**1、**用记事本编写java程序时，生成的class文件会乱码。java源文件的编码格式为UTF-8，编译器默认把它当做GB2312的编译格式编译成class文件，所以会出现乱码。

**2、**如果无法生成class文件，那是在环境变量中，CLASSPATH里的路径没有写好。

**3、**IDE（集成开发环境 integrated development environment）

**4、**JAVA中有两类数据类型：

基本数据类型（数值型（整数类型、浮点类型）、字符型、布尔型）

引用数据类型（类（class）、接口（interface）、数组）

基本数据类型是存数据本身

引用数据类型是存数据的空间地址

**5、**Java运算符分为：算数运算符，赋值运算符，比较运算符，逻辑运算符，条件运算符

**6、**数组排序 定义import java.util.Arrays;

对数组进行排序 Arrays.sort(数组名）；

对字符串进行排序 Arrays.toString(数组名）；

一维数组 int[] num = new int[7];

二位数组 int[][] num= new int[7][8];

int[][] num={{},{}};

**方法：**

定义一个方法的语法是：

访问修饰符 返回值类型 方法名（参数列表）

{

方法体

}

根据方法是否带参、是否带返回值，可将方法分为四类：

无参无返回值、无参带返回值、带参无返回值、带参带返回值。

**重载：**

判断方法重载的依据：

1、必须是在同一个类中

2、方法名相同

3、方法参数的个数、顺序或类型不同

4、与方法的修饰符或返回值没有关系

**类**

类是模子，确定对象将会拥有的特征（属性）和行为（方法）

类的特点

类是对象的类型

具有相同属性和方法的一组对象的集合成员变量和局部变量

1.成员变量

在类中定义，用来描述对象将要有什么

2.局部变量

在类的方法中定义，在方法中临时保存数据。

成员变量和局部变量区别

1.作用域不同

局部变量的作用仅限于定义它的方法

成员变量的作用域在整个类内部都是可见的

2.初始值不同

java会给成员变量一个初始值

Java不会给局部变量赋予初始值

3.在同一个方法中，不允许有同名局部变量；

在不同的方法中，可以有同名局部变量；

4.两类变量同名时，局部变量具有更高的优先级

**构造方法**

1.使用new+构造方法 创建一个新的对象

2.构造方法时定义在java类中的一个用来初始化对象的方法

构造方法与类同名且没有返回值

Java中的static使用之静态方法

与静态变量一样，我们也可以使用static修饰方法，称谓静态方法或类方法。

**使用static关键字声明静态方法**

public static void print（）

{}

直接使用类名调用静态方法 HelloWorld.print（）；

使用对象名调用。 HelloWorld hello = new HelloWorld();

hello.print();

1.静态方法中可以直接调用同类中的静态成员，但不能直接调用非静态成员

如果希望在静态方法中调用非静态变量，可以通过创建类的对象，然后通过对象来访问非静态变量。

2.在普通成员方法中，则可以直接访问同类的非静态变量和静态变量

3.静态方法中不能直接调用非静态方法，需要通过对象来访问非静态方法。

Java中的static使用之静态初始化块

静态初始化块只在类加载时执行，且只会执行一次，同时静态初始化块只能

给静态变量赋值，不能初始化普通的成员变量。

程序运行时静态初始化块最先被执行，然后执行普通初始化块，最后才执行构造方法。

由于静态初始化块只在类加载时执行一次，所以当再次创建对象hello2时并未

执行静态初始化块。

**封装**

概念：将类的某些信息隐藏在类的内部，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏新消息的操作和访问。

一个标准的面向对象的程序不能直接操作类的属性信息，是要通过方法间接的操作。

好处：

1. 只能通过规定的方法访问数据
2. 隐藏类的实例细节，方便修改和实现。

封装的实现步骤

1. 修改属性的可见性 设为private
2. 创建getter/setter方法 用于属性的读写
3. 在getter/setter方法中加入属性控制语句 对属性值的合法性进行判断。

**包**

系统中的包

1. java.（功能）.（类）
2. java.lang.(类)包含java语言基础的类
3. java.util.(类）包含java语言中各种工具类
4. java.io.(类)包含输入、输出相关功能的类

**this关键字**

1. this关键字代表当前对象

this.属性 操作当前对象的属性

this.方法 调用当前对象的方法

1. 封装对象的属性的时候，经常会使用this关键字

**Java中的内部类**

内部类可分为：成员内部类、静态内部类、方法内部类、匿名内部类。

内部类的主要作用：

1. 内部类提供了更好的封装，可以把内部类隐藏在外部类之内，不允许同一个包中的其他类访问该类。
2. 内部类的方法可以直接访问外部类的所有数据，包括私有的数据
3. 内部类所实现的功能使用外部类同样可以实现，只是有时使用内部类更方便。

**Java中的静态内部类**

1. 静态内部类不能直接访问外部类的非静态成员，但可以通过new 外部类（）.成员 的方式访问。
2. 如果外部类的静态成员与内部类的成员名称相同，可通过“类名.静态成员”访问外部类的静态成员；如果外部类的静态成员与内部类的成员名称不相同，则可用过“成员名”直接调用外部类的静态成员
3. 创建静态内部类的对象时，不需要外部类的对象，可以直接创建内部类 对象名 = new 内部类（）；

**Java中的方法内部类**

1. 方法内部类就是内部类定义在外部类的方法中，方法内部类只在该方法的内部可见，即只在该方法内可以使用。
2. 由于方法内部类不能在外部类的方法以外的地方使用，因此方法内部类不能使用访问控制符和static修饰符。

**继承**

继承是类与类的一种关系。

继承的好处：

子类拥有父类的所有属性和方法。

实现代码复用

**方法重写**

1.什么是方法的重写：

如果子类对继承父类的方法不满意，是可以重写父类继承的方法的，当调用方法时会优先调用子类的方法。

1. 语法规则：
2. 返回值类型
3. 方法名
4. 参数类型及个数

都要与父类继承的方法相同，才叫方法的重写。

**继承的初始化顺序**

1. 初始化父类再初始子类。
2. 先执行初始化对象中属性，再执行构造方法中的初始化。

final关键字

使用final关键字做标识有“最终的”含义

final可以修饰类、方法、属性和变量

final修饰类，则该类不允许被继承

final修饰方法，则该方法不允许被覆盖（重写）

final修饰属性

则该类的属性不会进行隐式的初始化（类的初始化属性必须有值）或在构造方法中赋值（但只能选其一）

final修饰变量，则该变量的值只能赋一次值，即变为常量。

super的应用

Super关键字

在对象的内部会用，可以代表父类对象。

Super的应用

子类的构造的过程中必须调用其他父类的构造方法

如果子类的构造方法中没有显示调用父类的构造方法，则系统默认调用父类无参的构造方法。

如果显示的调用构造方法，必须在子类的构造方法的第一行。

如果子类构造方法中既没有显示调用父类的构造方法，而父类有没有无参的构造方法， 则编译出错。

**Object类**

Object类是所有类的父类，如果一个类没有使用extends关键字明确表示继承另外一个类，那么这个类默认继承object类。

Object类中的方法，适合所有子类。

1. toString（）方法

在object类里面定义toString（）方法的时候返回的对象的哈希code码（对象地址字符串）

可以通过重写tostring（）方法表示出对象的属性

1. equals()方法

比较的是对象的**引用**是否指向同一块内存地址

Dog dog = new Dog()

