernate

hibernate 开发流程

1、是什么？

hibernate 是一种基于 orm 的轻量级的框架

2、有什么？

1.对外提供操作数据库的接口

2.事务的处理

3.简化数据持久化的编程任务

3、能干什么？

1.orm

2.提供操作数据库的接口

4、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

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

hibernate 运行流程

一、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

二 、save

1.to--->po
2.根基 model 和 cfg.xml 中映射文件的注册来找到 hbm.xml 文件
3.根据 hbm.xml 文件中的 unsave-value 属性来判断是 save 还是 update
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

三、update、delete

1.根据 model 的 id 在内存 hibernate 的实例池中查找该对象
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

四、Query

4.1 load

1.根据 model 的类型来找到 hbm.xml 文件
2.判断 id 对应的对象是否存在
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

4.2 get

4.执行 sql 直接在数据库中查找，如果没有查到就会返回 null

4.3 query

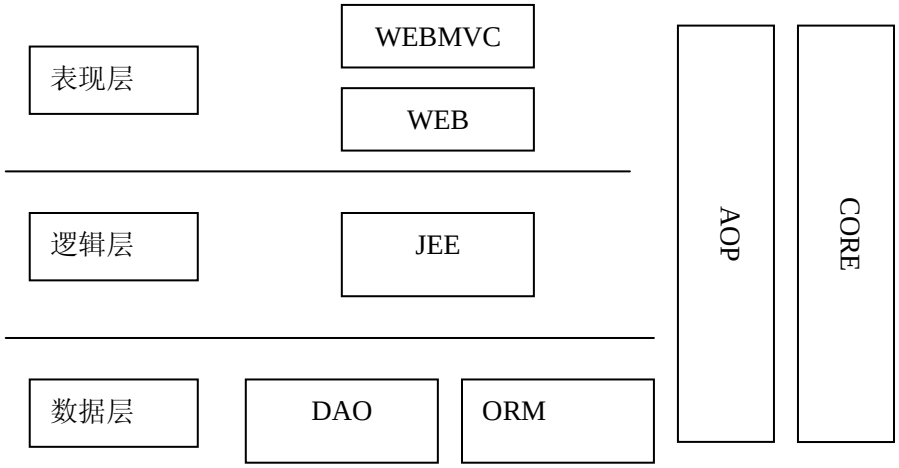
1.分析 hql 语句，得到 model 的类型
2.根据 moedel 和 cfg.xml 文件中映射文件的注册找到 hbm.xml 文件
3.根据 model 和 hbm.xml 文件来动态拼 sql
4.执行 sql 查询数据库
5.返回一个 resultset
6.循环 resultset 的值，放到 model 中在放到集合中

十二、spring

1．是什么？

Spring 是基于 JEE 的轻量级的应用框架

2．有什么？



每个包的功能：

WEBMVC：spring 本身提供的 web 框架

WEB：集成 web 应用的框架

JEE ：继承一系列的 jee 的技术

DAO：封装了 JDBC；

ORM：提供了对ＯＲＭ工具的集成

AOP ：面向切面编成

CORE：spring 的核心包，提供 bean 的公厂和 IOC 容器

3．能干什么？

把一列的 jee 的技术有效的组合在一起形成以良好的系统

4．容器和 bean

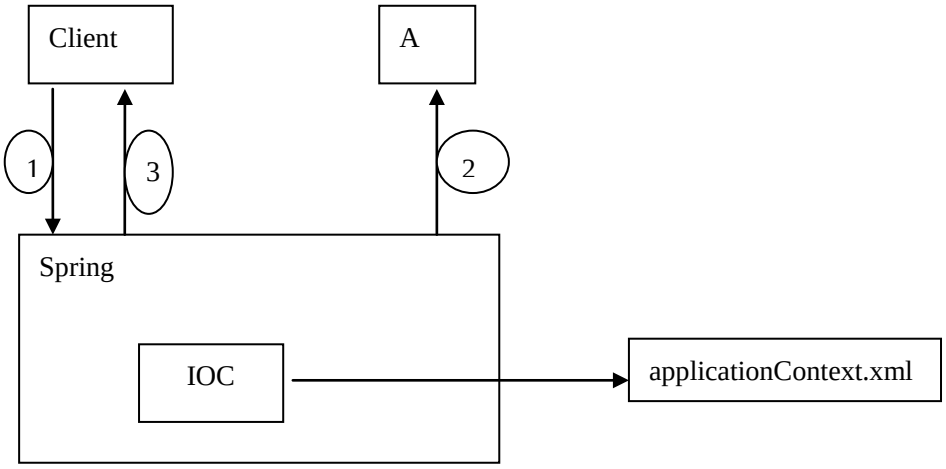
Bean ：是指受 spring 和 ioc 管理的对象称为 bean

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）

5．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

2．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

3．表现层调用逻辑层，可以让表现层作为客户端，而不要采用依赖注入。表现层只需要逻辑层接口一个资源

6.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=”dao” class=”daoImpl”>
        <property name=”cc” ref=”cc”>
</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

7、 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 targer)

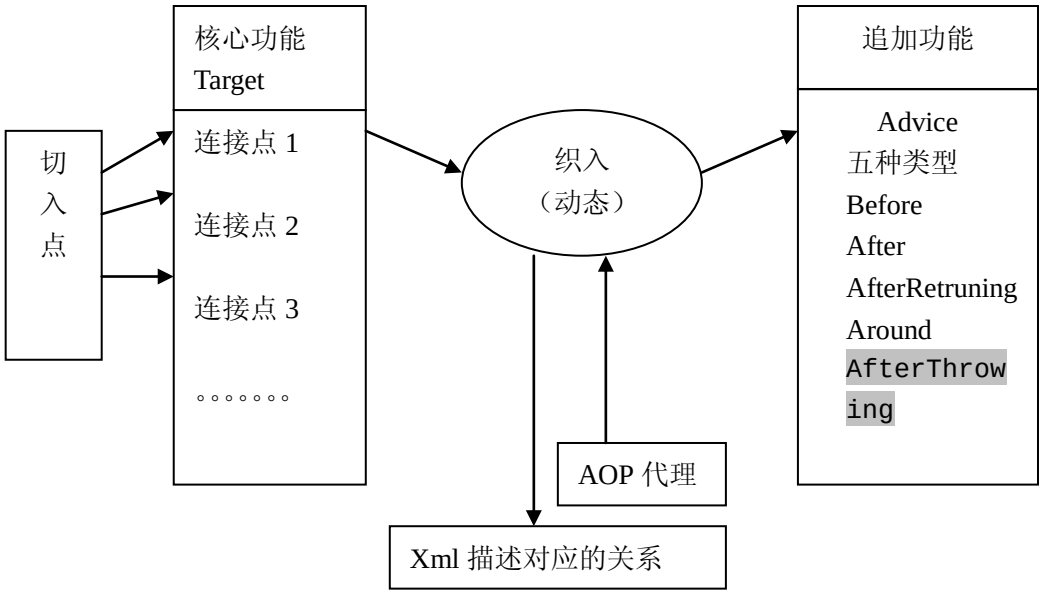
3.找到切入点

4.确定连接点

5.通过动态织入点 advice 和连接点对应起来

6.动态织入由 aop 代理来实现

7.xml 文件描述对应的关系



5.ascpet

5.1.在 dtd 中配置比较烦琐

所有的入口必须从一个代理（ProxyFactoryBean）开始

```
<bean id="proxy"
   class="org.springframework.aop.framework.ProxyFactoryBean">
   <property name="proxyInterfaces">
      <value>spring2test.aoptest.dtdv.Api</value>
   </property>
   <property name="target">
      <ref local="B"/>
   </property>
    <property name="proxyTargetClass">
          <value>true</value>
   </property>
    <property name="interceptorNames">
   <list>
      <value>staticPointcut</value>
      <value>staticPointcut2</value>
   </list>
   </property>
</bean>
```

5.2.配置切入点 （RegexpMethodPointcutAdvisor）

```
<bean id="staticPointcut2"
  class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
  <property name="advice">
     <ref local="myBeforeAdvice"/>
  </property>
```

```
 <property name="patterns">
  <list>
       <value>.*t.*</value>
  </list>
 </property>
</bean>
```

id 可以省去，由 spring 自动生成与类名相同的名字

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.javass.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.javass.spring.schemaaop.TestPointcut.t4()")

(2).