一般情况下比较两个对象时比较他的值是否一致，所以要进行重写。

**多态**

对象的多种形态

1. 引用多态

父类的引用可以指向本类的对象

父类的引用可以指向子类的对象

1. 方法多态

创建本类对象时，调用的方法为本类方法

创建子类对象时，调用的方法为子类重写的方法或者继承的方法

继承是多态的实现基础

多态中引用类型转换

1. 向上类型转换（隐式/自动类型转换），是小类型到大类型的转换（无风险）
2. 向下类型转化（强制类型转换），是大类型到小类型（有风险）
3. instanceof运算符，来解决引用对象的类型，避免类型转换的安全性问题。

抽象类

1. 语法定义：

抽象类前使用abstract关键字修饰，则该类为抽象类

1. 应用场景：
2. 在某些情况下，某个父类只是知道其子类应该包含怎样的方法，但无法准确知道这些子类如何实现这些方法。

（抽象类是约束子类必须有哪些方法而并不关注子类如何去实现）

1. 从多个具有相同特征的类中抽象出一个抽象类，以这个抽象类作为子类的模板，从而避免了子类设计的随意性。
2. 作用：

限制规定子类必须实现某些方法，但不关注实现细节。

1. 使用规则
2. abstract定义抽象类
3. abstract定义抽象方法，只有声明，不需要实现
4. 包含抽象方法的类是抽象类
5. 抽象类中可以包含普通的方法，也可以没有抽象方法。
6. 抽象类不能直接创建，可以定义引用变量。

抽象方法没有方法体以分号结束

**接口**

1. 接口概念：

接口可以理解为一种特殊的类，由全局常量和公共的抽象方法组成

类是一种具体实现体，而接口定义了某一批类所需要遵守的规范，接口不关心这些类的内部数据，也不关心这些类里方法的实现细节，它只规定这些类里必须提供某些方法。

1. 接口定义

和类定义不同，定义接口不再使用class关键字，而是使用interface关键字

[修饰符]interface 接口名[extends 父接口1，父接口2..]

{

零个到多个常量定义……

零个到多个抽象方法的定义……

}

接口就是用来被继承、被实现的、修饰符一般建议用public

注意：不能使用private和protected修饰接口.

**接口中方法不能有方法体。**

1. 接口定义：

常量：

接口中的属性是常量，即使定义时不添加

Public static final 修饰符，系统也会自动加上

方法：

接口中的方法只能是抽象方法，总是使用，即使定义时不添加public abstract修饰符，系统也会自动加上。

1. 使用接口

一个类可以实现一个或多个接口，实现接口使用implements关键字。Java中一个类只能继承一个父类，是不够灵活的，通过实现多个接口可以做补充。

继承父类实现接口的语法为：

[修饰符]class 类名 extends 父类 implements 接口1，接口2……

{

类体部分//如果继承了抽象类，需要实现继承的抽象方法；要实现接口中的抽象方法。

}

如果要继承父类，继承父类必须在实现接口之前。

使用接口：

接口在使用过程当中，还经常与匿名内部类配合使用

匿名内部类就是没有名字的内部类，

多用于关注实现而不关注实现类的名称

语法格式：

1、Interface i = new Interface() **\*（不常用）**

{

pulic void method(){

System.out.println(“匿名内部类实现接口的方式”);

}

};

i.接口名();

2、new Interface()  **\*（常用）**

{

Public void method(){

System.out.println(“匿名内部类实现接口的方式”);

}.接口名（）;

**异常**

异常简介：

有异于常态，和正常情况不一样，有错误出现。阻止当前方法或作用域，称之为异常

处理异常

try-catch以及try-catch-finall

抛出异常

自定义异常

异常链

Throwable

{Error Exception}

Exception {RE 是非检查异常 其他的为检查异常}

try{

//一些会抛出的异常的方法

catch(Exception e){

//处理该异常的代码

}

finally

{最终将要执行的代码

}

}

**Java中的异常抛出**

throw 将产生的异常抛出

throws 声明将要抛出何种类型的异常

自定义异常

class自定义异常类 extends异常类型

**Java中的异常链**

printStackTrace 在命令行打印异常信息在程序中出错的位置及原因

initCause 对异常进行包装，目的为了出问题能追根究底

**Java中的String类的常用方法**

int length() 返回当前字符串的长度

int indexOf(int ch) 查找ch字符在该字符串中第一次出现的位置

int indexOf(String str) 查找str子字符串在该字符串中第一次出现的位置

int lastIndexOf(int ch) 查找ch字符在该字符串中最后一次出现的位置

int lastIndexOf(String str)查找str子字符串在该字符串中第一次出现的位置

String substring(int beginIndex) 获取从beginIndex位置开始到结束的子字符串

String substring(int beginIndex,int endIndex) 获取从beginIndex位置开始到endindex位置的子字符串

String trim（） 返回去除了前后空格的字符串

boolean equals（Object obj） 将该字符串与指定对象比较，返回true或false

String toLowerCase() 将字符串转换为小写

String toUpperCase() 将字符串转换为大写

Char charAt(int index) 获取字符串中指定位置的字符

String[] split(String regex, int limit)将字符串分割为子字符串，返回字符串数组

byte[] getBytes() 将该字符串转换为byte数组

StringBuilder append(参数) 追加内容到当前StringBuilder对象的末尾

StringBuilder insert(位置，参数) 将内容插入到StringBuilder对象的指定位置

String toString() 将StringBuilder对象转换为String对象

int length() 获取字符串的长度

Integer(int value) 创建一个Integer对象，表示指定的int值

Integer（String s） 创建一个integer对象，表示String参数所指示的int值

byteValue() intValue() longValue() parseInt(String s)

doubleValue() floatValue() toString() valueOf(String s)

**Java中的集合框架**

集合的概念

是一种工具类，就像容器，储存任意数量的具有共同属性的对象。

集合的作用

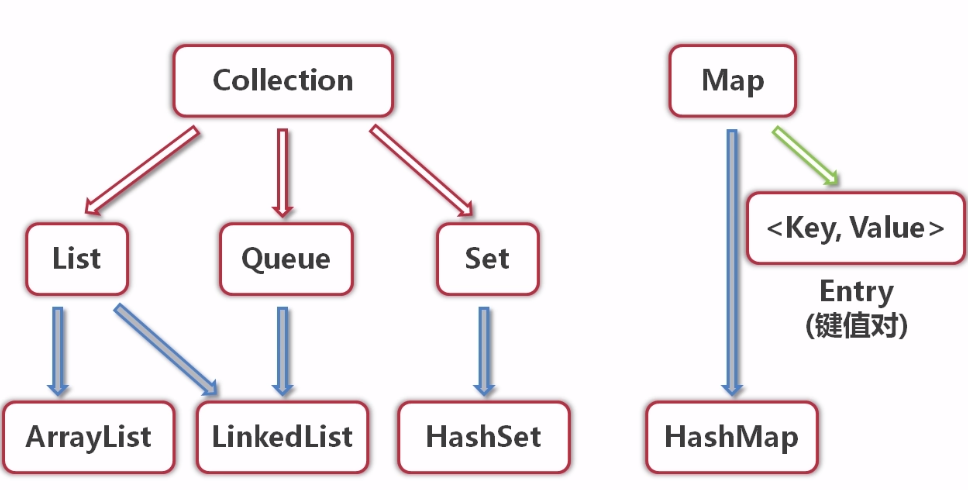
在类的内部，对数据进行组织。

简单而快速的搜索大数量的条目。

有的集合接口，提供了一系列排序有序的元素，并且可以在序列中间快速的插入或者删除相关的元素。

有的集合接口，提供了映射关系，可以通过关键字（key）去快速查找到对应的唯一对象，而这个关键字可以是任意类型。

Java集合框架体系



List接口及其实现类——Arraylist

List是元素有序并且可以重复的集合，被称为序列

List可以精确的控制每个元素的插入位置，或删除某个位置的元素

Arraylist——数组序列，是List的一个重要实现类

Arraylist底层是由数组实现的

使用ArrayList类

如果经常需要根据索引位置访问集合中的对象，使用由ArrayList类实现对List集合的效率较好。数组结构的缺点是向索引位置插入对象和删除指定所以对象的速度较慢。