2.直接引用匿名的 pointcut

(1).@AfterReturning("execution(

* com.javass.spring.schemaaop.Api.test4())")

(2).@AfterReturning(pointcut=

"com.javass.spring.schemaaop.TestPointcut.t4() &&

args(str)", returning="retVal")

```
@AfterReturning
(pointcut="com.javass.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.javass.spring.schemaaop.TestPointcut.t1()")
(2).@Around("com.javass.spring.schemaaop.TestPointcut.t2() && args(str)")

2.直接引用匿名的 pointcut
(1).@Around("execution(* com.javass.spring.schemaaop.Api.test1())")
(2).@Around("execution(* com.javass.spring.schemaaop.Api.test2(..)) && args(str)")

```
//      @Around("com.javass.spring.schemaaop.TestPointcut.t2()  &&
        args(str)")
        @Around("execution(*
       com.javass.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.javass.spring.schemaaop.TestPointcut.t1()")
(2).@Before("com.javass.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.javass.spring.schemaaop.Api.test1())")
(2).@Before("execution(* com.javass.spring.schemaaop.Api.test2(..)) && args(str)")
注意 args 后的名称与参数名相同

```
//      @Before("com.javass.spring.schemaaop.TestPointcut.t2()  &&
        args(str)")
```

```
@Before("execution(* com.javass.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.javass.spring.schemaaop.TestPointcut.t1()")

2.直接引用匿名的 pointcut

(1).@After("execution(*
com.javass.spring.schemaaop.Api.test1())")

```
@After("com.javass.spring.schemaaop.TestPointcut.t1()")
 public void testAfter(){
 System.out.println("after1==    >pointcut");
 }
```

5.4.5.@AfterThorwing

6.@configurable

在配置文件中需开启 <aop:aspectj-configured/>

在根目录下将一个 META-INF 文件夹，在该文件夹下建一个 aop.xm

文件由 aop.xml 文件用来定义切入点

这样就可以在客户端直接 new 对象来直接操作（依赖注入是由 aspectj 来代替 ioc）

设置运行时的参数 run -->Arguments --->VMarguments

在这里并没有直接用 new 出来的对象，而是通过 aspectj 依赖注入一个具有值的对象来代替他使用

7.中间数据层访问 事务：JDBC JTA（分布式事务）

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:ccc"/>
    <property name="username" value="sll"/>
    <property name="password" value="sll"/>
```

```
    </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.DataSourceTransa
      ctionManager">
  <property name="dataSource" ref="dataSource"/>
</bean>
```

2.5 事务监控所有的方法

```
<tx:advice id="txAdvice" transaction-manager="txManager">
  <tx:attributes>
    <tx:method name="*"/>
  </tx:attributes>
</tx:advice>
```

2.6 定义切入点

```
<aop:config>
  <aop:pointcut id="my"
                expression="execution(*
                com.javass.spring.schemaaop.Api.*.*(..))"/>
  <aop:advisor advice-ref="txAdvice" pointcut-ref="my"/>
</aop:config>
```

2.7 在客户端通过 DataSourceUtils.getConnection(dataSource)来获得连接

2.8 我们自己不要关闭连接

2 编程式事务管理：

1.注解@Transcational

指哪打哪（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"/>
```

3. DAO 的支持

1.JDBCTemplate

1.1、直接用来执行 sql 语句

2、ORM 工具进行数据的访问

2.1、Hibernate 支持，集成 hibernate

把在 hibernate.cfg.xml 中的配置搬到 applicationcontext.xml 中

```
<bean id="mySessionFactory"
      class="org.springframework.orm.hibernate3.LocalSessionFactoryBe
      an">
      <property name="dataSource" ref="dataSource"/>(连接数据库)
      <property name="mappingResources">
      <list>
          <value>
              com/javass/spring/schemaaop/UserModel.hbm.xml
          </value>（资源注册）
```

```
        </list>
        </property>
        <property name="hibernateProperties">（可选配置）
            <value>
                hibernate.dialect=org.hibernate.dialect.Oracle9Dialect
                hibernate.show_sql=true
             </value>
          </property>
       </bean>
```

2.2、同样需要事务

```
    <bean id="txManager"
            class="org.springframework.orm.hibernate3.HibernateTransacti
            onManager">
            <property name="sessionFactory" ref="mySessionFactory"/>
     </bean>
    <aop:config>
        <aop:pointcut id="my"
                        expression="execution(*
                        com.javass.spring.schemaaop.Api.*.*(..))"/>
            <aop:advisor advice-ref="txAdvice" pointcut-ref="my"/>
    </aop:config>
    <tx:advice id="txAdvice" transaction-manager="txManager">
            <tx:attributes>
               <tx:method name="*"/>
            </tx:attributes>
    </tx:advice>
```

3.hibernate 模板和 DAO 支持都需要注入 sessionFactory

4.在客户端继承 HibernateDaoSupport 来得到 hibernate 模板

通过 hibernate 模板来操作数据对象

十三、事务

一、 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

二、 事务模型

1. 经典事务模型（JDBC） 事务不允许嵌套
2. 分布式事务　　两阶段提交协议和稳定性
3. ＪＴＡ　　ＪＴＳ

三、 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

十四、EJB

一、是什么？

ＥＪＢ是用来解决企业待发中出现的大量的问题的组件体系结构

二、有什么？

1.session Bean

1.1 无状态会话 bean　stateless sessionBean　SLSB

当客户端的请求到达容器时，，由容器在实例池中随便挑出一个实例来响应，不与客户端邦定

在这里优先选择 SLSB

1.2 有状态会话 bean　stateful　sessionBean　SFSB

Home 接口中可以有多个 create（）方法 可以有参数

Romote 接口 SLSB 和 SFSB 的写法相同

Bean 的特点

1.一定有属性来存储状态（于 session 一样与客户端相连）

2.一定有蒐咯 ejbCreate（）方法与 Home 接口中的 create 方法相对应

在以下情况下选择使用 SFSB

1.访问的数量少，访问的次数少

2.不会出现并发的情况下使用

2.entity Bean

2.1 bean 存储管理 Bean Manager Persistent BMP

2.2 容器存储管理 bean Container Manager Persistent CMP

3.消息驱动 bean MDB

3.1 没有 Home、Romote 接口

3.2 只有一个 onMessage（）方法

三、开发流程

开发流程 SessionBean

1、写一个接口继承 javax.ejb.EJBHome

该接口中只有一个 create 方法，返回类型为 remote 接口

抛出两个例外，CreateException,RemoteException

2、写一个接口继承 javax.ejb.EJBObject

给接口的所有方法都必须抛出 RemoteException

3、写一个 bean 类实现 SessionBean

必须实现以下五个方法

```
public void ejbCreate(){
   在实例创建后由容器回调的方法
}
public void ejbRemove(){
在实例被销毁后由容器回调的方法
}
public void ejbActivate(){}
public void ejbPassivate(){}
public void setSessionContext(SesssionCOntext ctx){
  this.ctx = ctx;
}
```

实现 remote 接口中的方法，注意不需要抛出 RemoteException

4、写一个类作为客户端

1.设置具体要访问那一个服务端和端口，和要访问服务端的 Jndi 工厂

```
System.setProperty(Context.PROVIDER_URL,"jnp://127.0.0.1:1099");
System.setProperty(Context.INITIAL_CONTEXT_FACTORY,"org.jnp.interfaces.N
amingContextFactory");
```

2.

```
InitialContext context = new InitialContext();
```

3.

```
Object obj=context.lookup("java:slltest");
```

4.

```
MyEjbHome myejb =
(MyEjbHome)PortableRemoteObject.narrow(obj,MyEjbHome.class);
```

5.