使用LinkedList类

LinkedList类实现了List接口，由LinkedList类实现的List集合采用链表结构保存对象。链表结构的有点是便于向集合中插入和删除对象。

**泛型**

**集合中的元素，可以是任意类型的对象（对象的引用）**

如果把某个对象放入集合，则会忽略他的类型，

而把他当做Object处理

**泛型则是规定了某个集合只可以存放特定类型的对象**

会在编译期间进行类型检查

可以直接按指定类型获取集合元素

1. 泛型集合中的限定类型不能使用基本数据类型。
2. 可以通过使用包装类限定允许存入的基本数据类型。

int- Integer long-Long boolean-Boolean

**Set接口及其实现类——HashSet**

Set是元素无序并且不可以重复的集合，被称为集

HashSet——哈希集，是Set的一个重要实现类

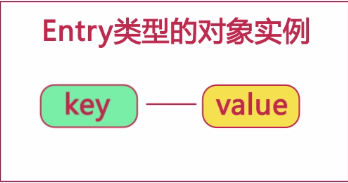
由HashSet类实现的Set集合的有点是能够快速定位集合中的元素

**Map和HashMap**

Map接口

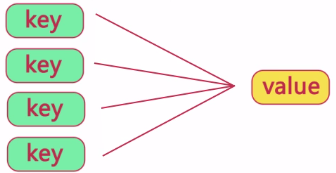
Map提供了一种映射关系，其中的元素是以键值对（key-value）的形式存储的，能够实现根据key快速查找value

Map中的键值对以Entry类型的对象实例形式存在



键（key值）不可重复，value值可以

每个键最多只能映射到一个值



Map接口提供了分别返回key值集合、value集合以及Entry（键值对）集合的方法

Map支持泛型，形式如：Map<K,V>

**HashMap类**

HashMap是Map的一个重要实现类，也是最常用的，基于哈希表实现

HashMap中的Entry对象是无序排列的

Key值和value值都可以为null，但是一个HashMap只能有一个key值为null的映射（key值不可重复）

HashMap实现的Map接口，HashMap的映射是无序的。

ketSet返回的是所有的键的集合

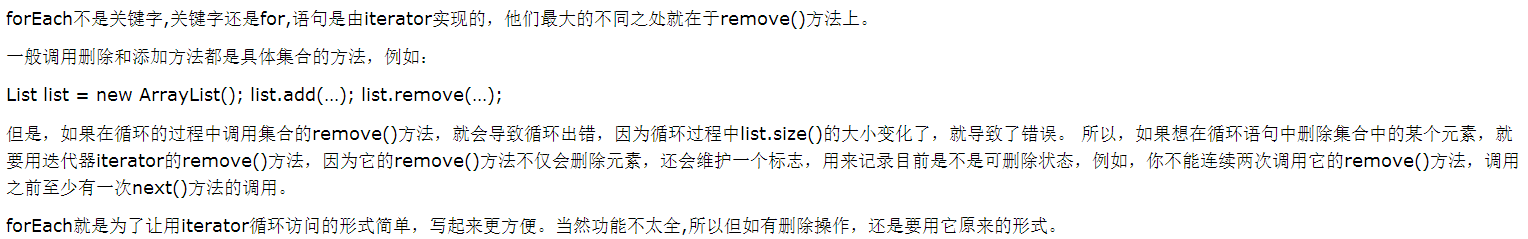
EntrySet返回的是所有的键值对的集合

**Collections工具类**

Comparable——默认的比较规则

Comparator——临时的比较规则

forEach 和 Iterator的区别



**Java I/O输入输出流**

java.io.File类用于表示文件（目录）

File类只用于表示文件（目录）的信息（名称、大小等），不能用于文件内容的访问。

编码问题

File类的使用

RandomAccessFile的使用

字节流的使用

字符流的使用

对象的序列化和反序列化

Java.io.File类用于表示文件（目录）

File类只用于文件（目录）的信息（名称。大小等），不能用于文件内容的访问

FileReader类是Reader类的子类，它实现了从文件中读取出字符数据，是文件字符输入流。

FileWriter类是Writer类的子类，它实现了将字符数据写入文件中，是文件字符输出流。

RandomAccessFile Java提供的对文件内容的访问，既可以读文件，也可以写文件，

RandomAccessFile支持随机访问文件，可以访问文件的任意位置

1. Java文件模型

在硬盘上的文件时byte byte byte存储的，是数据的集合

1. 打开文件

有两种模式“rw”（读写）“r”（只读）

RandomAccessFile raf = new RandomAccessFile(file,”rw”)

文件指针，打开文件时指针在开头pointer=0;

1. 写方法

raf.write(int)-->只写一个字节（后8位）,同时指针指向下一个文职，准备再次写入

1. 读方法

int b = raf.read()-->只读一个字节

1. 文件读写完成以后一定要关闭（Oracle官方说明）

**IO流（输入流、输出流）**

流是一组有序的数据序列。根据类型操作的类型，分为输入流和输出流。输入流的指向称为源，程序从指向的输入流中读取源中的数据。输出流的指向是字节要去的目的地。

字节流、字符流

1. 字节流
2. InputStream、OutputStream

InputStream抽象了应用程序读取数据的方式

OutputStream抽象了应用程序写出数据的方式

1. EOF=End 读到-1就读到结尾
2. 输入流基本方法

int b = in.read();读取一个字节无符号填充到int低八位、-1是EOF in.read（byte[] buf）读取数据填充到字节数组buf

In.read(byte[] buf,int start,int size)

1. 输出流基本方法

out.write(int b)写出一个byte到流，b的低8位

out.write(byte[] buf)将buf字节数组都写入到流

out.write(byte[] buf,int start, int size)

1. FileInputStream---> 具体实现了文件上读取数据
2. OutputStreamWrite是字节流通向字符流的桥梁。写出字节，并根据指定的编码方式，将之转换为字符流。
3. InputStreamReader是字节流通向字符流的桥梁。它可以根据指定的编码方式，将字节输入流转换为字符输入流。
4. FileOutputStream类是OutputStream类的子类。它实现了文件的写入，能够以字节形式写入文件中，该类的所有方法都是从OutputStream类继承并重写的。
5. FileInputStream类是InputStream类的子类。它实现了文件的读取，是文件字节输入流。该类使用于比较简单的文件读取，其所有方法都是从InputStream类继承并重写的。
6. file:File文件类型的实例对象。在file后面，加true会对原有内容进行追加，不加true会将原有内容覆盖。
7. BufferedReader类是Reader类的子类，使用该类可以以行为单位读取数据。
8. BufferedWriter类是Writer类的子类，使用该类可以以行为单位写入数据。
9. PrintStream是打印输出流，它可以直接输出各种类型的数据。
10. PrintStream是打印输出流，该流把Java语言的内构类型以字符表示形式传送到相应的输出流中，可以以文本的形式浏览。
11. System类是final类，该类不能被继承，也不能创建system类的实例对象。