```
MyEjb my=myejb.create();
```

6..调用逻辑层的方法

```
String s =my.add("ccccccccccccc");
```

5、配置文件

1.ejb-jar.xml

```
<ejb-jar>
    <enterprise-beans>
     <session>
          <ejb-name>sll</ejb-name>
          <home>com.javass.ejb03.test.MyEjbHome</home>
          <remote>com.javass.ejb03.test.MyEjb</remote>
          <ejb-class>com.javass.ejb03.test.MyEjbSession</ejb-class>
          <session-type>Stateless</session-type>
          <transaction-type>Container</transaction-type>
     </session>
    </enterprise-beans>
</ejb-jar>
```

2.jboss.xml

```
<jboss>
    <enterprise-beans>
        <session>
            <ejb-name>sll</ejb-name>
            <jndi-name>slltest</jndi-name>
        </session>
    </enterprise-beans>
</jboss>
```

6、jar 包

1.随便拷贝一个 jar 包，只留下 MANIFEST.MF 文件中的头的部分

2.将自己的配置文件放在 META-INF 文件夹下

3.将.class 文件放到根目录下（与 META-INF 平级）

7、发布

将 jar 包放到 jboss-->server-->default-->deploy 下

8、测试

开发流程实体 bean==EJB+ORM

四个类，三个配置文件

1、Home 接口

1.写一个接口继承 ejbhome

2.create()方法；可以写多个 create(String id,String name)（存储状态）

3.findPrimaryKey(String key)

a.返回 remote 接口或者是 romote 接口的集合

b.抛出 FindException

c.所有的 finder..都由 EJBQL 实现

2、Remote 接口

1.写一个接口继承 EJBObject

2.所有的方法都抛出 RemoteException

直接写 set、get 方法，不用写属性，容器会自动判断

3、PK 类（可以没有 PK 类，这种情况下设置）

1.写一个类实现可序列化
2.私有的属性
3.相应的 set、get 方法

4、EntityBean 类（abstract）

1.写一个类实现 EntityBean

2.实现 9 个方法

a.

```
private EntityContext cxt ;
public void setEntityContext(EntityContext arg0) throws EJBException,
RemoteException{cxt=arg0;}
public void unsetEntityContext() throws EJBException, RemoteException
        {cxt=null;}
public void ejbRemove() throws RemoveException, EJBException,
RemoteException {
//delete
}
public void ejbStore() throws EJBException, RemoteException {

//insert
}
public void ejbActivate() throws EJBException, RemoteException {}
public void ejbPassivate() throws EJBException, RemoteException {}
public void ejbLoad() throws EJBException, RemoteException {
  //select
}
public String ejbCreate(String id,String name)throws CreateException{
//update
   this.setId(id);
   this.setName(name);
   return id;
}
public void ejbPostCreate(String id,String name){
//调用 ejbcreate 方法后紧接着调用的方法
}
```

b. 实现自己的业务方法

5、配置文

1.ejb-jar.xml

```
<entity>
  <ejb-name>MyEntity</ejb-name>
  <home>ejbtest.MyEntityHome</home>
  <remote>ejbtest.MyEntity</remote>
  <ejb-class>ejbtest.MyEntityBean</ejb-class>
```

```
<persistence-type>Container</persistence-type>
<prim-key-class>java.lang.String</prim-key-class>//如果没有 PK 类，设置为
 String 要写全路径
<reentrant>False</reentrant>
<cmp-version>2.x</cmp-version>//版本号是必须要配置的
<abstract-schema-name>ccentity</abstract-schema-name>
//对象的属性的名称（不一定和数据库一一对应）
<cmp-field>
 <field-name>id</field-name>
</cmp-field>
<cmp-field>
 <field-name>name</field-name>
</cmp-field>
<primkey-field>id</primkey-field>//设置主键
</entity>
```

2.jboss.xml

```
<entity>
  <ejb-name>MyEntity</ejb-name>
  <jndi-name>cctest3</jndi-name>
</entity>
```

3.jbosscmp-jdbc.xml

配置数据源