**RandomAccessFile类**

使用RandomAccessFile类可以读取任意位置数据的文件。RandomAccessFile类既不是输入流类的子类。也不是输出流类的子类。

**过滤器流**

过滤器是为某种目的过滤字节或字符的数据流。基本输入流提供的读取方法，只能用来读取字节或字符。而过滤器流能够读取整数值、双精度值或字符串，但需要一个过滤器类来包装输入流。

**对象的序列化，反序列化**

1. 对象序列化，就是讲Object转换成byte序列，反之叫对象的反序列化
2. 序列化流(ObjectOutputStream)是过滤流----writeObject

反序列化流（ObjectInputStream）----readObject

1. 序列化接口（Serializable）

对象必须实现序列化接口，才能进行序列化，否则将出现异常

这个接口，没有任何方法，只是一个标准

**Servlet**

Servlet是在服务器上运行的小程序。一 个Servlet就是一个Java类，并且可以通过“请求-响应”编程模型来访问的这个驻留在服务器内存里的Servlet程序。

Tomcat容器等级

Tomcat的容器分为四个等级，Servlet的容器管理Context容器，一个Context对应一个Web工程。



编写一个Servlet首先要继承于HttpServlet，重写doGet和doPost方法，然后再web.xml里注册servlet，主要有两个标签：servlet和servlet-mapping

Servlet生命周期

1. 初始化阶段，调用init（）方法
2. 响应客户请求阶段，调用service（）方法。由service（）方法根据提交方式选择执行doGet（）或者doPost（）方法。
3. 终止阶段，调用destroy方法

Servlet生命周期

在下列时刻Servlet容器装载Servlet：

1. Servlet容器启动时自动装载某些Servlet，实现它只需要在web.xml文件中的<Servlet></Servlet>之间添加如下代码：<loadon-startup>1</loadon-statup>数字越小表示优先级别越高
2. 在Servlet容器启动后，客户首次向Servlet发送请求
3. Servlet类文件被更新后，重新装载Servlet



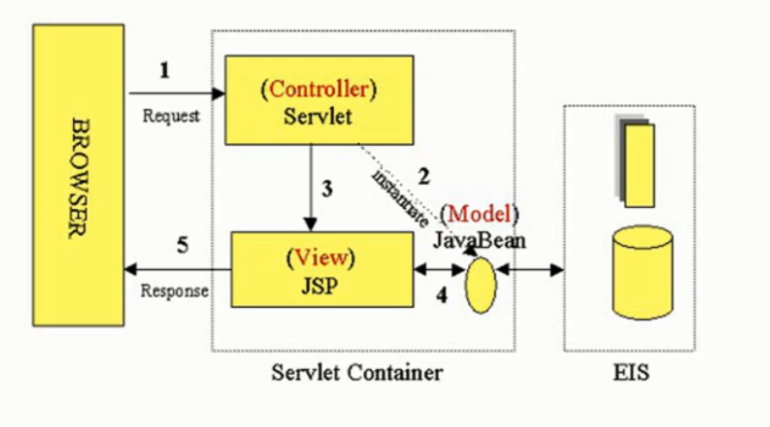
Servlet路径跳转

**Servlet高级**

获取初始化参数

在web.xml中配置Servlet时，可以配置一些初始化参数。而在Servlet中可以通过ServletConfig接口提供的方法来取得这些参数

**MVC**



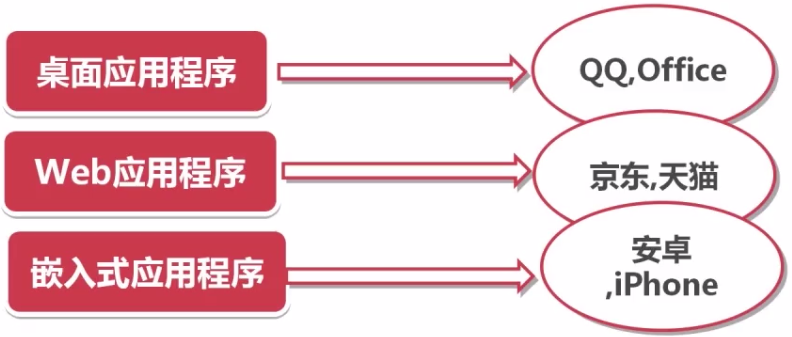
**JavaWeb简介**

Web应用程序

什么是Web应用程序是一种可以通过Web访问的应用程序。Web应用程序的一个最大好处是用户很容易访问应用程序。用户只需要有浏览器即可，不需要再安装其他软件。

为什么要学习Web应用程序

我们说Web应用程序开发，是目前软件开发领域的三大方向之一。



静态网页与动态网页

静态网页

表现形式：网页中的内容是固定，不会更新。

所需技术：HTML，CSS

动态网页

表现形式：网页中的内容通过程序动态显示的，自动更新。学习制作动态网页。

所需技术：HTML,CSS，数据库技术，至少一门高级语言（JAVA,C#,Php），Javascript，XML等。主流的动态网页脚本技术（Jsp/Asp.net/Php）

WEB-INF目录结构

1. WEB-INF是Java的WEB应用的安全目录。所谓安全就是客户端无法访问，只有服务端可以访问的目录
2. web. xml，项目部署文件
3. classes文件夹，用以放置\*.class文件。
4. lib文件夹，用于存放需要的jar包。

**Jsp**

JSP其根本是一个简化的Servlet设计，他实现了在Java当中使用HTML标签。Jsp是一种动态网页技术标准也是Java的标准。JSP与Servlet一样，是在服务器端执行的。

Jsp页面元素构成

**静态内容，指令，表达式，小脚本，声明，注释**

Jsp指令

page指令 通常位于jsp页面的顶端，同一个页面可以有多个page指令。

include指令 将一个外部文件嵌入到当前JSP文件中，同时解析这个页面中的JSP语句。

taglib指令 使用标签库定义新的自定义标签，在JSP页面中启用定制行为

JSP注释

HTML注释

<!-- --> //客户端可见

JSP注释

<%-- --%> //客户端不可见

JSP脚本注释

//单行注释

/\* \*/多行注释

jsp声明

<%! Java代码%>

Jsp表达式

<% =表达式 %>没有分号

JSP页面生命周期

JSP内置对象

out request(请求） response（响应） session application Page pageContext

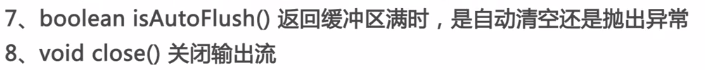
exception config

什么是缓冲区：Buffer，所谓缓冲区就是内存的一块区域用来保存临时数据

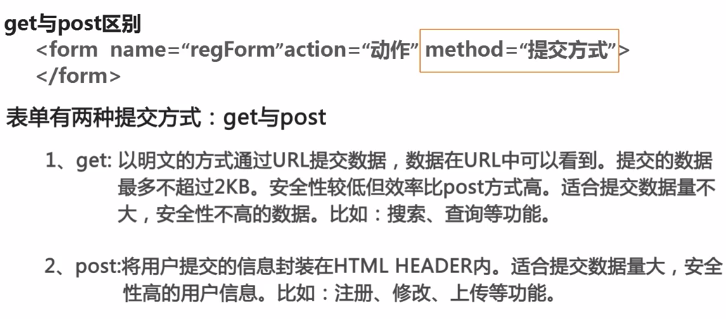
out对象：

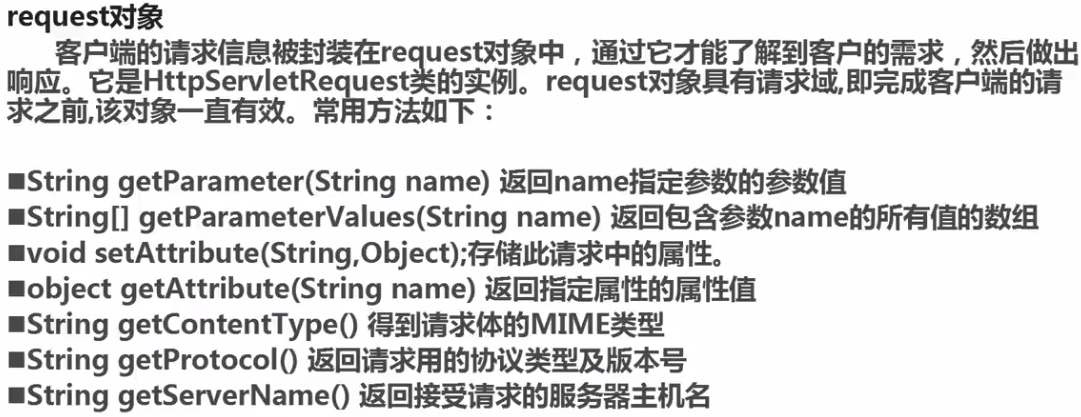
out对象是JspWriter类的实例，是客户端输出内容常用的对象。

1. void println（）向客户端打印字符串
2. void clear（）清楚缓冲区的内容，如果在flush之后调用会抛出异常
3. void clearBuffer（）;清楚缓冲区的内容，如果在flush之后调用不会抛出异常
4. void flush（）将缓冲区内容 输出到客户端
5. int getBufferSize（）返回缓冲区以字节数的大小，如不设缓冲区则为0
6. int getRemaining（）返回缓冲区剩余多少可用