```
<jbosscmp-jdbc>
  <defaults>
      <datasource>java:/sll</datasource>
      <datasource-mapping>MS SQLSERVER2000</datasource-mapping>
      <datasource-mapping>Oracle9i</datasource-mapping>
      <create-table>false</create-table>
  </defaults>
  <enterprise-beans>
      <entity>
          <ejb-name>MyEntity</ejb-name>
              <table-name>tbl_test</table-name>
          对象的属性对应数据库的字段
          <cmp-field>
              <field-name>id</field-name>
              <column-name>uuid</column-name>
          </cmp-field>
          <cmp-field>
              <field-name>name</field-name>
              <column-name>link</column-name>
          </cmp-field>
      </entity>
  </enterprise-beans>
```

```
</jbosscmp-jdbc>
```

4.配置数据源文件

4.1 oracle-ds.xml

```
<datasources>
    <local-tx-datasource>
      <jndi-name>sll</jndi-name>
      <connection-url>jdbc:oracle:thin:@127.0.0.1:1521:eb</connection-url>
      <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
      <user-name>sll</user-name>
      <password>sll</password>

    <exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.OracleExc
    eptionSorter</exception-sorter-class-name>
   </local-tx-datasource>
</datasources>
```

4.2 mssql-ds.xml

```
<datasources>
   <local-tx-datasource>
      <jndi-name>jdbc/sll</jndi-name>
      <connection-url>jdbc:jtds:sqlserver://localhost:1433;SelectMethod=cursor;Data
      baseName=sll</connection-url>
      <driver-class>net.sourceforge.jtds.jdbc.Driver</driver-class>
      <user-name>sa</user-name>
      <password></password>
   </local-tx-datasource>
</datasources>
```

EJB 的运行流程

一、客户端

1．设置系统参数

```
//1.设置具体要访问那一个服务端和端口，和要访问服务端的 Jndi 工厂
System.setProperty(Context.PROVIDER_URL, "jnp://127.0.0.1:1099");
System.setProperty(Context.INITIAL_CONTEXT_FACTORY,
        "org.jnp.interfaces.NamingContextFactory");
```

2．初始化上下文

```
InitialContext context = new InitialContext();
```

3．查找 JNDI

```
Object obj = context.lookup("java:slltest");
```

通过 rmi 和 jndi 在容器中查找到 Home 接口，由容器自动生成 Home 接口的实现类，再通过 rmi 和 jndi 返回给客户端

4.将 obj 转换成 Home 接口