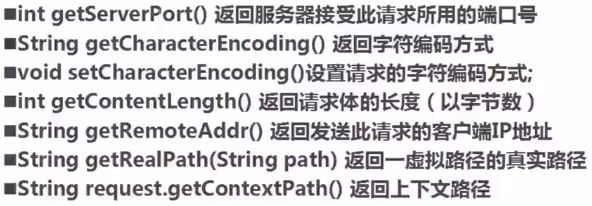


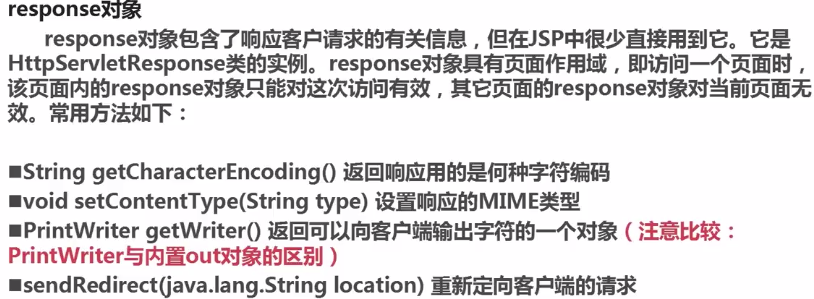
get与post区别





setCharacterEncoding解决中文乱码



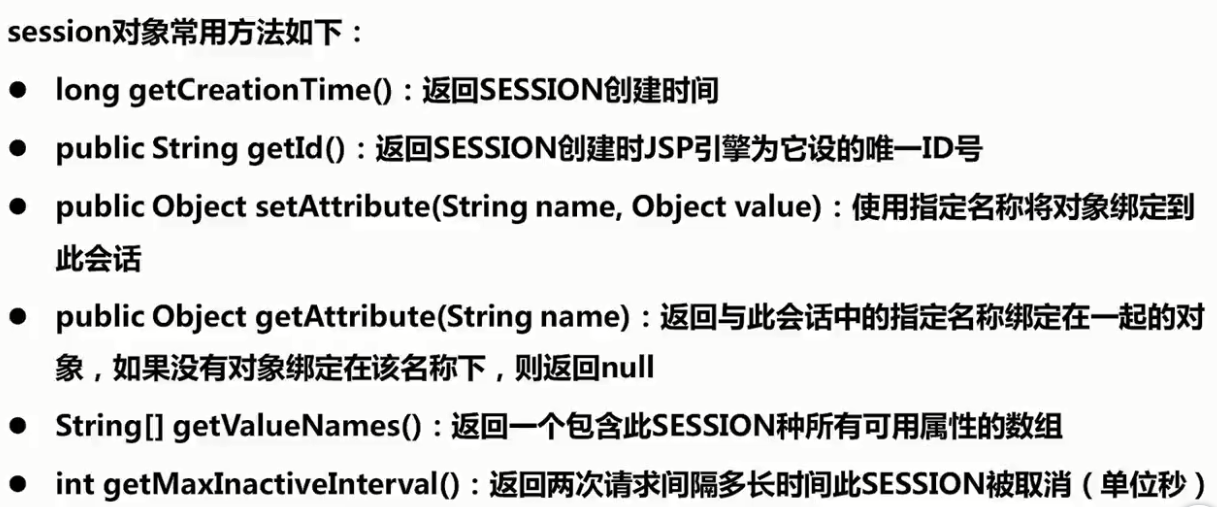




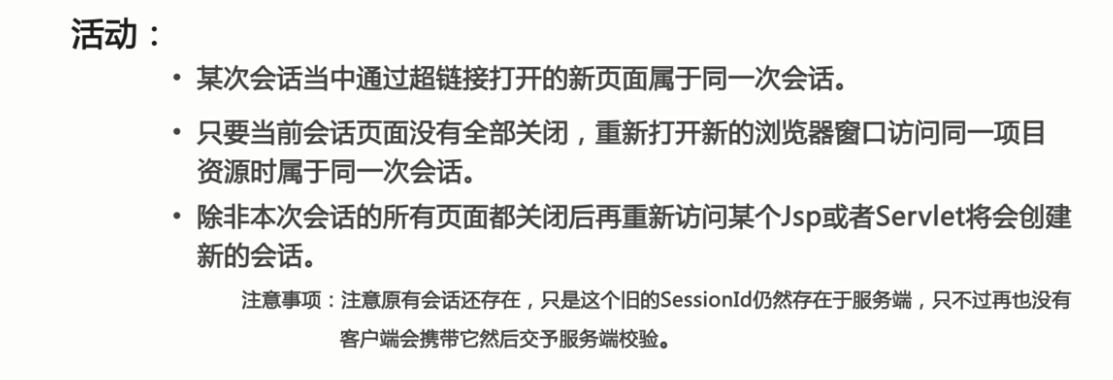
**Session**

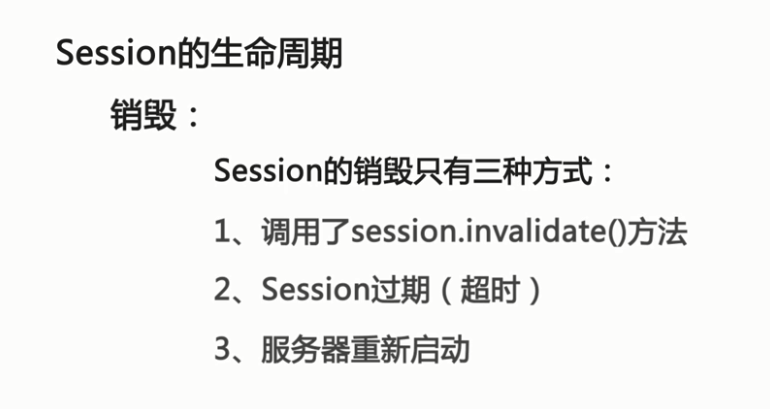
**session实际上是一个特定的时间概念**

**会话是保存在服务器里的**



**Session的生命周期**





**Application对象**

application对象实现了用户间数据的共享，可存放全局变量

application开始于服务器的启动，终止于服务器的关闭

在用户的前后连接或不同用户之间的连接中，可以对application对象的同一属性进行操作。

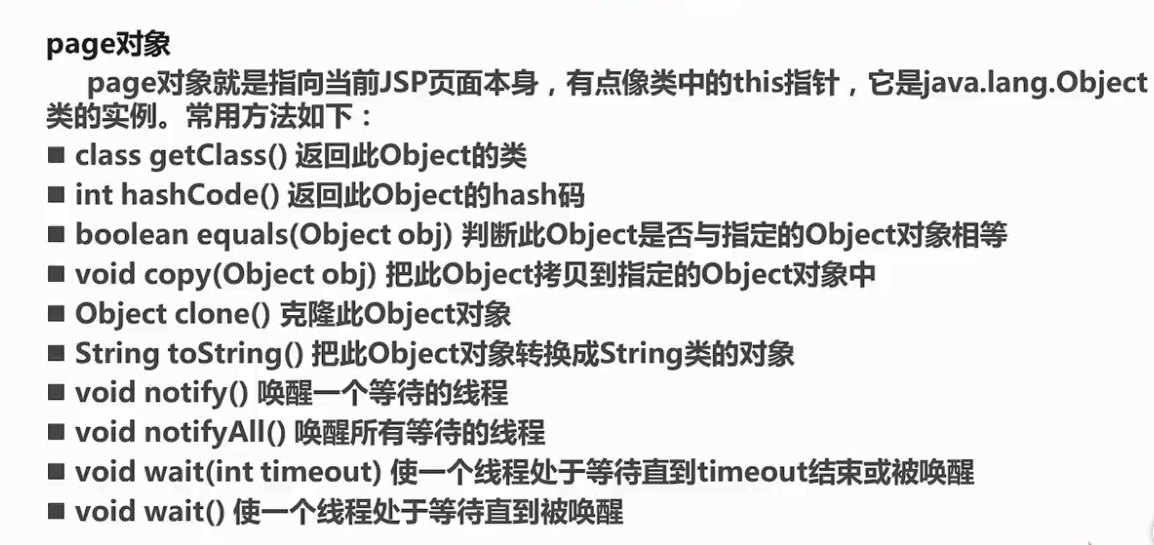
在任何地方对application对象属性的操作，都将影响到其他用户对此的访问。

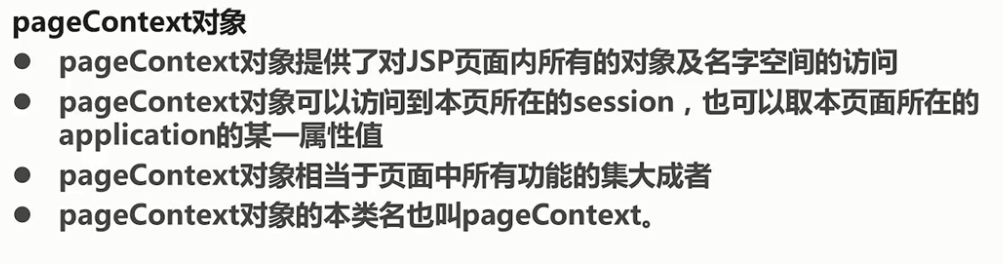
服务器的启动和关闭决定了application对象的生命。

application对象是ServletContext类的实例。

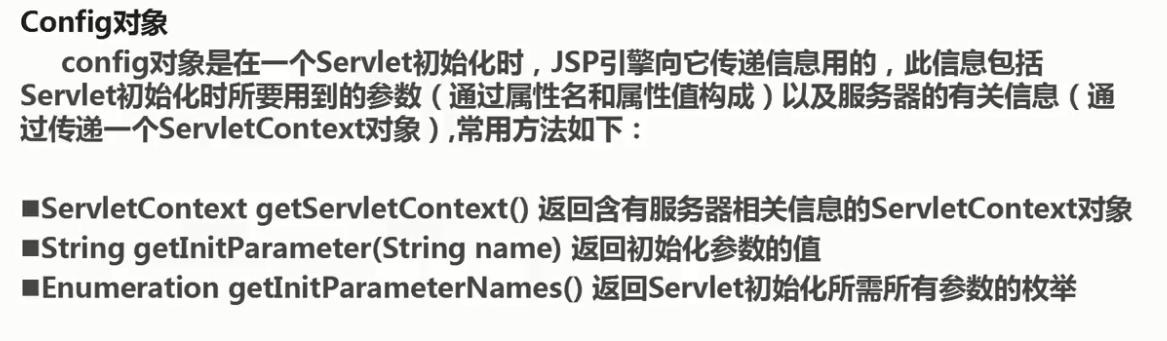
调用application中的属性用Enumeration定义和调用hasmoreElements方法

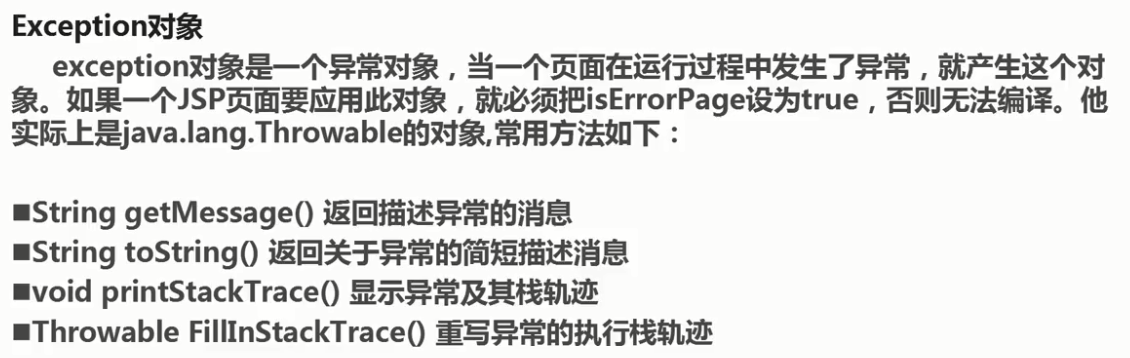
**Page对象**





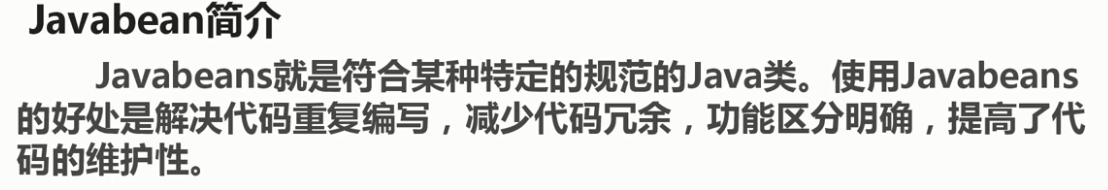


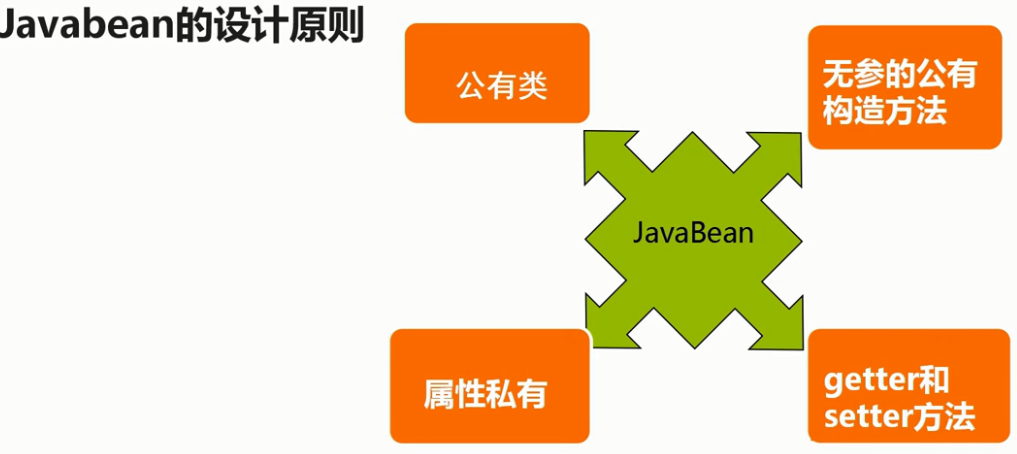


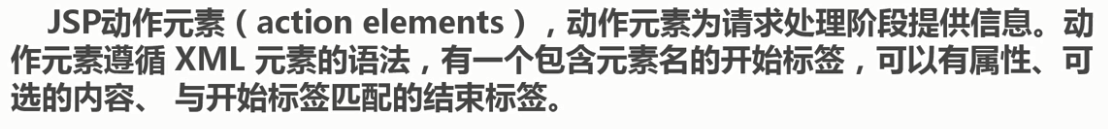


**errorPage=“需要跳转的页面” isErrorPage=“true”;**

**Javabean**

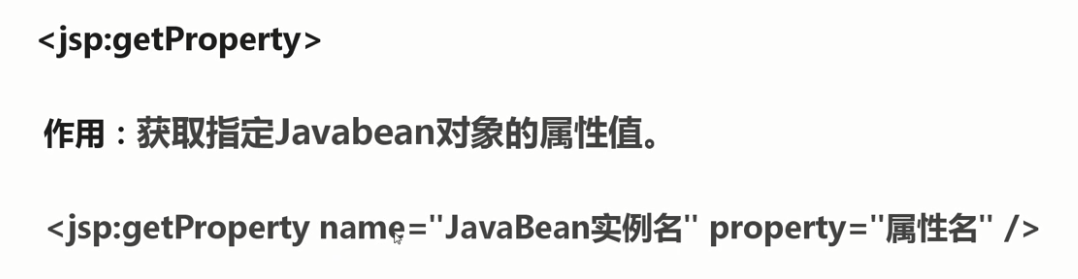






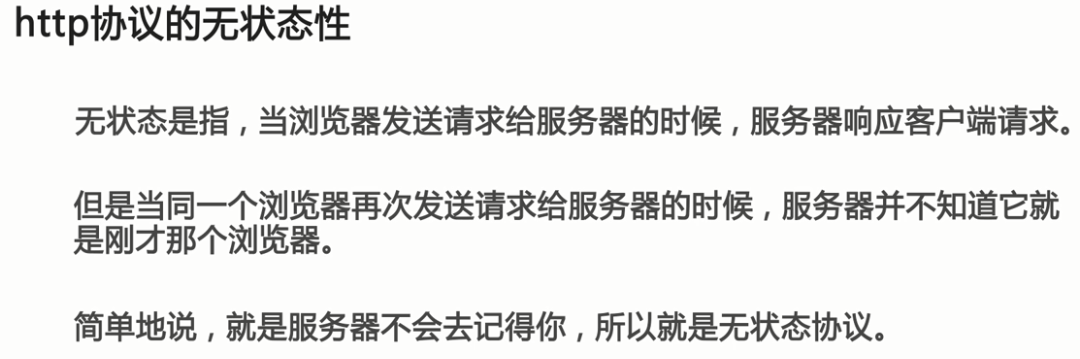






****

**JSP状态管理**



保存用户状态的两大机制

**Session**

**Cookie**

Cookie是Web服务器保存在客户端的一系列文本信息。

Cookie的作用

对特定对象的追踪

保存用户网页浏览记录与习惯

简化登录

安全风险：容易泄露用户信息

Jsp中创建与使用Cookie

创建Cookie对象

Cookie XX = new Cookie（String key，Object Value）

写入Cookie对象

response.addCookie(new Cookie)

读取Cookie对象

Cookie[] cookies = resquest.getCookies();



Session和Cookie的对比



**Include指令与动作**

**JSP三大指令 page、include、taglib**

include指令语法

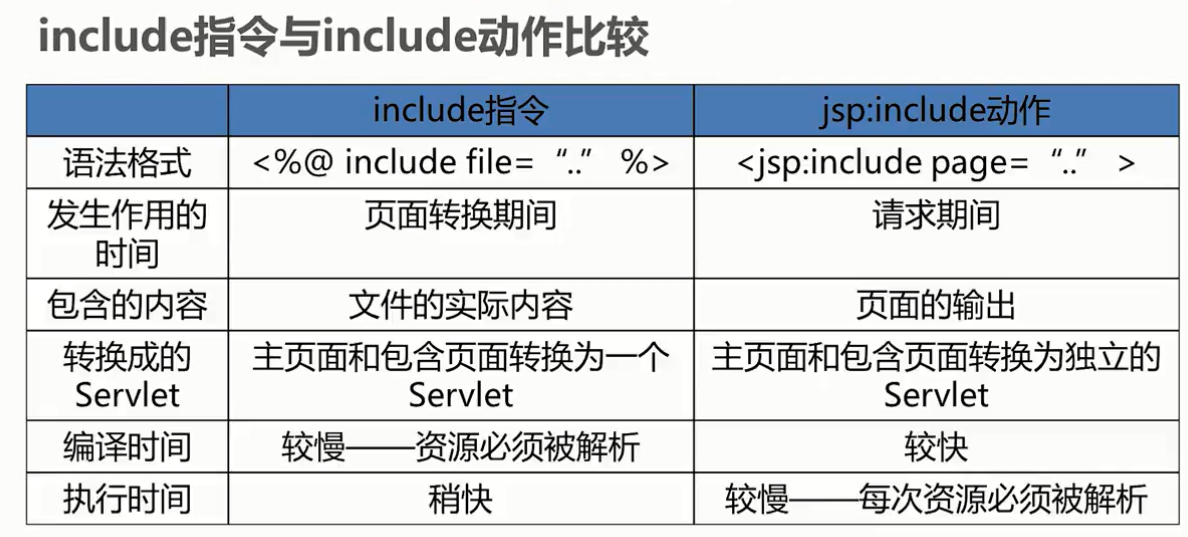
<% @include file=“URL”%>

include指令动作

<jsp:include page=“URL” flush=“true|flase”/>

page 是要包含的页面

flush 被包含的页面是否从缓冲区读取



forward动作

forward语法

<jsp: forward page=“URL”/>

**HTML**

head标签的作用。文档的头部描述了文档的各种属性和信息，包括文档的标题等。

title标签：在<title>和</title>标签之间的文字内容是网页的标题信息，它会出现在浏览器的标题栏中。网页的title标签用于高速用户和搜索引擎这个网页的主要内容是什么，搜索引擎可以通过网页标题，迅速的判断出网页的主题，每个网页的内容都是不同的，每个网页都应该有一个独一无二的title。