```
MyEjbHome myejb = (MyEjbHome) PortableRemoteObject.narrow(obj,
        MyEjbHome.class);
```

5.通过 Hemo 接口来得到 Remote 接口

yEjb myRemote1 = myejb.create();

通过 rmi 和 jndi 在容器中查找到 Home 接口，由容器自动生成 Home 接口的实现类，调用 Home 的 create 来创建 Remote 接口，由容器自动生成 Romote 的实现类将得到的 Romote 实例通过 rmi 和 jndi 返回给客户端

6.用自己的商务方法

同上

在容器中的 bean 的实例池中得到实例返回给客户端

String s01 = myRemote1.add("ccccccccccccc");

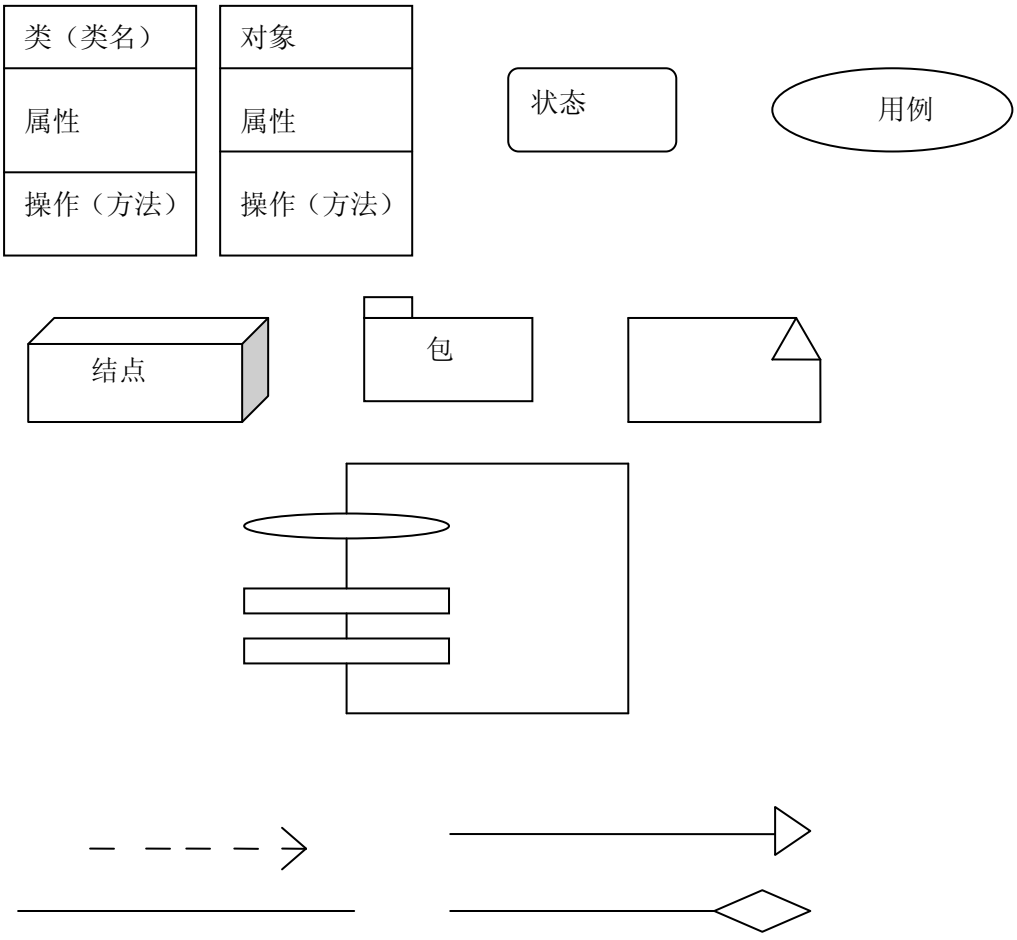
十五、UML

一、 是什么？

一种标准的图形化的建模语言

二、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



组合图形 ： 组件
卷边矩形 ： 笔记（注释）
虚箭头 ： 依赖
三角箭头 ： 通用化（继承）
菱形 ： 聚合
直线 ： 关联

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：

一、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

二、状态图的要素

1. 状态
2. 转移
3. 转移的条件

三、活动图的要素

1. 活动
2. 条件
3. 变迁（转移）

四、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；

十六、开发工具、

一、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跟目录下

二、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

三、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

四、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

3. 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

4. 生命周期

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

5. TestCase 套件

```
Public class MyTest{
   Public static Test suite(){
      TestSuite suite = new TestSuite();
      Suite.addTestCase(Test1.class);
      Suite.addTestCase(Test2.class);
   }

}
```

十七、SQL

一、SQL 分类：

DDL—数据定义语言(CREATE，ALTER，DROP，DECLARE)
DML—数据操纵语言(SELECT，DELETE，UPDATE，INSERT)

DCL—数据控制语言(GRANT，REVOKE，COMMIT，ROLLBACK)

二、基本语法

1、创建数据库

create database database-name

2、删除数据库

drop database dbname

3、备份 sql server

--- 创建 备份数据的 device

USE master

EXEC sp_addumpdevice 'disk', 'testBack',

'c:\mssql7backup\MyNwind_1.dat'

--- 开始 备份

BACKUP DATABASE pubs TO testBack

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 definition only

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)

8、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。

9、创建视图

create view viewname as select statement

删除视图：

drop view viewname

10、 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

11、 **使用外连接**

A、left outer 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 OUT JOIN b ON a.a =b.c

B：right outer 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 outer 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

12、 in 的使用方法

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

13、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)