在一段话中特别强调某几个文字，用<em>或<strong>,<em>默认用斜体，<strong>默认用粗体

<span>标签是没有语义的，它的作用就是为了设置单独的样式用的。

<q>标签，短文本引用

<blockquote>标签，长文本引用

<br>标签分行显示文本 &nbsp添加一些空格 <hr />标签，添加水平横线

<address>标签，为网页加入地址信息

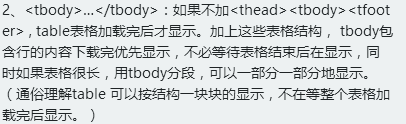
<code>标签，加入一行代码 <pre>标签为网页加入大段代码

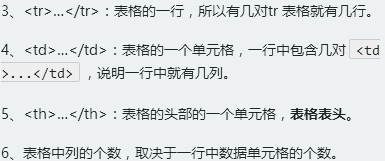
ul添加信息列表 <ul> <li>\*\*\*\*</li> </ul> 信息前为圆点

ol添加信息列表 <ol> <li>\*\*\*\*</li> </ol> 信息前为序号

<div>标签的作用就相当于一个容器，可以把一些独立的逻辑部分划分出来。 给div命名使逻辑更加清晰 <div id=“\*\*”>

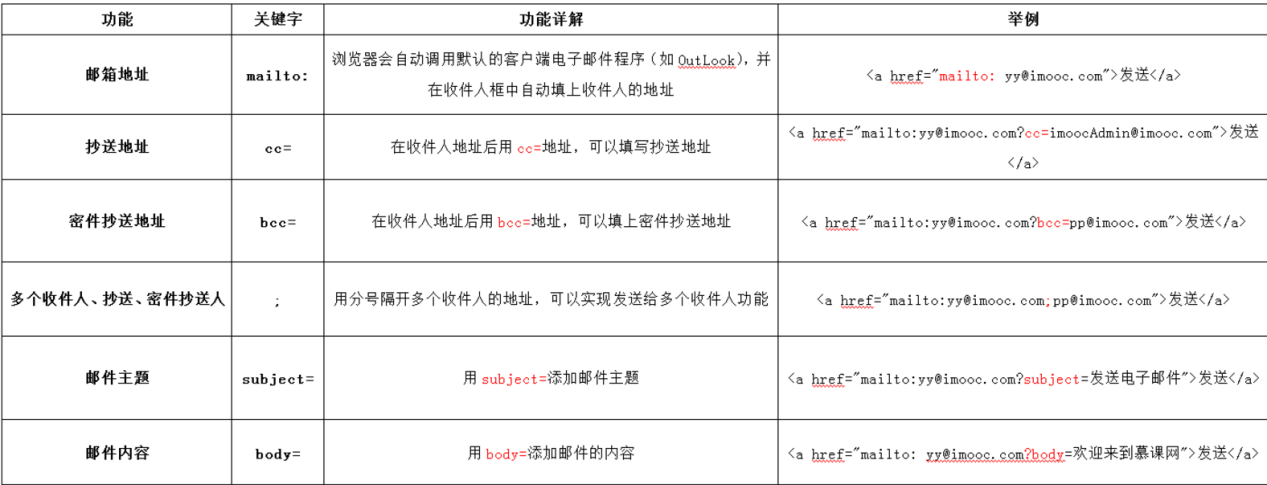
<table>标签，网页上的表格



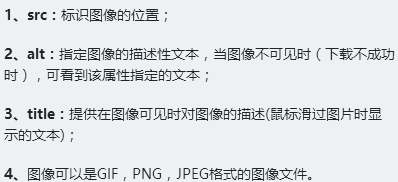


<caption>标签，添加标题，用以描述表格内容，标题的显示位置：表格上方

在新建浏览器窗口中打开链接<a href=“目标网址” target=“\_blank”>\*\*\*\*</a>



<img>标签，为网页插入图片



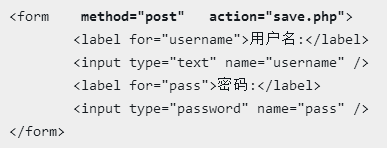
使用表单标签，与用户交互。

<form method=“传送方式” action=“服务器文件”>

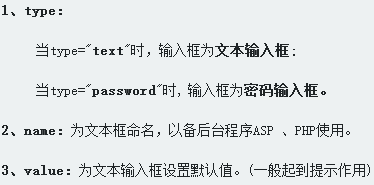
<form> 标签是成对出现的，以<form>开始，以<form>结束。

action 浏览者输入的数据被传送到的地方

method 数据传送的方式（get/post）



文本输入框、密码输入框



文本域，支持多行文本输入

<textarea rows=“行数” cols=“列数”>文本</textarea>

**CSS元素分类**

****常用的块状元素有：****

<div>、<p>、<h1>...<h6>、<ol>、<ul>、<dl>、<table>、<address>、<blockquote> 、<form>

****常用的内联元素有：****

<a>、<span>、<br>、<i>、<em>、<strong>、<label>、<q>、<var>、<cite>、<code>

****常用的内联块状元素有：****

<img>、<input>

****块级元素特点：****

1、每个块级元素都从新的一行开始，并且其后的元素也另起一行。（真霸道，一个块级元素独占一行）

2、元素的高度、宽度、行高以及顶和底边距都可设置。

3、元素宽度在不设置的情况下，是它本身父容器的100%（和父元素的宽度一致），除非设定一个宽度。

内联元素特点：

1、和其他元素都在一行上；

2、元素的高度、宽度及顶部和底部边距不可设置；

3、元素的宽度就是它包含的文字或图片的宽度，不可改变。

在网页中，元素有三种布局模型：  
1、流动模型（Flow）  
2、浮动模型 (Float)  
3、层模型（Layer）

流动（Flow）是默认的网页布局模式。也就是说网页在默认状态下的 HTML 网页元素都是根据流动模型来分布网页内容的。

流动布局模型具有2个比较典型的特征：

第一点，块状元素都会在所处的包含元素内自上而下按顺序垂直延伸分布，因为在默认状态下，块状元素的宽度都为100%。实际上，块状元素都会以行的形式占据位置。

第二点，在流动模型下，内联元素都会在所处的包含元素内从左到右水平分布显示。（内联元素可不像块状元素这么霸道独占一行）

层布局模型就像是图像软件PhotoShop中非常流行的图层编辑功能一样，每个图层能够精确定位操作，但在网页设计领域，由于网页大小的活动性，层布局没能受到热捧。但是在网页上局部使用层布局还是有其方便之处的。

层模型有三种形式：

1、绝对定位(position: absolute)

2、相对定位(position: relative)

3、固定定位(position: fixed)

**使用JSP+servlet技术生成报表**

报表在项目中的地位：1.面向管理层和决策层

1. 充分展现系统数据价值